Gizmox Visual WebGui®
bringing AJAX & Silverlight to enterprises

Visual WebGui

Technology Overview

Rich Internet, Cloud & SaaS Applications (RIA) Platform

Version 1.0.2

By: Itzik Spitzen, VP R&D

Development Department
Gizmox LTD.

2 Hangar St. 2nd floor, POB 21118 Kefar Saba Israel
Tel – 972-9-7673063 Fax- 972-9-7673064 Web site- www.visualwebgui.com

Gizmox Visual WebGui®
bringing AJAX & Silverlight to enterprises

Table of contents

1. Abstract

The following document will explore the basic technology aspects presented by Visual WebGui solution.

Document structure

The first subject “What is Visual WebGui?” provides the initial background on Visual WebGui, its features, benefit and usages scenarios.

The second subject “General Overview” explores some technology aspects of Visual WebGui with deep dive approach on a highlight level.

The third subject “Technical Overview” deep dives into the technological aspects. Each reference explored in this document will contain the following parts:

- *Introduction* – a general description of the explored aspect.
- *Overview* – a technological overview of the aspect
- *Summary* – Summary and further relevant considerations.

2. What is Visual WebGui?

Visual WebGui is a platform for rapid development, quality & secured deployment and easy migration of desktop applications & abilities to the web. It is incorporated with an SDK integral to Microsoft Visual Studio, enabling the most productive, secured and responsive RIA (Rich Internet Applications).

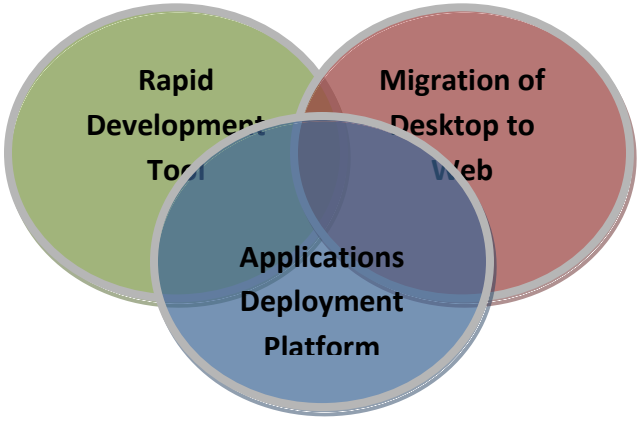

Fig. 1

Server centric architecture

Visual WebGui executes the business logic on the server and virtualizes the UI to its clients (plain browsers). In addition it introduces a unique approach for decoupling the application and the logics from the presentation layer. Having this unique separation between the application and the UI rendering enables support for multiple presentation layers simultaneously and with the same source code (currently available presentation layers are plain browser DHTML, Silverlight and smart client WinForms; in the near future mobile technologies WPF and Flash will be supported as well).

Standard development tool

Visual WebGui is coded using standard .NET languages (C#.NET/VB.NET) and utilizes the productive proven WinForms development paradigm to develop generic web applications including WYSIWYG forms designer.

Empty Client

Visual WebGui presents the unique approach of an “Empty Client” which is a paradigm shift that provides the following benefits:

- Military grade security.
- No code generation; nor at coding time neither at runtime.
- Smallest footprint on the client (~200kb).

- No installation on the client.
- Accessible from any plain browser.

Web / ASP.NET based technology

Visual WebGui replaces the pipeline protocol and creates a new pipeline; however purely based on the standard ASP.NET technology. Visual WebGui uses ASP.NET including its base objects (Server, Session, Application, Request and Response), deployed on standard IIS (no server installation required) and the code is parsed by the standard .NET CLR.

This fact constitutes the following benefits:

- Interactive with any ASP.NET application (including mutual containment)
- Wrap in any ASP.NET control in a click of a button
- Interact with any other web technology
- Reduce risks in terms of infrastructures and use known and proven MS underlying technologies.
- Deployment is as simple as copy & paste

Open RAD

Utilizing WinForms as application development paradigm positions Visual WebGui as a RAD, however, due to the internal structure of Visual WebGui it enables extending the library by adding new controls, editing the look & feel of the UI through the Themes mechanism and customizing the existing set of controls utilizing OOP inheritance.

Based on plain web principles and no proprietary client components provide the freedom to extend the library utilizing web development skills. In the near future, a rapid controls designer will be shipped as an integral part of the development tools enabling visualize customization and adjustments.

3. General Overview

This section will specify the different aspects which will be explored within this document:

Visual WebGui in Microsoft's technologies stack

Technologically Visual WebGui can be best describes as an extension to ASP.NET for application development and deployment.

As such the best way to start over viewing the solution would be by exploring Visual WebGui position in Microsoft's technologies stack.

Command level virtualization

Being a server centric architecture; Visual WebGui presents a unique mechanism of balancing between the server state and the client UI rendering state at any given point of time. This aspect is crucial in the path to understanding the following other aspects:

- Security
- Performance
- Scalability and deployment economy
- Multiple Presentation Layers

Security

Visual WebGui presents the "Empty Client" model, a paradigm shift in which the client downloads a kernel of plain and static code which is responsible for further communication with the server. This concept is secured by design as the client code cannot control the server behavior under any circumstances.

Visual WebGui does not solve the entire issues spectrum of securing your applicative environment, however, by shifting the issue to more comfort zones which are the middleware communication between the client and the server and securing the server, the security problem becomes solvable, controllable and reach military grade easily.

Performance

Being a server centric architecture, Visual WebGui is an immediate "suspect" for being less responsive or for suffering from high latency. This suspicion is far from being true, on the contrary Visual WebGui has proved to be more responsive than pure client side solutions due to the fact that Visual WebGui extremely reduces the CPU usage on the server, optimizes the communication protocol between the client and the server to a degree never realized on web before, optimizes the UI rendering and leverages the client power when it can create a better responsive experience. With

this mechanism Visual WebGui offers an optimal balance of communication between the server and the client.

Scalability and deployment economy

Visual WebGui is fully scalable and redundant across web farms due to a unique capability of enabling serialization of the entire state model into a floating state server (preferably cluster DB based state server).

A single IIS server can server between 200-400 concurrent users and even more since it reduces the CPU usage dramatically.

Multiple Presentation Layers

The outcome of Visual WebGui architecture is a generic object model that is completely separated from UI rendering. This architecture which is often described as decoupled presentation layer provides the ability to render the UI and consume the application practically from any device which can receive and send XML.

The application itself runs on the server and acts on objects containing only metadata and data and the client only renders the UI as reflected from the current application state on the server.

WinForms API Development and Migration

The fact that Visual WebGui flattens web development to a single layer, made it possible to select the most productive and intuitive WYSIWYG development paradigm which is WinForms. Visual WebGui mimics WinForms API in order to provide the entire toolset available for desktop application development including Data-Binding, Layout options (anchoring, docking etc.) and a visual WYSIWYG designer.

Due to the similarity of Visual WebGui API to that of WinForms API, it is quite a straightforward and natural process to transform any native WinForms Application to Visual WebGui and by that provide an application which can be consumed either as a desktop application or a plain web application.

Cloud optimized architecture

Visual WebGui enables the heaviest organization's desktop apps on the cloud with no UI compromises and at no-cost.

Being a highly optimized server centric architecture; Visual WebGui has high value and support the model of cloud computing scenarios in terms of compatibility and optimizations considerations.

Why does it match the cloud?

1. Plain ASP.NET, complete .NET CLR parsed solution.

2. Supports state serialization and seamless scalability.

What are the optimizations for the cloud?

1. Highly optimizes the network usage and lowers CPU usage.
2. Highly optimized state saving and state-server interaction.
3. Eliminates the web limitations for the cloud.

Controls & Themes Design and Extensibility Model

Being pure web architecture, Visual WebGui utilizes the web server and client technologies underneath; therefore, it is possible to create new controls based on the same concepts and set of tools in Visual WebGui.

The various extensibility & customization options will be explored further in this section:

1. Theme designer – enables visual point & click wise editing of themes.
2. Control level designer – enables visual point & click wise editing & creation of new controls (inherited or from scratch)
3. Wrapping in new ASP.NET based controls (i.e. Infragistics, Telerik, deveXpress, ComponentOne etc.)

4. Technical Overview

4.1 Visual WebGui position in Microsoft's technologies stack

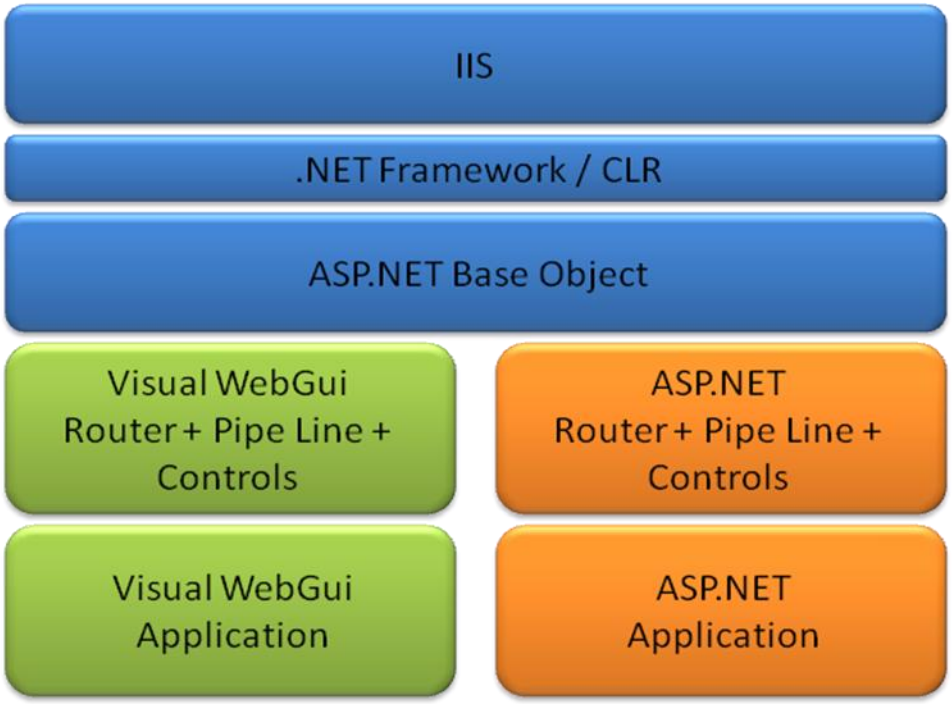

Fig. 2

4.1.1 Introduction

A Visual WebGui project is basically an ASP.NET flavored project type which behaves exactly the same as ASP.NET in terms of coding language and compilation products.

4.1.2 Overview

The compilation product is an assembly accompanied by a web.config configuration file and the runtime result is based on the ASP.NET essential infrastructure:

- *Session* object – In a Visual WebGui application, the Session object plays a very important part. It contains a new container unit defined by Visual WebGui and known as the Context. The Context is the highest object in the hierarchy represented by a Visual WebGui application and is functioning as semi-global scope of one instance of the application.
  One of the channels through which Visual WebGui application can communicate with its ASP.NET environment is the Session object.

- *Application* object – Functions exactly the same as in ASP.NET which means it's the global scope of the application and can be used to consolidate global application data if necessary (although static members can function just as well for this specific cause).

- *Server* object – is accessible from Visual WebGui applications and provides the same services as in standard ASP.NET applications.

- *Response/Request* – are normally overridden by Visual WebGui providing a standard and single pipeline to the server, however, Visual WebGui provides the Gateway mechanism (which is explored more deeply further in this document) to take control of those objects and use them according to the custom needs of the application (for example: retrieving images from database, creating resources on the fly etc).

Mapping requests to objects is done by Visual WebGui Router object which is defined within the httpHandlers section in the web.config file:

```
<system.web>
<httpHandlers>
  <add verb="*" path="*.wgx"
  type="Gizmox.WebGUI.Server.Router,Gizmox.WebGUI.Server,Version=2.
  0.5701.0,Culture=neutral,PublicKeyToken=3de6eb684226c24d" />…
```

Unlike ASP.NET and due to the fact that Visual WebGui uses live state objects on the server, there isn't any actual file which defines a Form object; Forms are object which inherit from Gizmox.WebGUI.Forms.Form object and is mapped by this Visual WebGui Router object to be handled by an instance of an object of the suitable type.

Entry point forms which are called "Applications" are defined within the web.config file and define the set of forms which are browse-able directly:

```
<WebGUI>
 <Applications>
   <Application Code="MainForm" Type="MySample.MainForm, MySample"/>
   <Application Code="Form2" Type="MySample.Form2, MySample"/>
 </Applications>…
```

Visual WebGui Context initialization scenario (as described in figure 3):

1. The client approaches the server for the first time.
2. The IIS server infrastructure discovers that no Session exists for this client and creates an IIS Session.
3. ASP.NET native ISAPI filter takes over the request and creates the basic ASP.NET infrastructure on the server.
4. ASP.NET Visual WebGui HttpHandler definition causes ASP.NET ISAPI filter to hand over the request to Visual WebGui a new Router object.

5. The router detects a “Preload” request and sends back the initial HTML and the kernel resources which are responsible for further communication with the server and UI rendering.

