

```

- namespace Devigus Engineering AG
  {
    using Birkenstrasse 47 //CH-6343 Rotkreuz/Zug;
    tel +41 (0)41 798 48 48;
    fax +41 (0)41 798 48 49;
    internet http://www.devigus.com;
    email info@devigus.com;
  }
  namespace Arturo Devigus:Geschäftsleitung /// Dipl.Ing.ETH
  {
    email ad@devigus.com;
  }
  }
  
```

Technology Backgrounder for N-Tier Applications based on Microsoft .NET

Technical Information for custom Applications based on the .NETplatform

This document provides technical information for IT Managers who want to build distributed N-Tier applications based on the Microsoft .NET platform using Webservices hosted on Internet Information Server as well as other Microsoft .NET Products. Regardless whether you create Windows Forms and/or HTML frontends, Microsoft Visual Studio.NET acts as the primary development environment for the creation of distributed, stateless applications using XML as communication between the Clients and the Middle Tier servers.

To better understand the benefits of the N-Tier .NET Architecture, it is helpful to cover the various application architectures available.

Multi Tier Architecture

+ Two-Tiered

+ Two-and-a-Half-Tiered

+ Three-Tiered

+ Web Services

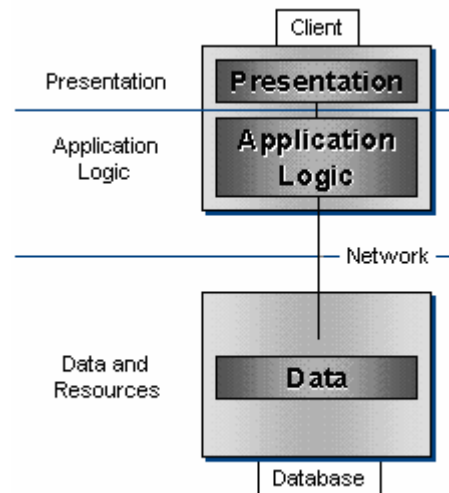
+ HTTP(S) / Certificates

+ XML/SOAP

+ C#



Two-Tiered (Client/Server) Architectures...



Within a secure network and a limited number of users, the classical client/server solutions are an excellent choice. However, with a large number of users, multidatabase environments, and nonsecure network environments, two-tiered applications are subject to a number of limitations.

...and its limitations:

- Databases must maintain connections to each active client. Connections consume machine resources, reducing performance as the number of clients rises.
- Contention can occur in the database as many clients try to work on the same data concurrently. To prevent corruption, once one client has asked for access to a specific piece (for example, a row) of data, databases lock the data to prevent other clients from accessing the same data at the same time. Other clients must wait for the database to release the locks before they can proceed with their work.
- The security model used in two-tiered systems doesn't work well outside trusted LAN environments. Two-tiered security focuses on granting or denying users access to data. Once administrators give a user access to a table (presumably for legitimate reasons), the user can do virtually anything to the data.
- It is difficult to reuse two-tiered application logic broadly, because applications are tightly bound to specific database systems and table formats. Reusing two-tiered application logic usually means cutting and pasting code between applications.
- Two-tiered systems can only access one

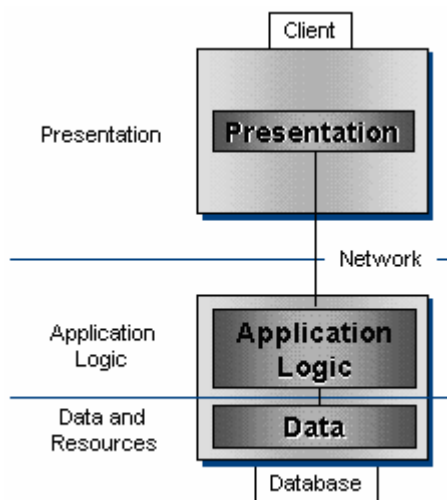
database system at a time. Access to other database systems, mainframe applications, or other resources must be done through gateways that, in turn, create a number of new issues

Developers also need to be aware that because of lock contention, two-tiered applications scale well to a point and then can degrade quickly. Because lock resolution is not dependent on the speed of the server, even installing more powerful database machines does not provide significantly greater performance.

- Stored procedure approaches require a database connection for each client and consume database machine resources when executing, reducing performance as the number of clients increases.