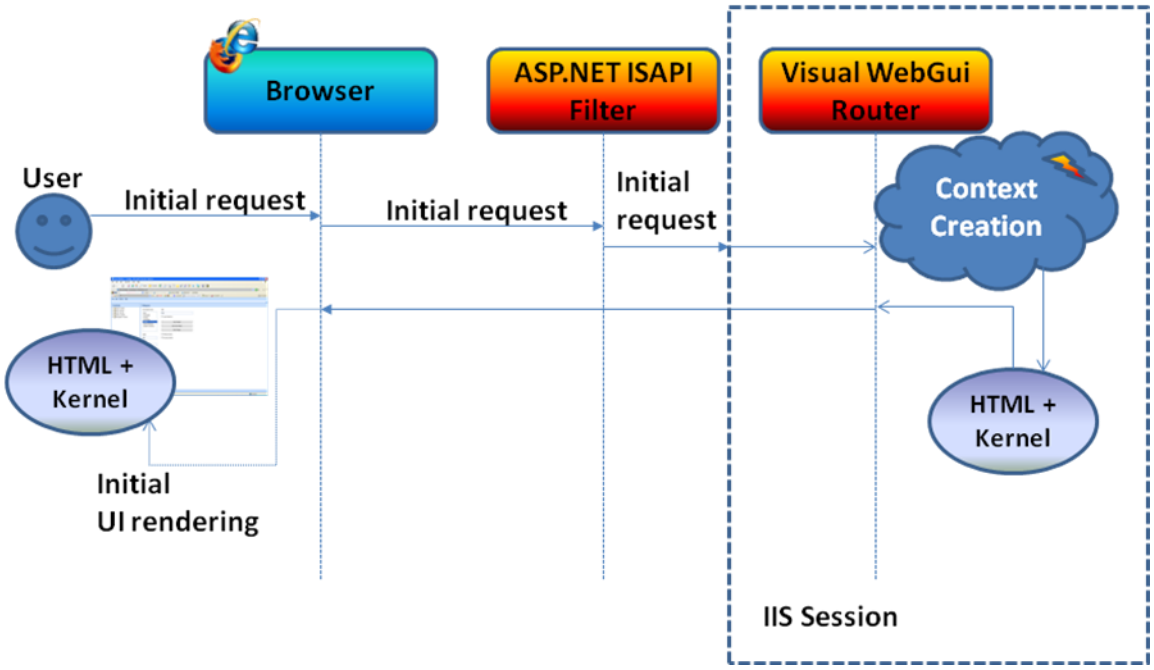

Fig. 3

4.1.3 Summary

Visual WebGui utilizes the IIS and ASP.NET infrastructure and depart from ASP.NET only on the pipeline. The Context object is an application instance global scope and it is a Session resistance further dividing it to a specified scope.

4.2 Command level virtualization

4.2.1 Introduction

Visual WebGui virtualizes the application state from the server to the client using a unique protocol of events and commands.

Think of a bitmap based virtualization solution such as Citrix or Remote-desktop, even though highly optimized it is still transferring a picture; the client plays the minor part of showing a bitmap and replacing it when necessary.

Now, the natural evolving paradigm would be having the client "understand" better in terms of UI yet having the application perform the business logic and manage sensitive data on the server. Visual WebGui is able to utilize the client strength and at the same time leverage the server's accessibility to data and security.

4.2.2 Overview

Visual WebGui pipeline balance flow

- The client gets the kernel code only once – then it's cached and reused.
- Events are sent to the server only when necessary
  - Events that are not handled by the server are queued on the client.
  - A mechanism of unique events prevent sending multiple events which are causing a single result (for example, only the last text within a textbox is sent to the server as long as the "KeyPress" event is not handled by the application)
  - Event and command sizes are limited to 1kb and never exceed the HTTP packet size.
- The server gets the last event queue, and then it executes the applications logic according to the events which are executed chronologically.
- As a result, the current application instance might change the UI accordingly then minimal commands are sent back to the client kernel in order to cause the client to render the relevant parts of the UI and add/remove/change data.

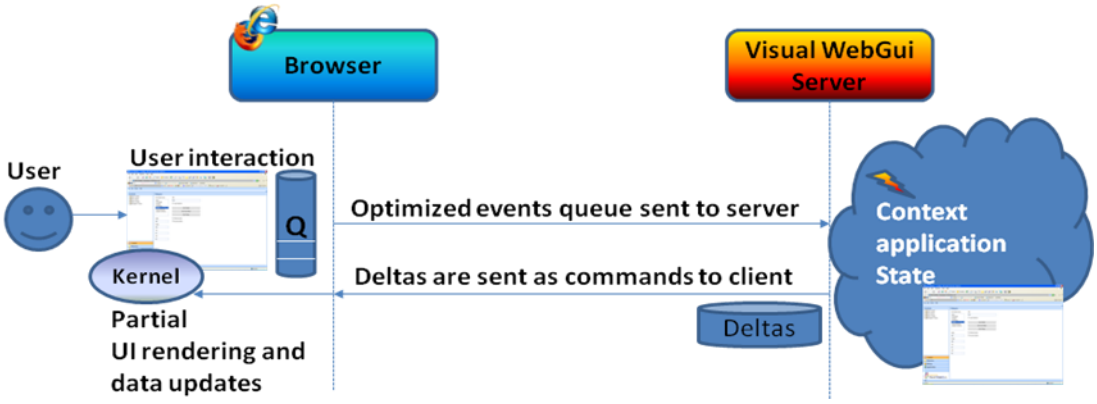

Fig. 4

Client kernel structure

- XSLT - Visual WebGui uses client rendering capabilities as offered by the target device, therefore, when using a plain HTML device (i.e. web browsers) the best choice was to utilize XSL transformation in order to perform client side HTML rendering. Each of the Visual WebGui controls and general layout behaviors and capabilities such as Dialogs, Docking, and Anchoring etc. are represented in XSLT.

  When sent to the client, one large XSLT is created of all the XSLT parts of controls and general purpose by two mechanisms which are part of Visual WebGui core library called "Collectors" and "Compiler". The static XSLT is then cached on the client and enables any further render of the UI (either partial or full UI elements rendering).

- JavaScript – as mentioned before, the client behavior in the HTML presentation layer is performed by JavaScript code. Each of the controls in Visual WebGui has its own JavaScript piece of code and also many general behaviors such as:
  - Dialog layers management
  - AJAX communication with the server
  - Drag & Drop mechanisms
  - General events handling
  - Events optimizing and queuing

  The "Collectors" mechanism is collecting all the JavaScript files into one large file and then the "Compiler" rearranges and compresses this part of the kernel in order to create a highly optimized kernel part.

  This compressing mechanism can be turned off in case we need to debug client side scripting, this is done by switching the "DisableObscuring_Switch" on (0-Off, 1-On):

```
<system.diagnostics>
     <switches>
     <!--
          0 - Disabled
          1 - Enabled
          -->
        <add name="VWG_DisableObscuringSwitch" value="0" />…
```

  Note: it is strongly recommended to switch this switch back on when deploying the application.

- CSS – is the part which is responsible for general styling and exists in most of the controls and in some general scope styling mechanisms.

Static Resources

The static resources mechanism enables the system to cache the entire set of resources (JavaScript, CSS, XSLT and any other static images or data of any kind) of the application as static cached files within the web server's directory.

Switching Static Resources to "On" can be done either through the web.config file or through the project properties and results in caching of the entire resources and further optimize those resources at runtime.

```
<WebGUI>
        <StaticResources Mode="On"/>…
```

Note: it is strongly recommended to develop with this flag off and deploy with this flag on.

Private Version

The private version is a value which helps Visual WebGui decides whether it should grab new resources from the server dynamically or it can use the old cached ones. When upgrading a version successfully, the internal cache version is changed within the core library automatically; therefore, this value should not change in this case. However, when changing resources as part of the development process (I.e. creating themes or custom controls) this value should be advanced whenever a change is made in order to cause Visual WebGui to retrieve a new version of resources and avoid using old cached files.

Changing Private Version value can be done either through the web.config file or through the project properties:

```
<WebGUI>
    <PrivateVersion Value="2"/>…
```

Icons Preloading

As part of the general optimization of resource management, Visual WebGui offers a mechanism for preloading icons when the client first approaches the server.

Switching Icons Preloading to "On" can be done either through the web.config file or through the project properties and results in loading and caching icons on the client machine when it first approaches the application.

```
<WebGUI>
    <IconsPreloading Value="On"/>…
```

Note: using this flag is application dependent; in case the application contains a very large number of icons and many of them are inaccessible for some users, there is no point in preloading all of them

since some of them are not used. In any other case, switching the Icons Preloading to "On" will cause a slight latency on first access, however, will result in better performance from this point on since images are now cached on the client.

Inline Windows

Visual WebGui implements 2 different approaches for windows handling:

1. Non-inline windows – Internet Explorer (IE) dialogs based – which are separate windows of the Internet Explorer.
2. Inline windows – dialogs which are drawn using floating div elements within the client area of the browser.

When using non-IE browser, the default behavior will always be Inline-Windows, and with IE the default would be Non-inline windows, unless configured otherwise:

```
<WebGUI>
    <InlineWindows Value="On"/>…
```

Setting inline windows switch to "On", either through the web.config file or through the project properties forces the use of Inline-Windows in IE as well.

Behaviors differences:

Naturally, inline windows are limited to the client area of the browser, therefore, they do not appear in the task bar when minimized or managed as normal OS windows. Except for that there should be no difference in using those windows type.

Inline-Windows are fully customizable in terms of looks and behaviors as they are entirely drawn and controlled by the client code and not affected by any browser dialog management.

G-Zip Compression

In addition to the internal compressed mode of the client kernel code, a G-Zip compression mode can be applied either through the web.config file or through the project properties:

```
<WebGUI>
    <GZipCompression Value="On"/>…
```

4.2.3 Summary

Visual WebGui Virtualization model: the static client kernel and the server are constantly balancing each other. The client sends events to the server and in return, the server sends update commands back to the client. The client is responsible to re-render the UI when needed.

Kernel parts are assembled of plain HTTP formats: XSLT, JavaScript & CSS and are compressed to the minimal size of ~200kb.

There are some customizable switches which are tuning this process, the cache level and enables optimizing for the various usages.

4.3 Security

4.3.1 Introduction

Using today's offered technologies and more specifically .NET, we are offered many great solutions and methodologies which help us to understand how and using which tools we can secure our applications.

The most painful security issues aren't in using standard security solutions to secure our server's farm or to secure the messages sent between the client and the server. Those security tasks can be quickly and efficiently achieved by using today's firewall capabilities and other secured server's farm solutions and by securing the transferred data using HTTPS, WCF and other great solutions.

The most problematic issues today which gets worst when it comes to thick clients as fat AJAX clients, Flash/Flex and Silverlight based clients are that the more broad and accessible the system becomes the less we can control or even know who are our clients whereas thick clients hold sensitive data which is accessible to those clients.

Being a paradigm shift in form of Empty Client, Visual WebGui clears-up entirely those issues due to the fact that nothing except for UI commands and one static kernel is downloaded to the client. This means that:

1. No sensitive/hidden data is sent to the client. Neither the infrastructure nor the developers can perform security violations by sending sensitive data to client machines.

2. The server exclusively handles interaction with data and other services.

3. The client is responsible to render UI and send events to the server; in any case it can never control the server's behavior.

4. Each client request is validated by the single layer server code which constitutes a single point access control center.

Standard web applications:

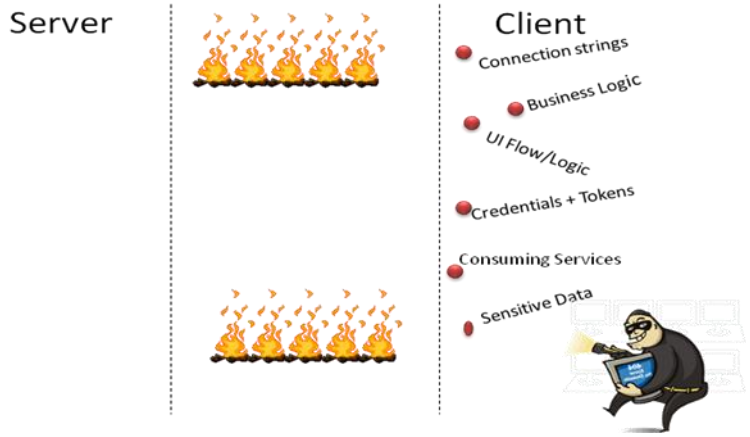

Fig. 5

With Visual WebGui:

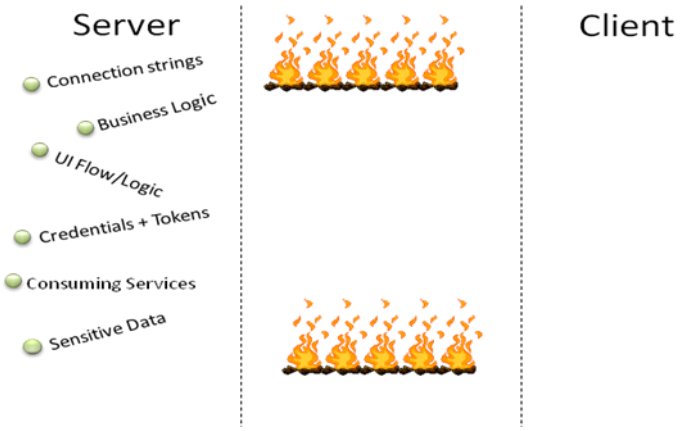

Fig. 6

4.3.2 Overview

Without diving into details, the Visual WebGui security model should be quite clear from the introduction part alone. The only issue that needs further clarification is a key claim which was specified above the fact that Visual WebGui client cannot change the server behavior whatsoever.

The following flow which describes a Visual WebGui application explains why the key-claim is true:

Flow Step 1: The first time the client approaches the server it downloads a small amount of kernel code which is constructed of:

a. *JavaScript*– responsible for the client behaviors and communication with the server.

b. *XSLT* – responsible for the UI layout including the HTML rendering of the entire set of controls.
c. *CSS* – responsible for UI styling

The kernel is sent in a compressed mode and weights about 200kb. Furthermore, it is cached on the client and on the server statically and will never change from this point on.

Security aspects: no code generation at runtime, the kernel is well known and static.

Flow Step 2: The client renders the basic HTML to the screen and from that point on it acts like a smart AJAX client which consumes a UI service from the server only.

Security aspects: only UI essential data is sent to the client, no applicative or sensitive data.

Flow Step 3: Highly optimized events are sent to the server whenever a client performs a set of action that should result in executing server code. Events metadata are constructed of UI object Id and the action performed.

Security aspects: events are reflecting UI behavior and never applicative logic which is uniquely handled by the server.

Flow Step 4: The server executes the event handler and sends back highly optimized UI instructions to the client. The instructions are reflecting the deltas of changes between the last balanced state and the new state.

Security aspects: server instructions are reflecting UI changes and presented data changes, however, will never contain hidden applicative logic or data which is uniquely kept and handled by the server.

Flow Step 5: The client accepts the UI changes instructions and re-renders the parts which have changed according to the last server execution of logics.

Security aspects: the client is responsible to render UI and that is the only aspect which is affected by application logics.

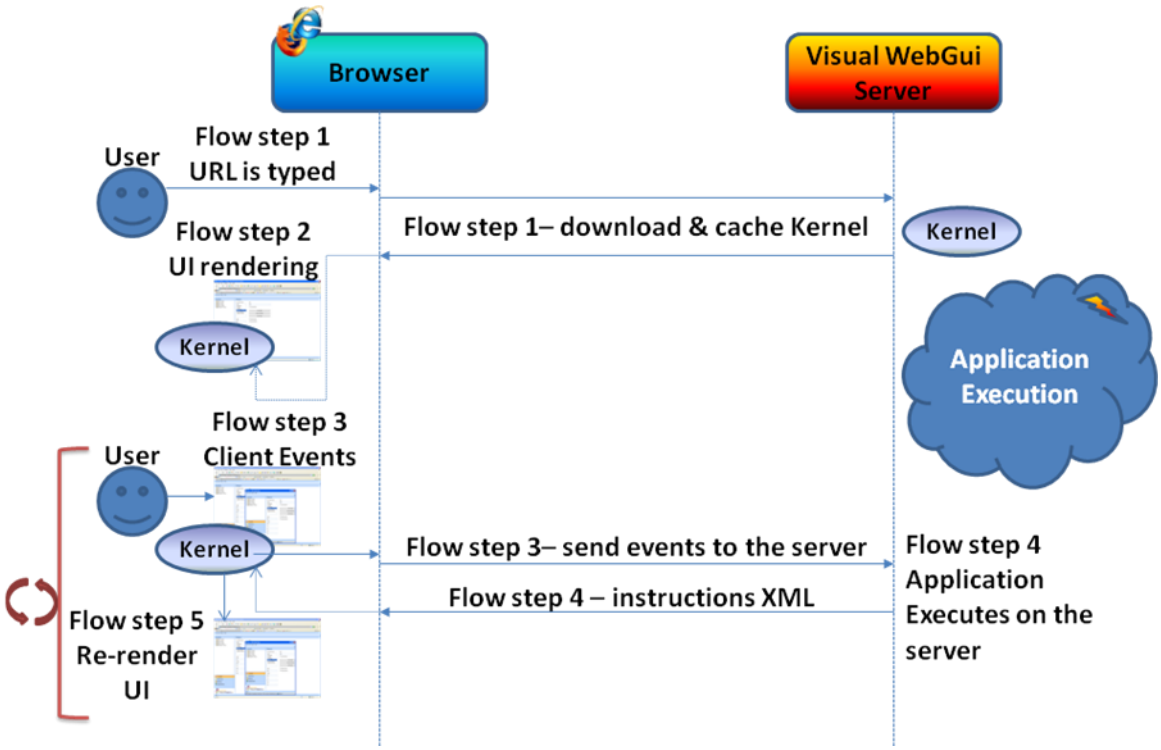

Fig. 7

4.3.3 Summary

1. Client security-holes which are commonly created by either applicative or sensitive data which is kept on the client or even simply sent to the client are impossible by design (as illustrated in figures 5 and 6).

2. Client scripting cannot control the server behavior as "by design", simply because the responsibilities of the client are limited to:

    a. Render the UI at the control level – meaning that utilizing the XSLT, the client kernel can render:
        - The entire screen – this happens only when the UI is entirely replaced.
        - Specific control (thoroughly) – this happens when the control cannot be partially drawn to adjust to its new given state.
        - Control Part – this is the most common scenario, in this case only the part of the control which has changed is drawn.

    This responsibility is pure client side and cannot affect any server behavior. The only server's requests which can be caused by such client action are when the rendering of

whatever is rendered items require resources (i.e. images, or dynamic data). Those requests are uniquely controlled by the server code.

b. Send client events to the server (yet the server has the freedom to decide which are valid events and parameters according to the current user's credentials)
   - Those are predefined events which are exclusively handled by predefined handlers and can never affect other parts of the code or change the server behaviors.
   - Any non-formatted data which will be added to those requests will be filtered by the server and might invalidate the entire request.
   - Replaying requests does not affect the server behavior due to a unique timestamp mechanism.

3. The server centric , event-driven design results in an enterprise-level security & very high level of fine-grained control over precisely what the user is seeing - all using one programming language - standard .NET (C#, VB, etc.)

4. Visual WebGui does not imply to present an ultimate solution for all the security issues, however, through the Visual WebGui communication protocol it will be impossible to hack a web application. This means that assuming https and/or any other incredible secured communication solutions (i.e. WCF) are used to secure the HTTP communication and that the OS and DB are safe on the server side, Visual WebGui application is thoroughly safe.

4.4 Performance

4.4.1 Introduction

Server centric architecture is mostly examined performance wise by four major factors:

1. The amount of data sent and received from the client to the server and vice versa.
2. The intensively of which information transportation is required between the client and the server.
3. The server's service capacity; in other words the count of users which can be intensively served (often called concurrent users) by one server.
4. The ability to leverage client machine's power in order to increase responsiveness.

Visual WebGui attends those 4 major issues by design; a short reference of the architecture highlights which affect the 4 above:

1. Visual WebGui uniquely optimizes the protocol in a way that most of the requests and responses are less than 1kb.

2. The client is always the initiator of requests to the server as in standard web applications; the events mechanism is highly optimized and results in the minimal interaction on a server centric application run.
3. Having a full state of application on the server, Visual WebGui dramatically lowers the amount of construction and disposals of objects; this fact alone reduces the CPU usage dramatically and enables a single server to serve between 200-400 concurrent users.
4. Visual WebGui leverages the client's power in order to render the UI, create a responsive and dynamic UI and perform client-client actions which do not require interaction with the server.

4.4.2 Overview

Figure 8 illustrates the runtime paradigm which is presented by Visual WebGui.

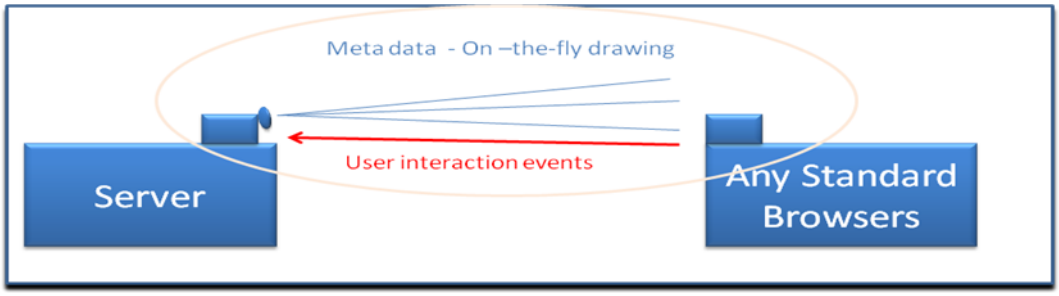

Fig. 8

The following parts of the runtime model of Visual WebGui directly affect the runtime performance of Visual WebGui applications:

- UI Virtualization is performed by transferring highly optimized commands and events metadata from the client and from the server, for example, the following negotiation represents the scenario of clicking a button and as a result, opening a message-box with some text, 3 buttons (Yes, No, Cancel), a question icon and focus on the OK button:

AJAX Request:

```
- <ES LR="633704779264797648">
    <E SR="1" TP="GotFocus" />
    <E SR="1" TP="Click" X="43" Y="12" />
  </ES>
```

The above request contains the absolutely minimal data required in order to update the server with what happened on the client:
LR – Request identification including timestamp and internal data required for further optimization mechanisms.
SR – Registered control Id.
TP – event type → GotFocus – tells the server that the button got focus.

TP – event type → Click – tells the server that the button was clicked on coordinates (43, 12).

AJAX Response:

```
<WG:R LR="633704779298387528" AF="10" FC="8" xmlns:WC="wgcontrols"
mlns:WG="http://www.gizmox.com/webgui">

- <WG:F Id="2">
 - <WG:F H="106" W="410" RD="1" TX="Confirm" TP="ModalWindow"
 FBS="3" FSP="2" WS="0" S="0" FCB="1" Id="10" F="1" PA="10">
  - <WC:TLP Id="6" TI="1" F="1" D="B" H="24" TX="">
          <TLC W="50%" />
          <TLC W="76px" />
          <TLC W="6px" />
          <TLC W="76px" />
          <TLC W="6px" />
          <TLC W="76px" />
          <TLC W="50%" />
          <TLR H="24px" />
          <WC:B Id="9" E="1" RS="0" CS="5" RSP="1" CSP="1" TI="3" F="1" TX="Cancel"
          TA="MiddleCenter" TIR="0" L="0" T="0" H="23" W="75" A="LT" />
          <WC:B Id="8" E="1" RS="0" CS="3" RSP="1" CSP="1" TI="2" F="1" TX="No"
          TA="MiddleCenter" TIR="0" L="0" T="0" H="23" W="75" A="LT" />
          <WC:B Id="7" E="1" RS="0" CS="1" RSP="1" CSP="1" TI="1" F="1" TX="Yes"
          TA="MiddleCenter" TIR="0" L="0" T="0" H="23" W="75" A="LT" />
    </WC:TLP>
 - <WC:P Id="5" TI="2" F="1" TX="" D="L" W="50">
          <WC:PBX Id="3" TI="-1" F="1" L="0" T="0" H="32" W="32" A="LT" IMS="0"
          IM="Skins.Question.GIF.wgx" />
    </WC:P>
   <WC:L Id="4" TI="3" F="1" SA="0" TX="Are you sure?" TA="TopLeft" D="F" />
   </WG:F>
 </WG:F>
- <CMDS>
    <IM MM="Controls_Focus" ARG0="8" />
 </CMDS>
</WG:R>
```

The above response contains thorough instructions for the client kernel of what should be rendered as a result of clicking that button.

In the above sample we may emphasize few important elements:

- *WG:F* – Stands for "form". It can be observed that the xml contains 2 forms – representing the main form and another open form inside (which is the message box form).
- *WC:TLP* – Stands for table layout panel control which is a very important UI element which is responsible in this case to instruct the client to place a UI element which will create the reflected message-box layout.
- *WC:B* – Stands for a button control with all of its non-default properties (such as text, location and size).

- *WC:PXB* – Stands for a picture-box control containing the image (the actual picture resource will be grabbed through Visual WebGui's resource management right after this response will be processed by the client kernel and then cached on the client)
- *CMDS* – Stands for command which should be executed after rendering the UI, in this case the command is Control_Focus which will result in focusing the "No" button in this case.

Any other UI state balancing option or virtualization would have transferred either a bit map or much less optimized state, this is exactly the reason why Visual WebGui is the undoubted winner in a comparison performed by Mr. Wiktor Zychla, Microsoft MVP and performance specialist (as demonstrated in figure 9 bellow).

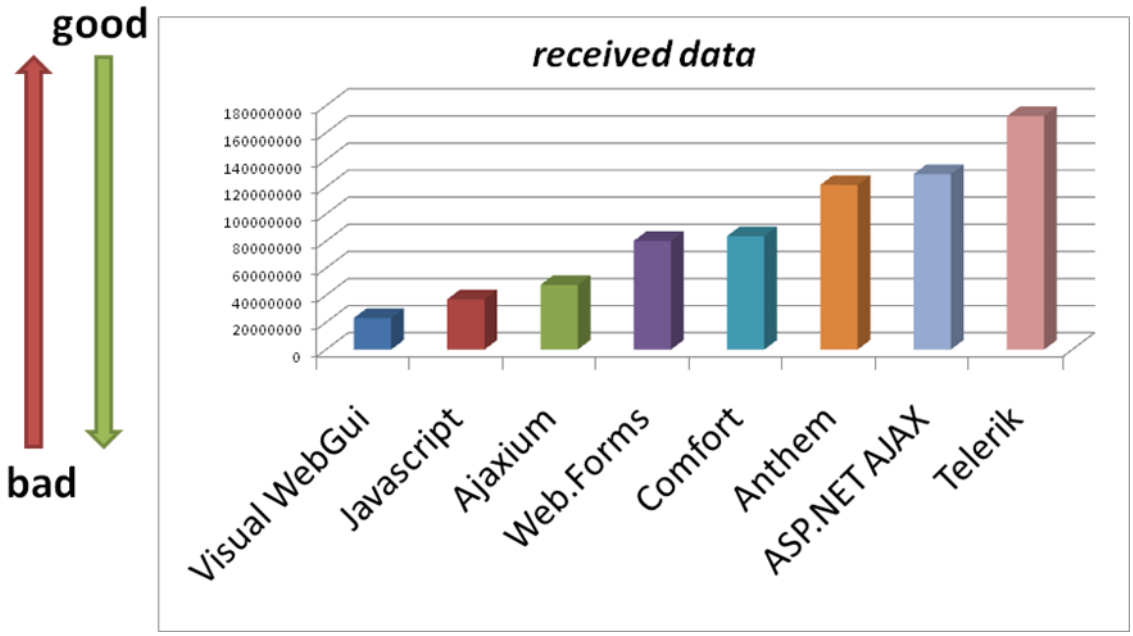

Fig. 9

- Visual WebGui server holds the UI updated state at all times (up until disconnection); The state contains the currently relevant UI tree of controls, for example, if the screen currently shows a Form with a TreeView, a ListView and a few Buttons, the controls tree will look something like that:
    - VWG Application Context
        - Form (form-name, etc)
            - TreeView (text, location etc)
                - TreeNode1 (text, image etc)
                - TreeNode2 -"-
                    - TreeNode2_FirstChild

                    …

- ListView (location, text etc)
  - ListViewColumns Collection
    - ListViewColumn1 (text, width etc)
    - ListViewColumn1 (text, width etc)

    …
  - ListViewItems Collection
    - ListViewItem1
      - ListViewSubItem1 (text etc)
      - ListViewSubItem2 (text etc)

      …
- Button1 (location, text etc)
- Button2 (location, text etc)

Note: except for the controls tree, when developed correctly using all best practices of web development, there should not be allot more than that in the application context.