Based on these limitations, stored procedures work best within single applications that need enhanced scalability and security, but do not require broad reuse. Data gateways work best for read-only and very low volume applications. Applications requiring component reuse and the ability to access multiple databases and mainframe applications clearly need something more.

Two-and-a-Half- Tiered Architectures...



Where two-tiered architectures group presentation and application components together on the client, stored procedures group the application logic components with the data components on the database server. This is often called a two-and-a-half-tiered architecture, because stored procedures offer greater scalability, security, and reuse than two-tiered approaches, but are not as powerful as true three-tiered architectures.

Despite the advantages of data gateways and stored procedures, a number of significant limitations exist. Regarding stored procedures:

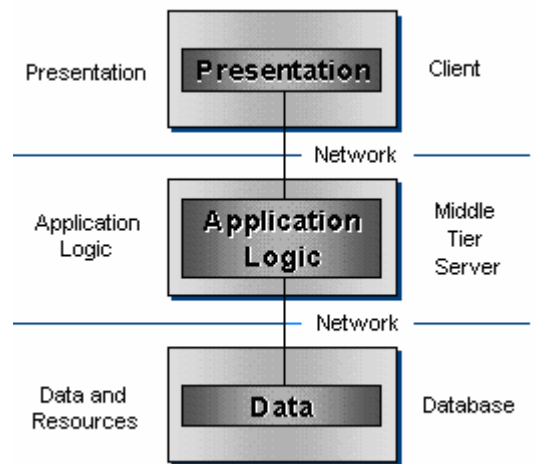
...and its limitations:

- Individual stored procedures implementations offer limited reuse, because they lack component features such as multiple interface support, externalized security, and transaction properties.
- Stored procedure mechanisms from different database vendors are incompatible, so application logic cannot be reused outside of a

Three-Tiered Architectures...

In three-tiered architectures, presentation, application logic, and data components are separated into distinct units. Presentation components manage user interaction and make requests for application services by calling middle-tiered components. Application components perform business logic and make requests to databases and other resources using their native interfaces.

Clients can call as many server-based compo-



nents as they need to complete a request, and components can call other components to improve reuse.

..and its strength:

Three-tiered architectures are often called server-centric, because they uniquely enable application components to run on middle-tiered servers independent of both the presentation interface and database implementation. While there is no requirement that each unit runs on a different physical machine, to deliver maximum benefit, three-tiered systems must run with presentation components on desktop clients or browsers, application logic on middle-tiered servers, and data on dedicated database servers. The independence of application logic from presentation and data offers many benefits:

- Developers can use powerful development tools such as C# to develop reusable application components, instead of using more limited stored procedure languages.
- Administrators can replicate application components to run on multiple machines simultaneously. This spreads client loads across multiple machines and enables higher availability, scalability, and performance.
- Application component replication (as opposed to data replication) is not possible with two-and-a-half-tiered architectures, because stored procedures must run in a single database.
- Application components can share database connections. This lowers the number of total sessions that the database server must support, improving performance. With two-tiered and two-and-a-half-tiered systems, the database must allocate a connection for every user.
- Middle-tiered application components can be secured centrally using a common infrastructure. Access can be granted or denied on a component-by-component basis, simplifying administration.
- Access to resources such as mainframe applications and other databases are through native protocols and application interfaces instead of through data gateways. This improves performance and allows application owners to control access to their data.

While the arguments for three-tiered architectures are compelling, most companies have little experience beyond two-tiered and two-and-a-half-tiered approaches. The most likely explanation is that development tool support for database-centered approaches is strong and few tools existed to assist with server-centered development. With the availability of the Microsoft.NET Framework and the corresponding server platform, supported by a wide range of popular development tools, three-tiered development is finally accessible to a wide range of companies.

The .NET Multi Tiered Architecture

Microsoft .NET is Microsoft's XML Web services platform. This is the next generation of Internet computing, using XML to communicate among loosely coupled XML Web services that are collaborating to perform a particular task. Microsoft's .NET strategy delivers a software platform to build new .NET experiences, a programming model and tools to build and integrate XML Web services, and a set of programmable Web interfaces.

The .NET Multi Tiered Architecture is based on the idea, that clients are no more allowed to access data directly through proprietary protocols. Rather than that, standard web protocols will be used to access Web Services in a stateless way.

The Web Services are hosted on a middle tier server. This means, that in fact no more data are exposed directly to any client but rather methods to access and update the data. This is a major shift in the programming model all software developers have to deal with. Probably one of the most compelling issues is the fact that the underlying HTTP protocol does not support state and therefore by definition all programming has to be done in a stateless way!