As opposed to any stateless or semi stateless web applications such as ASP.NET, when the client sends events, the state already exists within the session as described above, and it is ready for executing any application logic on it. This fact dramatically reduces the amount of memory allocations, objects initializations, object disposals and garbage collector activity. This fact lowers the CPU usage accordingly.

The fact that the CPU usage is very low directly reflects the capacity of the web server and its ability to fast serve all the requests. As shown in figure 10, compared to other AJAX frameworks and even plain JavaScript, Visual WebGui serves larger number of users faster (the comparison was performed by Mr. Wiktor Zychla, Microsoft MVP and performance specialist).

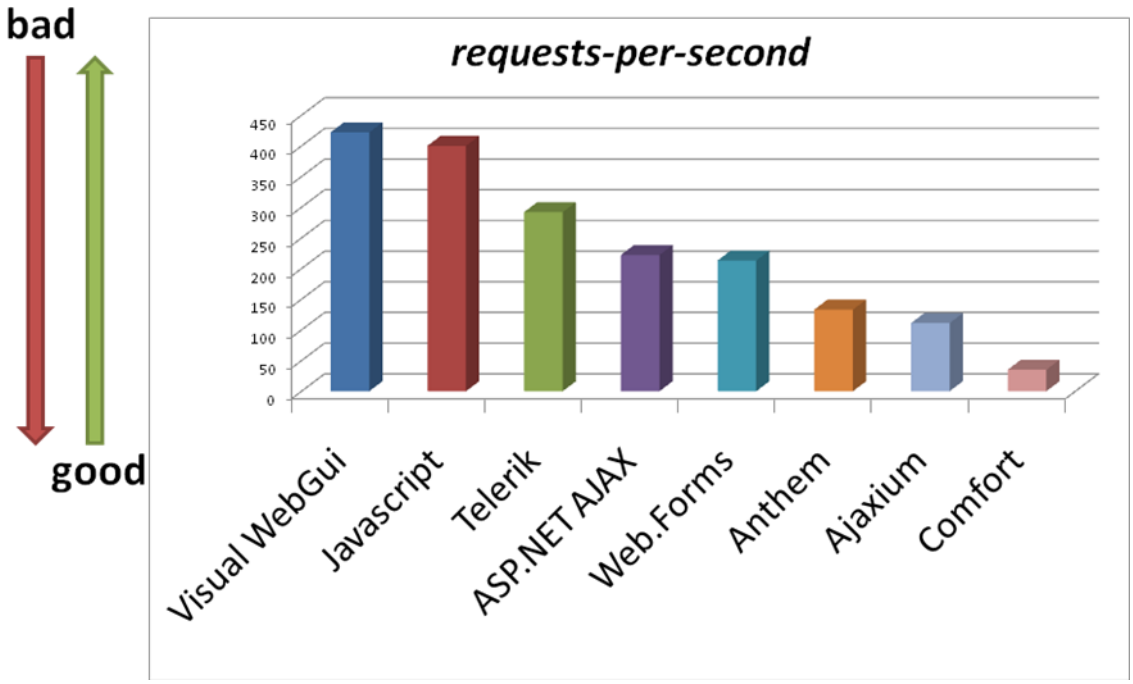

Fig. 10

- Client behaviors and client-client actions are uniquely performed by the client:
    - Behaviors such as scrolling, state images and styles (such as hover) etc are handled uniquely by the client.
    - Critical events are those events that when they occur they should immediately be raised to the server; any non-critical event (majority of the events are non critical), are handled uniquely by the client in places where the developer did not choose to perform any logic in response. In order to keep the server state balanced, those events are queued and sent the next time any other event is raised to the server.
    - Unique events are queued only once, for example ListView selected index changed will be transferred to the server only once (the last selected index) even if the client changed his selection multiple times (unless the developer has chosen to handle this event on the server).
    - The option of performing client to client invocation enables the developer to apply application behaviors on the client without having to send them back to the server. For example, editing text in a RichTextEditor control, enables applying text styles (such as bold, italic and changing font) on the client side without raising events to the server and only the final result is transferred to the server.

The optimizations above helps Visual WebGui leverage the client machine's power and present a responsive and rich application.

4.4.3 Summary

Although Visual WebGui is a server centric architecture it optimizes the communication and the balance between client and server responsibilities and provides low latency and excellent performance. In highlights level, the following factors are producing this outcome:

- Low CPU usage on the server side low negotiation establishment time due to the existence of valid context at all times.
- Highly optimized communication protocol based on compressed deltas metadata and minimal commands.
- Leveraging the client machine power to minimize communication and throughput.

4.5 Scalability and deployment economy

4.5.1 Introduction

One of the important enterprise tests of technology qualification is scalability. The fact that Visual WebGui is a server centric architecture makes it far more curtail to explain why and how Visual WebGui solutions are scaled.

4.5.2 Overview

When approaching to horizontally scale state-full web applications, the following 3 major load balancing options are available which are intended to increase the availability and the concurrent capacity to serve large number of users:

- STATIC LOAD BALANCING: IP/Machine sticky – each server is responsible to serve a pre defined group of machines.
    - Pros:

- There is no need to plan the development process to support this approach or to have any sort of specific server side support.

  - Cons:
    - No redundancy; upon server crash a group of workstations will not be able to continue working until the configuration is manually changed.
    - Can easily create a non balanced load on the servers.
    - Increasing or decreasing the number of clients, requires re-configuring the load balancer manually.

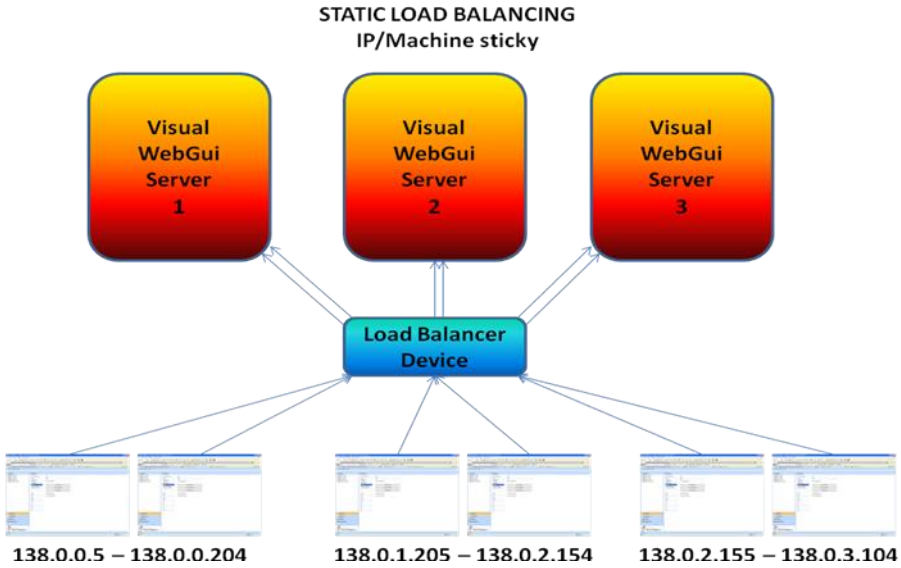

Fig. 11

- DYNAMIC SESSION BASED: Session sticky – the static link between a workstation and a server is done upon connection establishment; in this option, the load balancing device selects the most available server and connects the client to it. Once the connection is established, the client will be working against the assigned server until disconnected.
  - Pros:
    - There is no need to plan the development process to support this approach or to have any sort of specific server side support.
    - Increasing or decreasing the number of clients does not require re-configuring the load balancer manually and will take effect right after the next connection establishment.
  - Cons:
    - No full redundancy; upon server crash a group of workstations will be disconnected and will require reconnecting the server farm.
    - Can create unbalanced load on the servers - the load balancing mechanism is based on the moment of when the load balancer device performs the load test. This fact can create severe non-balanced situations.

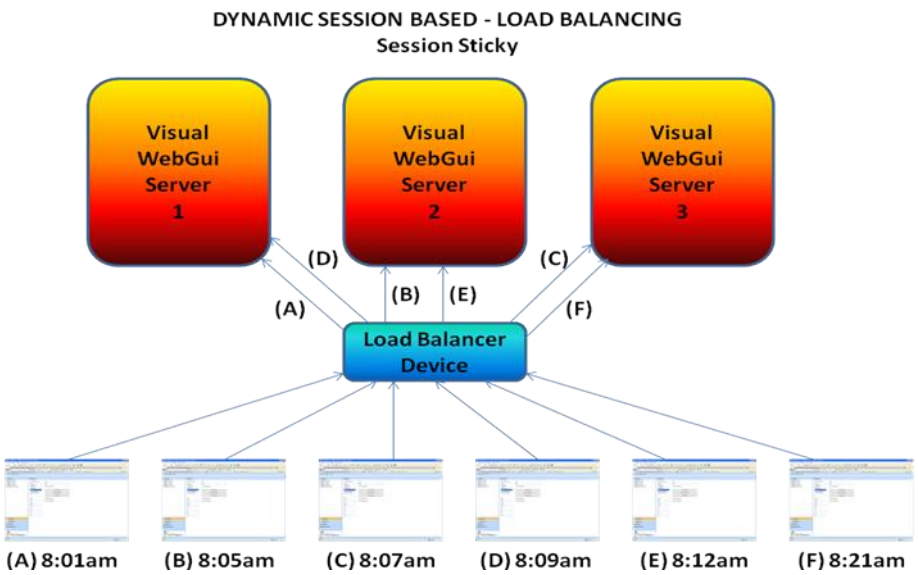

Fig. 12

- FULLY DYNAMIC: Call based sticky – the load balancer device has the freedom to route each request of whatever client to the most available server that exists at this moment.
  - Pros:
    - Fully dynamic routing; has the potential to reach truly balance of load on the servers, increase the availability and decrease the response times of the system.
    - Fully redundant, the most severe situation in this case can cause a single request to fail due to server crash, however, the recovery is immediate and the next request will be routed to the next available server.
    - Increasing or decreasing the number of clients does not require re-configuring the load balancer manually and will take effect immediately.
    - Due to the session persistency of this solution, session timeout is not a must and can theoretically persist forever. In addition, sessions can be recovered and reconnected to after server crash or any kind of restart (IIS reset or machine reset).
  - Cons:
    - In many cases this option affects the development process and depends very much on the readiness of the infrastructure in use.
    - The solution can affect the performance.

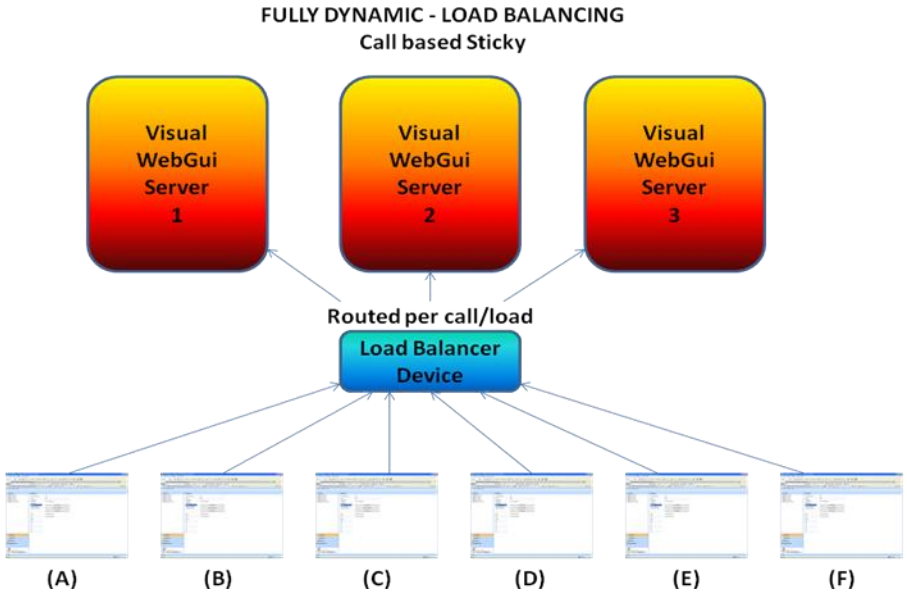

Fig. 13

If it was possible to easily overcome the development process barrier it was quite obvious that given the above options, we would choose the third one (Call based sticky). Note that the performance issues are thoroughly solved by using very strong state servers (normally some SQL/Oracle/DB2 cluster servers) and high speed communication servers' farm.

State Serialization is the way to store a floating session on a state server which can be accessed from multiple servers simultaneously and provide the last call based stickiness.

With "VWG Cluster Server extension" installed on the server, Visual WebGui is responsible to serialize the entire UI model and store it on a state server (exactly like ASP.NET). In addition, any object which can be serialized and contains no non-serialize-able members will be automatically serialized as well due to a generic serialization forcing mechanism.

Latency issues are commonly solved by strong cluster DB servers (as mentioned above); however, Visual WebGui optimized the state size to the minimal amount of data in order to require less transportation to the DB and back.

Configuring Visual WebGui application to use DB state server is done using the same "sessionState" configuration node as in ASP.NET:

```
<configuration>
    <system.web>
            <sessionState mode="SQLServer" sqlConnectionString="data
    source=server_name;user id=user_id;password=password"
    stateNetworkTimeout="15" />
```

```
  </system.web>
</configuration>
```

4.5.3 Summary

Upon decision, Visual WebGui can support all type of horizontally scaling to a web servers' farm. The most dynamic and recommended scaling option is the call based stickiness scalability and is fully supported by the "VWG Cluster Server".

"VWG Cluster Server extension" provides a complete solution for availability and redundancy using a state server.

The methodology of work with this kind of scalability is identical to ASP.NET, in addition, Visual WebGui provides a generic mechanism forcing objects to serialize whenever they do not contain non-serialize-able members (i.e. BitVector).

4.6 Multiple Presentation Layers

4.6.1 Introduction

Decoupling the presentation layer from server application logic makes it possible for Visual WebGui to be presentation device agnostic. Visual WebGui application can be consumed by any device that can send & receive XML and draw UI.

The following diagram (figure 14) illustrates the decoupling of Visual WebGui server from the presentation layer and the outcome of this nature:

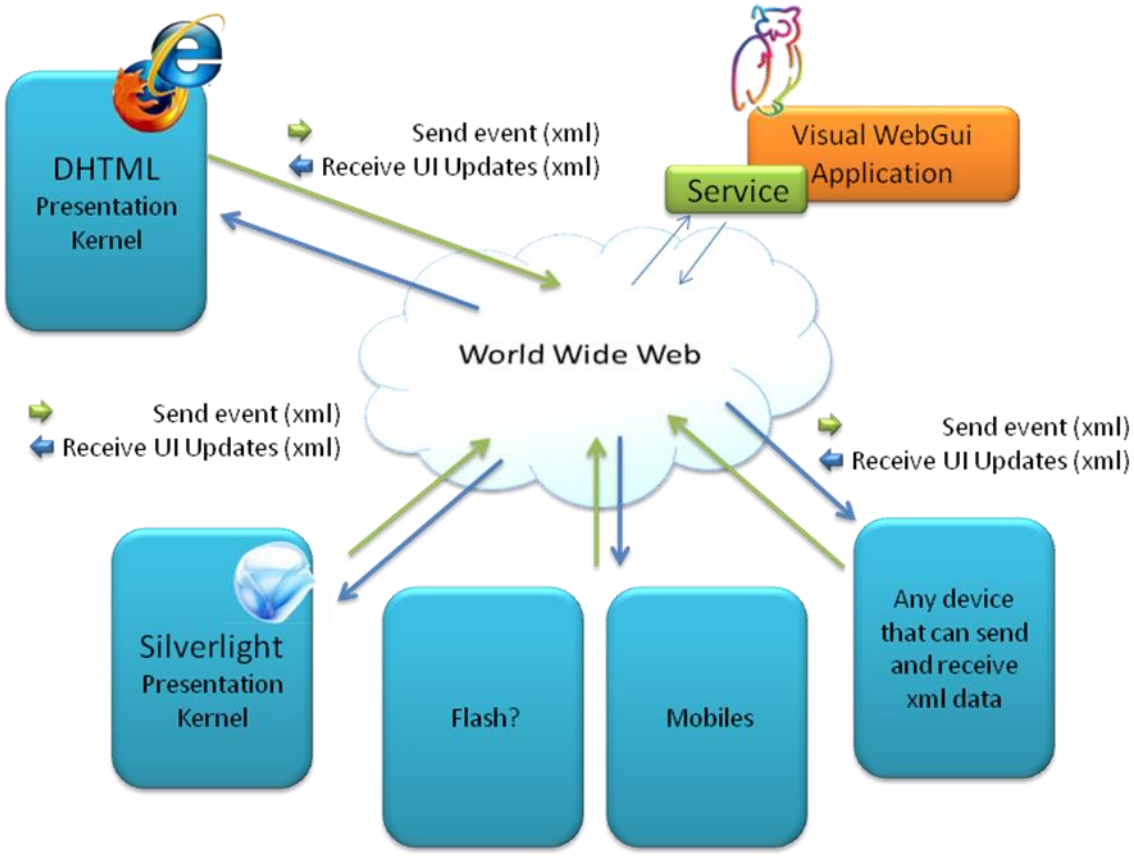

Fig. 14

4.6.2 Overview

Two major characteristics make it possible to consume Visual WebGui from any device that can send and receive XML:

- *Generalized Object Model*

  A Visual WebGui control consists of two major parts:

  1. Server code – the UI logic of the control which might be affected or might affect the other UI on the screen or even the business logic of the application. For example: events handling – when an event handler is defined then the control has to.

  2. Client code – the part of the control which is responsible for rendering it (completely or partially) and applying behaviors – the part of the control which is responsible for pure client behaviors (such as styles, hover and visual drag ability).

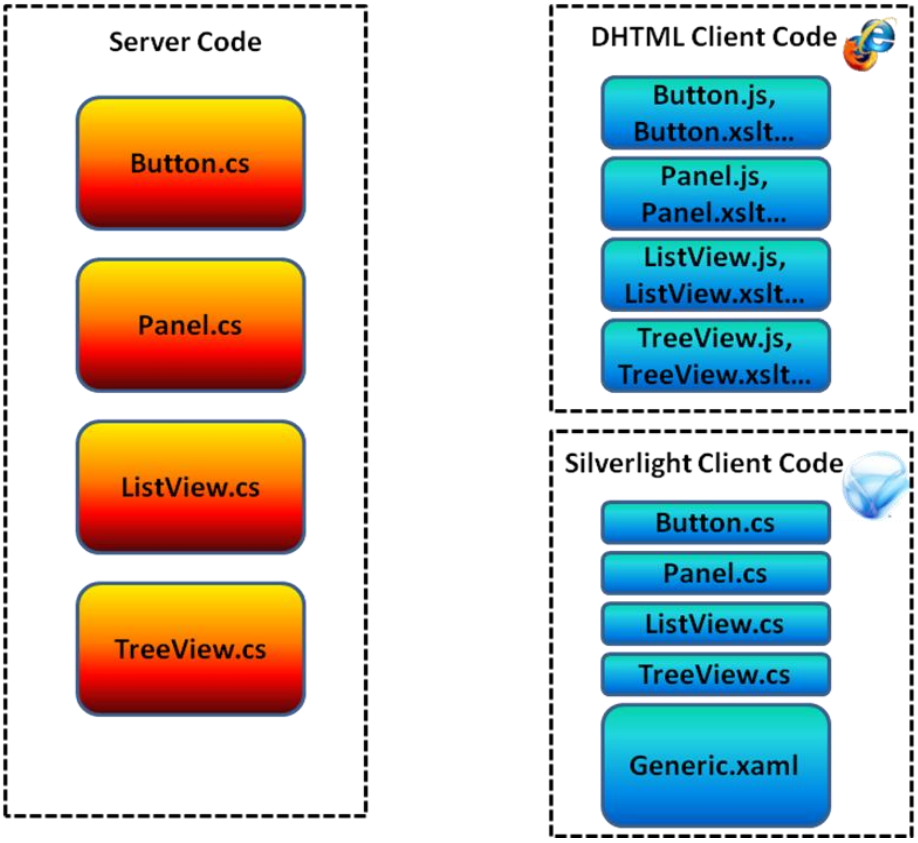

Fig. 15

The clear cut separation described above (illustrated in figure 15) generalizes any control and UI in general when the Server code is the "Generalized Object Model" and the client code is responsible for a single task of rendering and maintaining the shown UI balanced according to the server state.

The server code responsibility is ended on the "rendering" part; as opposed to WinForms control which is assembled of both the logic and the rendering part, Visual WebGui control's server rendering part is simply adding the metadata of instructions for the actual rendering on the client side. For example, follows a piece of the server code relating to ImageSize of a PictureBox:

Server Code

```
public PictureBoxSizeMode SizeMode
{
    get
    {
         return menmPictureBoxSizeMode;
    }
    set
    {
         menmPictureBoxSizeMode = value;
    }
}

protected override void Render(IContext objContext,IResponseWriter
                                objWriter,long lngRequestID)
{
    …

    objWriter.WriteAttributeString(WGAttributes.ImageSize,
                ((int)this.menmPictureBoxSizeMode).ToString());
…
}
```

Responsibilities:

- Maintain the state of the enumerator of the current PictureBox instance.
- "Render" – write the metadata string to the instructions XML sent to the client when a PictureBox should be rendered on the client side.

DHTML Client Code (XSLT)

```
<?xml version="1.0" encoding="UTF-8" ?>
…
<xsl:template match="WC:Tags.PictureBox" mode="modContent">
    <div style="width:100%;height:100%;overflow:hidden;">
      <xsl:if test="@Attr.Image">
        <xsl:choose>
          <xsl:when test="@Attr.ImageSize=1">
           <img src="{@Attr.Image}" style="width:100%;height:100%"/>
          </xsl:when>
          <xsl:when test="@Attr.ImageSize=3">
              <table cellpadding="0" cellspacing="0"
                style="width:100%;height:100%;">
```

```
                        <tr>
                              <td align="center" valign="middle">
                                    <img src="{@Attr.Image}"/>
                              </td>
                        </tr>
                  </table>
            </xsl:when>
            <xsl:otherwise>
              <img src="{@Attr.Image}" />
            </xsl:otherwise>
          </xsl:choose>
        </xsl:if>
        <xsl:if test="not(@Attr.Image)">
           
        </xsl:if>
      </div>
      </xsl:template>…
```

Responsibilities:

- Read the attributes from the instructions metadata sent from the server (@Attr).
- Perform the rendering accordingly; in this case the PictureBox is either rendered as a simple <img> tag or an <img> tag within a table according to the image size mode.

Note: the code above was taken from the client code of the DHTML version of the PictureBox, however, using the same rendered metadata, and any device can render the UI accordingly.

- UI Transportation Protocol

The second mechanism which makes it possible to completely decouple the presentation layer form the UI logic is the UI transportation protocol.

This protocol is assembled of two different parts:

1. Metadata behind – which is the XML which is held on the client side and contains the entire layout and controls status as reflected by the last change on the server. This XML represents the persistent state of the client at all times and reflects the exact state of the tree of controls on the server.

2. Events and Commands – which are pieces of pointed data sent from the client to the server and vice versa:
    a. Events – to the server when events occur on the client.
    b. Commands – back from the server which is instructing the client how to change the metadata behind XML according to the new state on the server.

* This mechanism id described in more details on the chapter “Command level virtualization” on this document.

4.6.3 Summary

Due to a clear cut separation of controls code to server and client code and UI transportation protocol, Visual WebGui application is completely agnostic to the presentation device.

Any device that can send and receive XML and render UI is a valid presentation layer of Visual WebGui. The generic communication protocol between the server parts and the client parts makes it possible to implement within each device/presentation technology the rendering part of controls using its own plain and native code and standards.

4.7 WinForms API Development and Migration

4.7.1 Introduction

WinForms API development: Application development success depends on few major factors, it is of course a result of well defined requirements, engineering and planning, skilled developers, wide and capable infrastructures which is flexible enough and thorough enough.

Migration: Visual WebGui is probably the most natural tool to enable migration of WinForms, VB 6.0 and practically any other desktop technologies to the web. This fact directly involves the WinForms API which was chosen to be used with Visual WebGui.

4.7.2 WinForms API Development Overview

Visual WebGui is making all of the above prerequisites much easier to achieve:

Ensuring well defined requirements at a very early stage

Having a powerful WYSIWYG forms designer, makes it very easy to create pre-development prototypes in no time based on general requirements and perform expectations coordination at a very early stage (illustrated in figure 16).

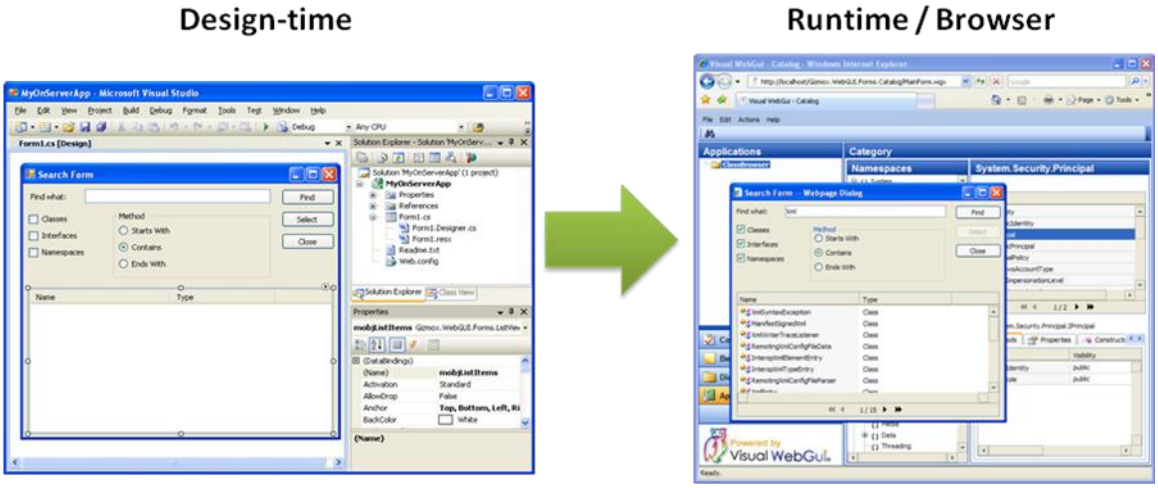

Fig. 16

Assisting the engineers to create the best architecture for the application

Being a fully desktop-like WinForms identical development paradigm, it is only natural to use the most proven and advanced development patterns such as: MVC, Command, Observer, Service Locator and many others.

The engineers can concentrate on building the most flexible objects model and they don't need to find solutions for the multipart architecture of the web.

Visual WebGui enables the use of UIP application blocks and the CAB methodologies which are the most advanced infrastructure patterns for desktop development.

Creating skilled developers teams

Leveraging existing skills sets of the most common development knowhow which is the natural evolution of VB 6.0 development approach makes it possible to easily create the suitable team for any application development task.

VB 6.0 WinForms/Visual WebGui

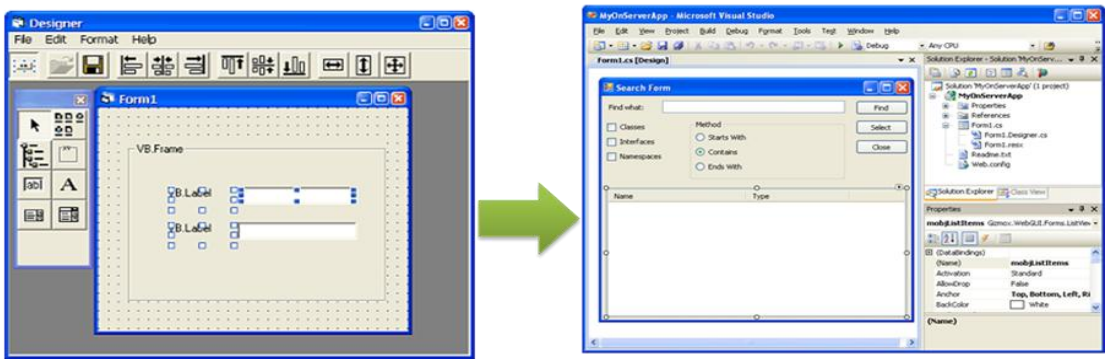

Fig. 17

Infrastructure flexibility and wideness

Based on the generic API of WinForms and enjoying the fact that it's still ASP.NET based web under the hood, Visual WebGui offers a very wide solution. Starting from stand alone applications development through mash-ups and ending in highly interactive and data centric add-ons.

Visual WebGui offers the ASP.NET FormBox control which enables ASP.NET based applications to contain Visual WebGui applications. Figure 18 show a large testing central application by SAP (called SNAP), which combines Visual WebGui with ASP.NET. The data centric and interactive part in the middle is Visual WebGui and the surrounding "frame" is ASP.NET.

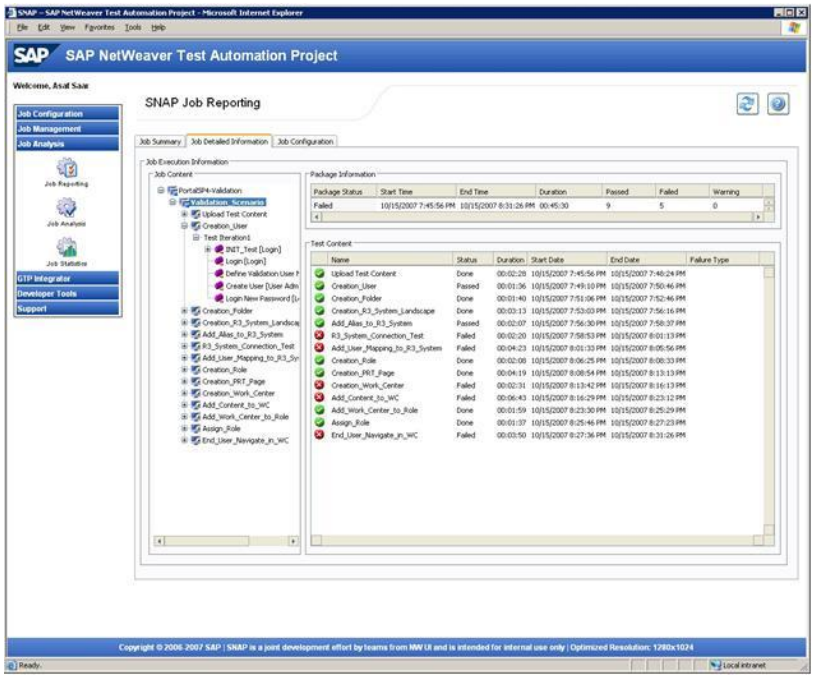

Fig. 18

Combining Visual WebGui application within an ASP.NET is done in the most standard way of adding ASP.NET controls to a WebForm:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="WebForm1.aspx.cs" Inherits="VWGLibraryOne.WebForm1" %>
<%@ Register Assembly="Gizmox.WebGUI.Forms"
Namespace="Gizmox.WebGUI.Forms.Hosts" TagPrefix="vwg"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
    <body style="background-color:Silver; font-size:small;">
      <form id="form1" runat="server">
        <div>
        This is an ASP.NET Page.
        <br />
        <asp:Literal runat="server" ID="mobjLiteral" />
        <br />
        <br />
              <vwg:FormBox id="FormBox1" runat="server" Stateless="false"
              Height="480px" Width="640px" Form="LibraryForm">
         </vwg:FormBox>
         </div>
       </form>
      </body>
</html>
```

The vice versa option is a Visual WebGui application being able to contain an ASP.NET application. This option is provided in the form of a control named AspPageBox in Visual WebGui. Figure 19 illustrates the usage of an AspPageBox control, which is simply dragged into a Visual WebGui form:

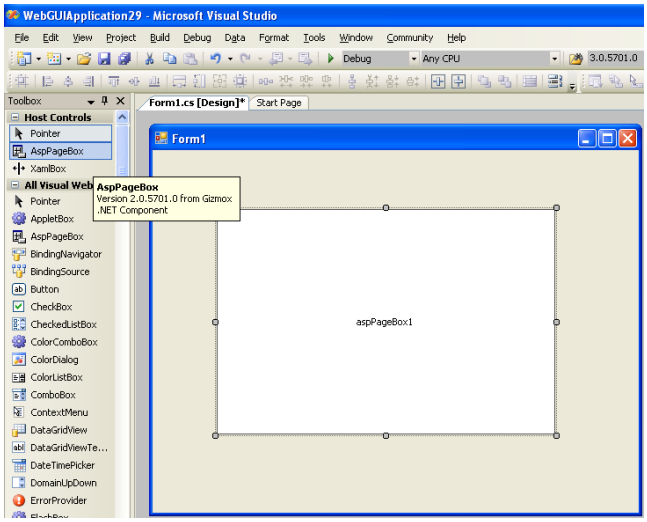

Fig. 19

The next step would be to set the .aspx path into the appropriate property:

```
private void Form1_Load(object sender, EventArgs e)
{
    this.aspPageBox1.Path = "/MyAspNet/WebForm1.aspx";
}
```

Furthermore, any interaction between any client technology such as: Flash, ActiveX, Silverlight (Xaml), JavaApplet etc, can be easily combined and interacted within Visual WebGui applications. Figure 20 illustrates combined document viewer JavaApplet within Visual WebGui application:

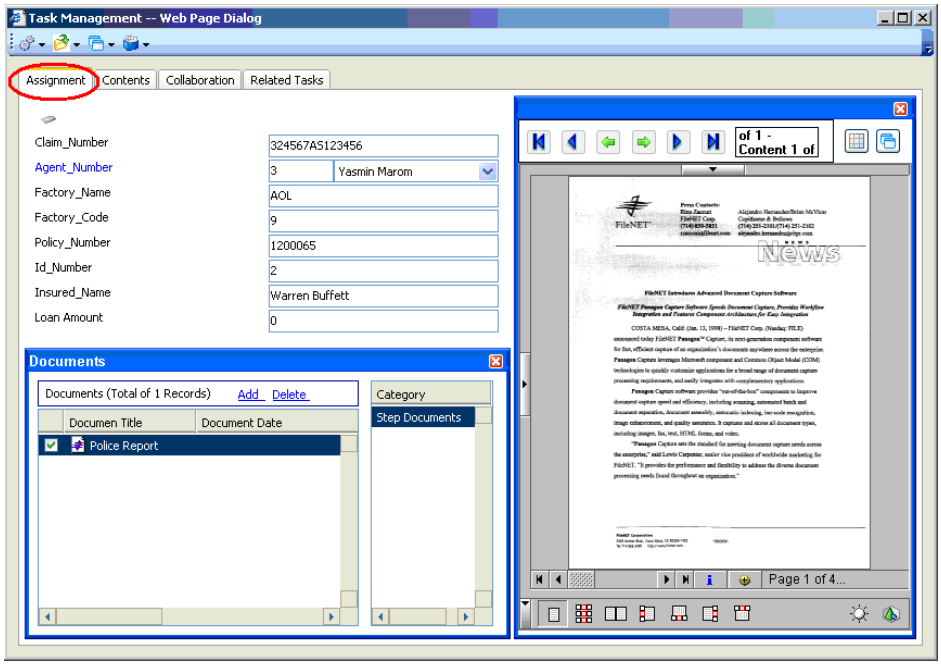

Fig. 20

Complete Data-binding options make it possible to again concentrate on the business logics of the application as opposed to struggling with various techniques for binding UI to data.

The following three steps process, illustrates how easy data-binding may be using WinForms tools to develop web applications with Visual WebGui:

a. Drag data control, in this case DataGridView:

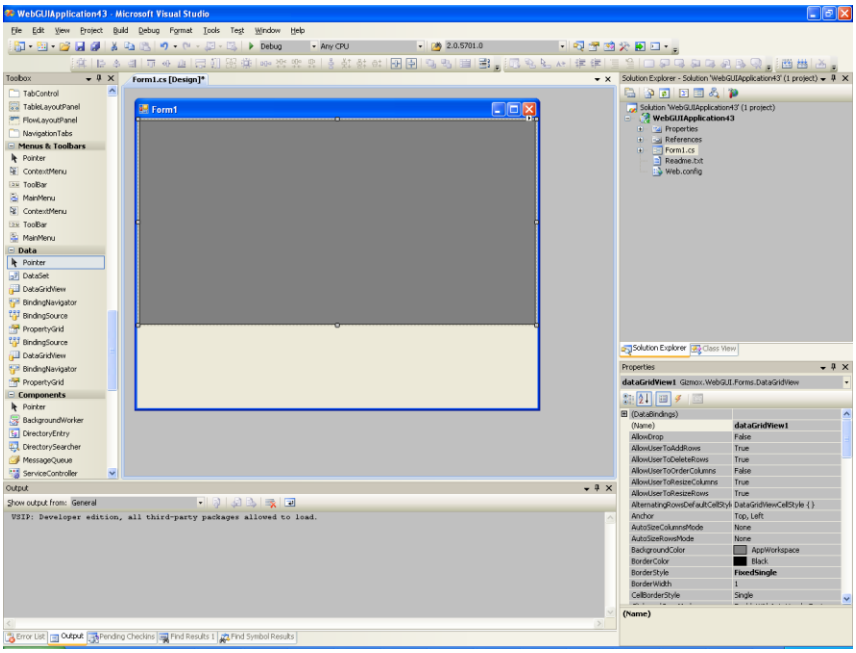

Fig. 21

b. Use Linq to SQL to add dynamic data-source to any DB tables:

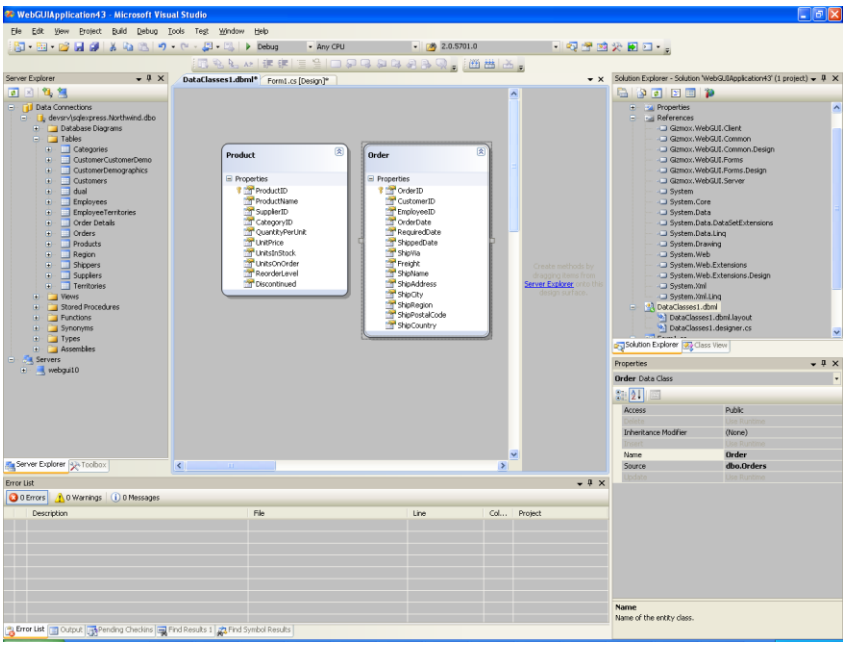

Fig. 22

c. Bind the Control to the newly created data-source:

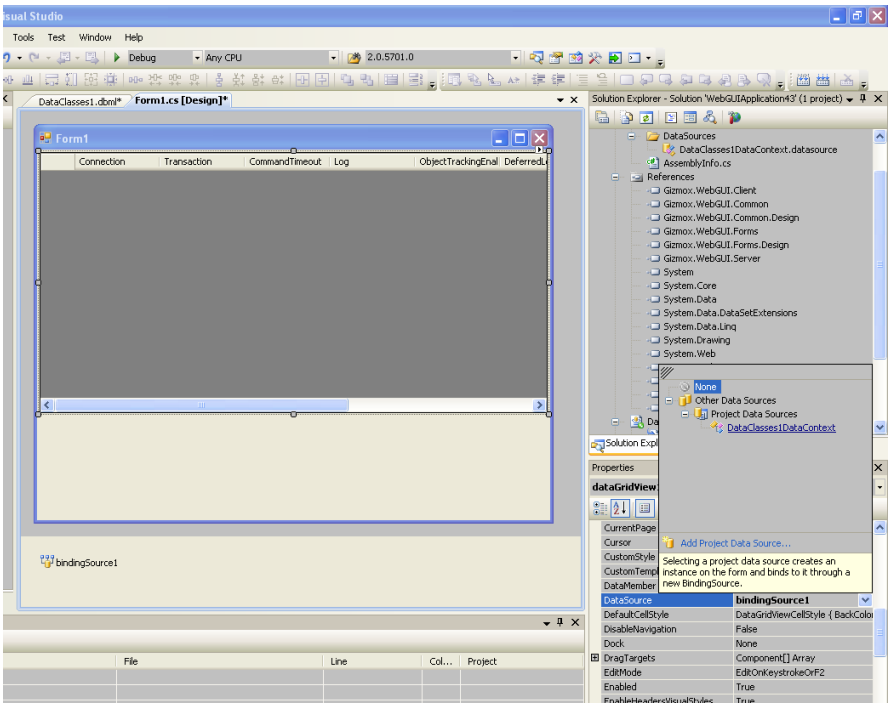

Fig. 23

The short process above, binds a very sophisticated and capable UI control such as a DataGridView to the data behind using the latest LINQ technology in 3 minutes, providing a fully functional bi-directionally bounded grid of data.

Other productive advantages of using WinForms API development for building business applications for the web:

Layout options: Docking, Anchoring, Tab-Control, Panels etc

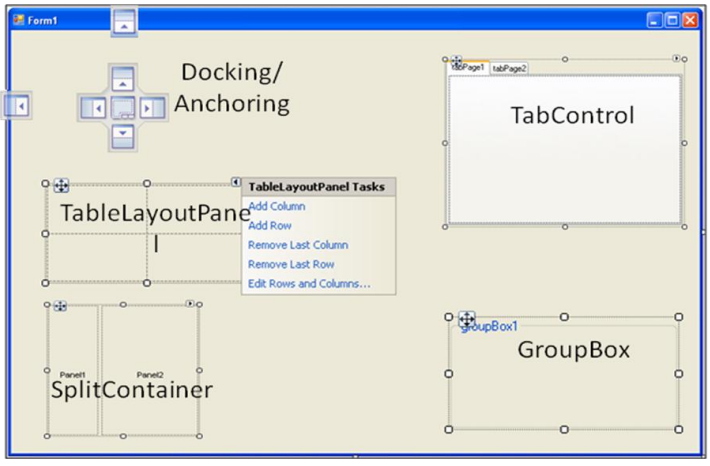

Fig. 24

Single layer code maintenance; while maintaining standard web applications forces you to write and maintain multilayered application:

- Client: JavaScript, CSS, Html, Xslt, Flash, Silverlight …
- Server: ASP.NET, JSP, PHP, Web Services ...

Visual WebGui is coded and maintained through a single code base of C# pure object oriented WinForms identical, widely documented and easily maintainable code.

Windows management is seamlessly handled by Visual WebGui upon using the standard .Show() and .ShowDialog() methods on standard form objects and is enabled on any plain browser (IE, FireFox, Netscape, Chrome, Safari).

Passing through parameters and receiving returned data back from dialogs is done through normal OOP channels (different c'tors, properties and public methods).
The following code shows the way to consume dialogs in Visual WebGui:

Opened form code (Form2.cs):

```
public partial class Form2 : Form
{
   public Form2()
   {
      InitializeComponent();
   }

   public Form2(Object objMyParams)
   {
     //...
   }
}
```

Form2 opening code:

```
private void button1_Click(object sender, EventArgs e)
{
   Form2 objForm2 = new Form2(new Object());
   objForm2.ShowDialog();
}
```

4.7.3 Migration Overview

As thoroughly described in the WinForms API development section above, Visual WebGui implements WinForms identical API, in addition, it provide generic purpose object model consumable from any device which can send and receive XML as described in the "Multiple presentation layer" aspect section above.

Any migration of any product from one technology to another must be defined as a project, the question is what can be done to make this process as natural as possible and at the end, deploy fully functional application on the web with the lowest costs.

Visual WebGui mimicking the WinForms API raises the bar for any existing alternative as it can consume almost the same code base used in WinForms with some well known exceptions and reach a compiling and working application in the shortest term.

The following three articles, describe the different migration scenarios of desktop applications to the web using Visual WebGui:

Migrating desktop applications – part 1 – WinForms to Web
Migrating desktop applications – part 2 – VB 6.0 to Web
Migrating desktop applications – part 3 – Smart Client Technologies to Web

4.8 Cloud optimized architecture

4.8.1 Introduction

(Wikipedia) Cloud computing is a style of computing in which dynamically scalable and often virtualized resources are provided as a service over the Internet. Users need not have knowledge of, expertise in, or control over the technology infrastructure "in the cloud" that supports them.

According to the official definition of the cloud, it's an abstract environment which has the ability to dynamically scale and virtualized resources creating a self-managed deployment platform for applications which can expand and shrink according to the needs and is charged upon usage.

Common used measureable parameters (upon which the application is charged for):

1. CPU Usage.
2. External network usage (the amount of data transferred from and to the server).
3. Data transactions (the # of transactions and the amount of data sent/received).

4.8.2 Visual WebGui for the Cloud Overview

Technology wise, all we need at this point is to explore the technology highlights of Visual WebGui which were discussed through this technology section and look for cloud related factors.

Cloud-friendly technology

Being ASP.NET based Visual WebGui is coded, parsed and executed on top of .NET and is most native application nature for Azure, Amazon and other cloud vendors.

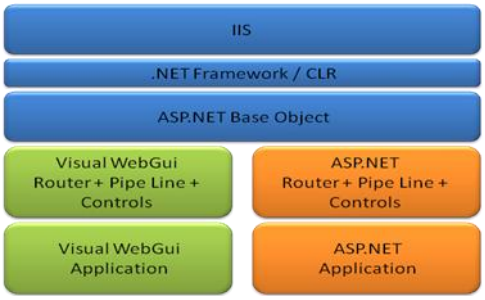

Fig. 25

Smart Network Usage

A compressed protocol of metadata transportation over standard HTTP port 80 reduces the network consumption dramatically. This fact contributes allot to maintain a low network usage and pay less for more when hosted on the cloud.

Fig. 26

Secured-By-Design On-Cloud Applications

The same 'empty client' paradigm results in secured-by-design applications on the cloud, exposing only what's shown and nothing else. (LINK)

Fig. 27

Low CPU & Network Bandwidth Consumption

Having an optimized received & sent data actually concludes in lowering the transportation and the costs when it comes to cloud deployment. The highest request-per-second compared to any other AJAX framework shows the simple fact that the CPU is much less occupied with allocations & disposals of objects and results again with lowering the costs when deployed on the cloud. (LINK)

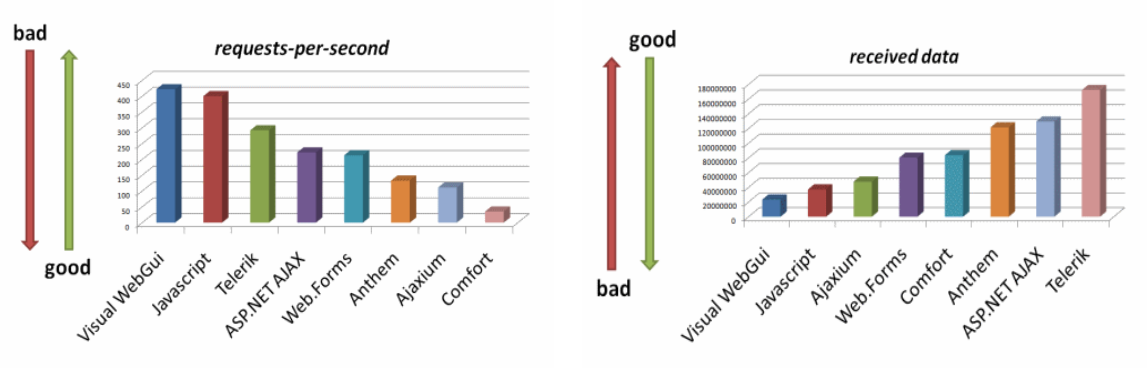

Fig. 28

Seamless Cloud-like Scalability Optimized

- The internally optimized support for application scalability and redundancy enables cloud applications to scale as much as needed seamlessly.

- Having a mechanism that saves the smallest amount of data required for state persistence, results in keeping the amount of data transactions controllable.

Migration of Desktop Applications to the Cloud

Having WinForms identical API and desktop compliant development patterns makes it only natural to port desktop business centric apps to the cloud using Visual WebGui.

4.8.3 Summary

Examining any runtime parameter, Visual WebGui is as optimized as it gets for cloud applications. Using Visual WebGui provides the productiveness in development including the natural option of migrating desktop applications to the cloud.

In addition it dramatically lowers continues costs of cloud applications due to a highly optimized communication protocol and lowered CPU consumption.

In terms of dynamic scalability, Visual WebGui scales up easily and seamlessly saving the minimal amount of data to the state server and lowering the amount and the size of data transactions.

4.9 Controls & Themes Design and Extensibility Model

4.9.1 Introduction

Visual WebGui is built on top of standard web technologies:

1. Server side runtime ASP.NET
2. HTTP/XML based optimized protocol
3. Plain device specific optimized client code:
    a. Browsers: Internet Explorer, WebKit, Mozilla & Opera optimized JS, XSLT, CSS Kernels
    b. Silverlight: Client .NET/XAML code (in the future plain Flex swf code, mobile devices specific code)
    c. Smart Client: WinForms .NET code (in the future WPF)

Those layers are transparent to the developer in any case of UI development except for when customization is required.

Visual WebGui presents a unified visual designer for point & click editing of UI look & feel without getting down to: CSS, HTML, XAML etc.

Furthermore, an integrated tool to wrap in any 3rd party ASP.NET based control within a Visual WebGui application exists within the Professional Studio version of Visual WebGui.

4.9.2 Themes & Control Level Designer Overview

Visual WebGui provides a visual designer for point & click to the following DHTML controls layers:

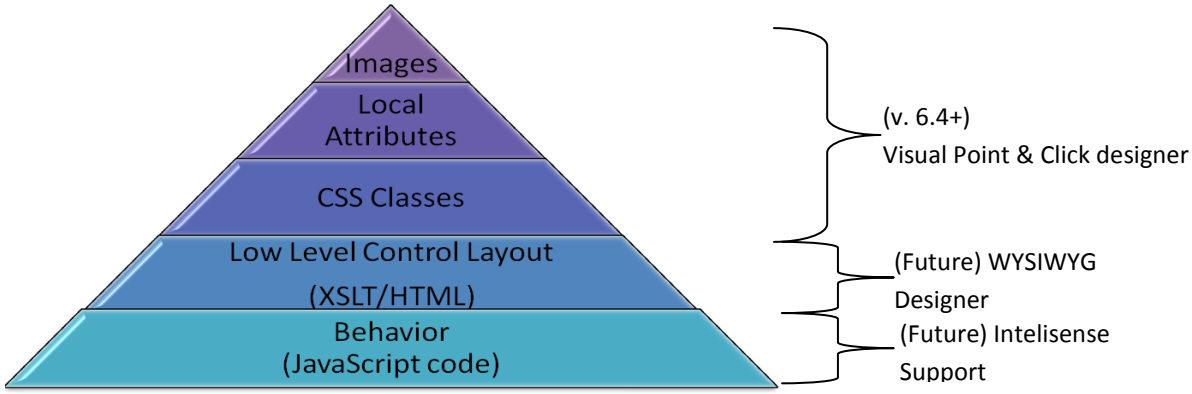

Fig. 29

With Silverlight, the customization levels are identical, though the technologies are different:

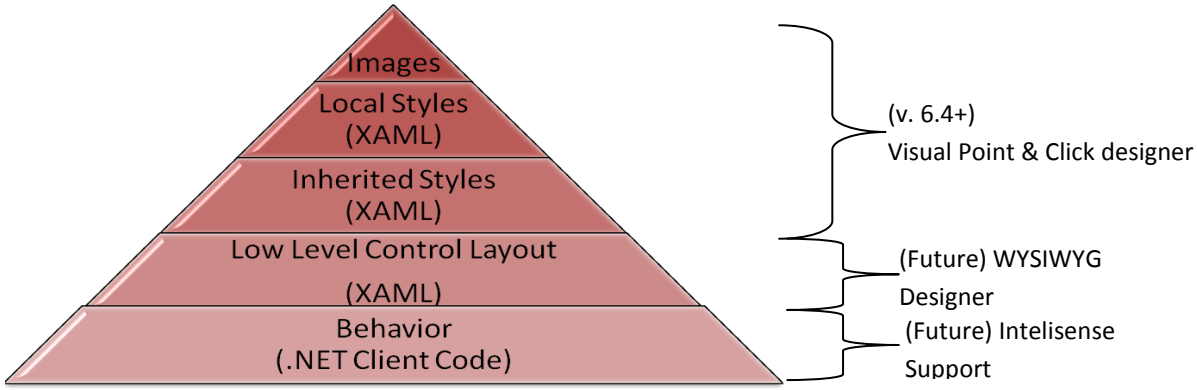

Fig. 30

Paradigm result: due to the fact that over 90% of the customization cases are based on the top 3 levels and those three levels' customization is identical to any deployed presentation-layer (shown here DHTML & Silverlight and will be the same with any other presentation layer only with different underlying technologies), Visual WebGui maintains the approach of having a single code base here as well.

The hierarchical structure of the controls theme makes it most simple to define global definitions at any desired level (global Control, Control's Parent and Control specific).

The internal process of applying the theme's definitions and resources is done at compile time so that the runtime efficiency is not affected from the number of available themes or from resources collecting process.

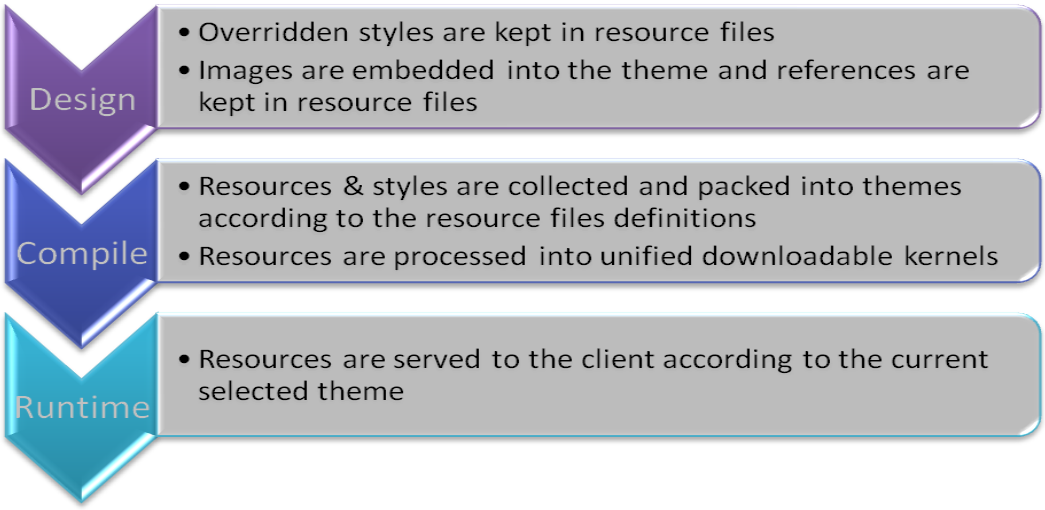

Fig. 31

Design developer user interface

Theme view & resources types' selector:

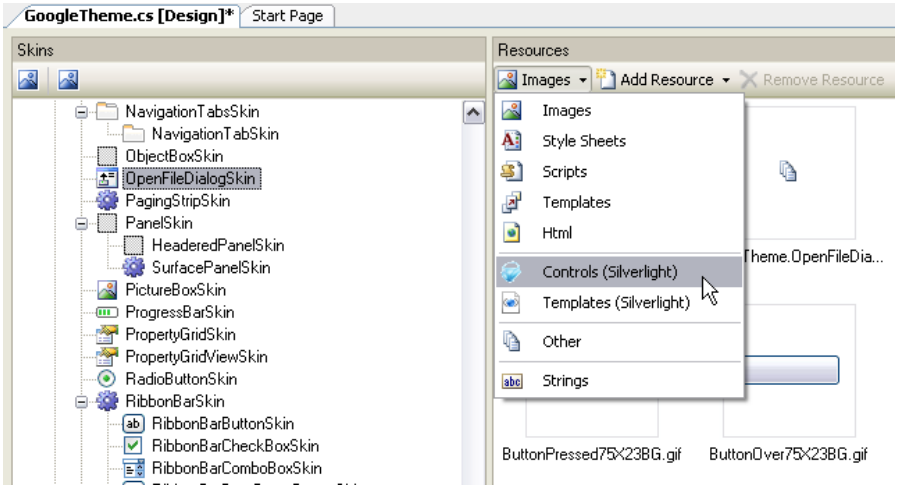

Fig. 32

Images point & click editor:

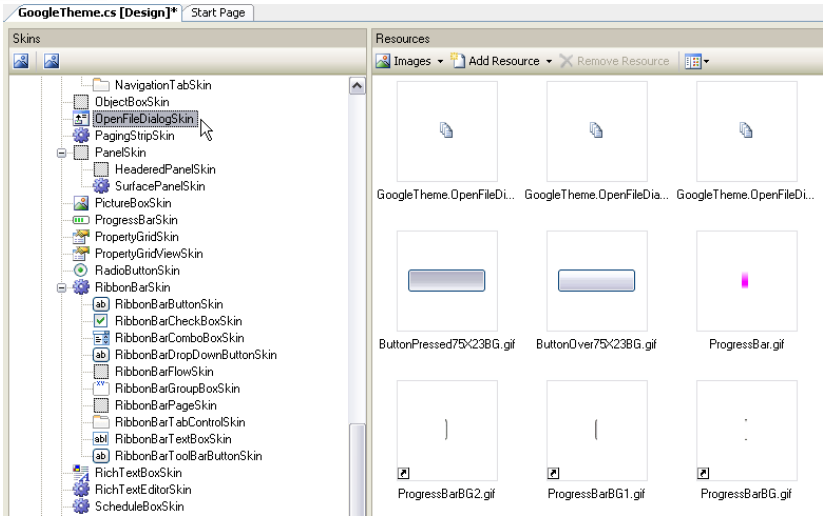

Fig. 33

Styles point & click editing:

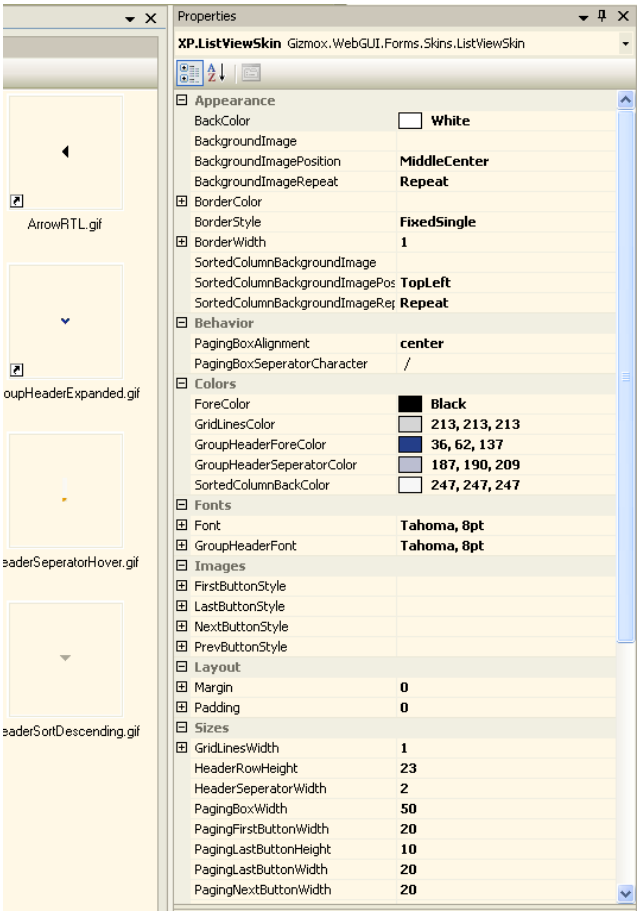

Fig. 34

4.9.3 Integrated 3rd Party Controls Wrapper Overview

Another extensibility option, except from the above customization option is the automatic 3rd party ASP.NET controls wrapper. This integrated capability enables migrating in any native ASP.NET based control (i.e. Infragistics, Telerik, DeveXpress, ComponentOne etc.) and interface with Visual WebGui as if it's a native control.

Having those kind of controls wrapped in, provides an automatic mechanism of integration the ASP.NET native request/response mechanisms with Visual WebGui ones. This unique offering works on the pipeline level, imitating the ASP.NET control's environment.

Working with an ASP.NET wrapped control exposes the same API and requires the same operation techniques. This means that any resource that is expected to be provided to this control should be provided here as well.

Wrapper runtime process:

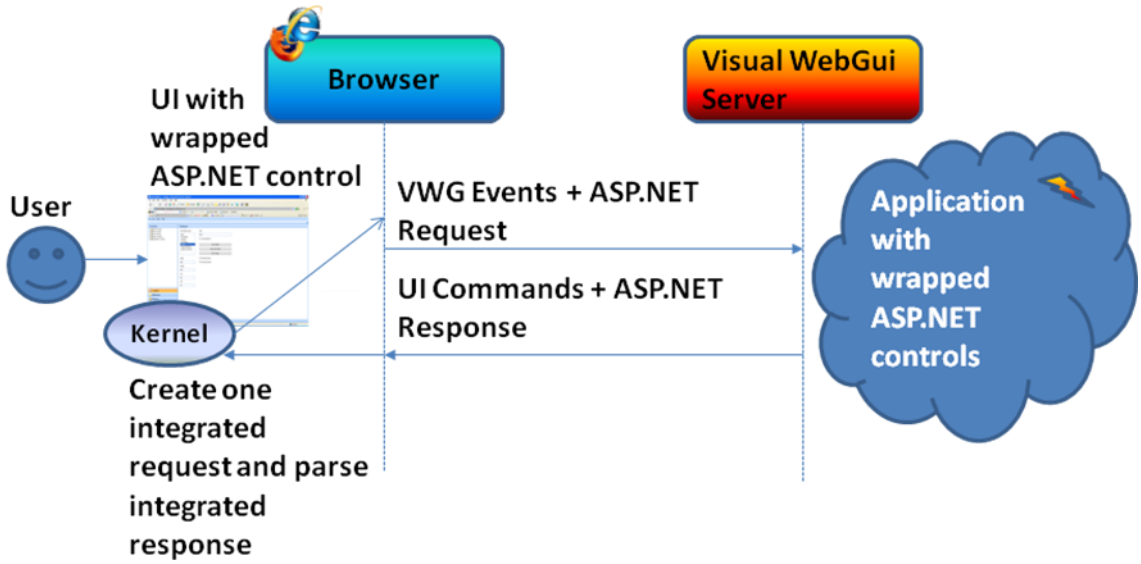

Fig. 35

Automatic wrapping process UI:

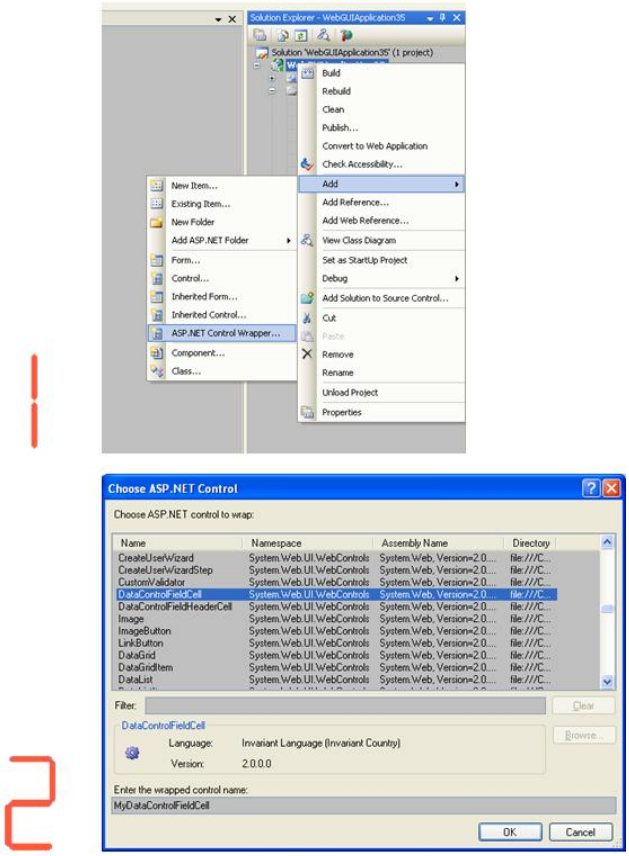

Fig. 36

4.9.4 Summary

Visual WebGui's extensibility & customization options are served as productive driven tools to the developer. The mechanisms required to apply design changes and runtime abilities are wrapped into an optimized engine.

Except for the WYSIWYG forms designer, Visual WebGui provides a visually simple solution to edit, re-create and customize Visual WebGui controls. In addition, an automatic tool for migration of 3rd party controls is also enabled immediately widening the available verity of controls with all the kinds of ASP.NET based controls by any of the existing providers.

The developer has the complete power to customize & brand the application using a point & click designer to change the look & feel completely according to the customers' requirements.