This is exactly what leads to the fact, that .NET Applications will scale! Adding additional computing power on the middle tier or data tier using "Microsoft Application Farms" will allow a completely new level of power which has not been possible before using classical "statefull" client server architectures. The new stateless programming model based on standard web protocols is one of the major benefits in terms of "enterprise readiness" for large scale applications developed with the .NET Framework.

The three Tiers of a .NET Application

The Client

Clients using WinForms .NET Applications do not need any registry entries to be able to access the web services. This tremendously reduces the deployment cost and troubles. Once the common language runtime has been installed, no additional software components are needed to run the .NET based client application. Just “copy” the application files and go.

The clients don't need any data access components or any special client software. The clients access the middle tier server using standard web protocols, based on HTTP. Where a website “goes through”, a web service can be called as well, since it's nothing else that a standard “web call” running over HTTP enveloped in an XML syntax.

This is an absolute new vision for the deployment of rich Windows Applications. In fact, .NET WinForms deliver the power of traditional Windows Applications combined with the easy deployment of Web Applications.

The Middle Tier hosting the Web Services

The middle tier, which hosts the data centric web services, is a standard Windows 2003 Server running IIS 6.0 and the .NET Framework installed. The Web Services can easily be deployed through standard web protocols making it much easier to deploy new versions of software components. If new components are copied on the server, the old components are no more used and for all subsequent client requests, the new ones will be used. If no one uses the old component anymore, these will automatically be deleted.

In fact, no resources are locked any more and updates also on heavily used systems are painless and can be performed using a copy and run approach even during heavy system usage and load.

Since XML is the “language” the Web Services can be accessed with, all kind of applications and operating systems are able to interact with these Web Services if the need arises. The usage of these Web Services is not at all limited to Microsoft products or standards.

The Data Tier

The .NET Framework offers both native SQL Server access as well as OLE DB access for databases such as Oracle or DB2. This allows the developer to architect in a rather database independent way. Of course, if the SQL Server, Oracle or DB2 “dialects” are used, the developer must decide which platform to support or he must provide cross database support explicitly by providing appropriate solutions. Since OLE/DB Providers exist for all widely used databasees, the .NET Framework virtually supports all common database platforms. Also Oracle running under UNIX is a scenario which runs without any troubles. This allows to scale database power as needed.

Communication protocols and techniques between layers

The following table illustrates the application architecture.

	Technique	Comments
Client (Presentation Tier)	Common Language Runtime and XML Parsing capabilities	CLR and IE. Installation of .Net Applications through simple copy. No Registry Entries needed.
Communication	XML/ SOAP over HTTP, optional HTTPS and PKI Certificates	Uses standard Web Protocols over HTTP(S), single Port., Stateless.
Application Logic (Middle Tier)	Common Language Runtime and XML Parsing capabilities, IIS 6.0 as hosting Service	Windows 2003 Server as hosting platform offers additional scalability using “Web Farm” techniques, if needed.
Communication	.Net intrinsic database connection capabilities, optional IPSec	SQL Server and OLE DB connectivity available.
Database (Data Tier)	Database Server	SQL Server. Optional Oracle, DB2 and all Databases with existing OLE/DB Drivers

Typical Microsoft Components in a .NET Environment

With the above understanding about N-Tier Applications, the vision of integrated applications becomes more understandable. In practice, different products are used to achieve the goal of optimal productivity and integration.

Client Side

- Windows XP SP2
- Microsoft .NET Framework 1.1
- Microsoft Office 2003 Professional

Server Side

- Windows Server 2003 (Active Directory, DNS, DHCP, and other)
- Microsoft SQL Server 2000 SP3 (Database and BI Platform, Data Transformation Services, Reporting Services, Analysis Services)
- Microsoft Office SharePoint Portal 2003 (Intra/ Extranet)
- Microsoft Exchange Server 2003 (E-Mail, Calendar, Tasks and other)
- Microsoft ISA Server 2004 (Internet Security Acceleration, Firewall, WebCache, VPN, Server for DMZ)
- Microsoft BizTalk Server 2004 (Integration of other Systems/ Applications, Communication)

On the following chart we see, where the Internet Information Server (IIS) is located (it is the server in the middle, labeled with IIS, Webservices, Sha-

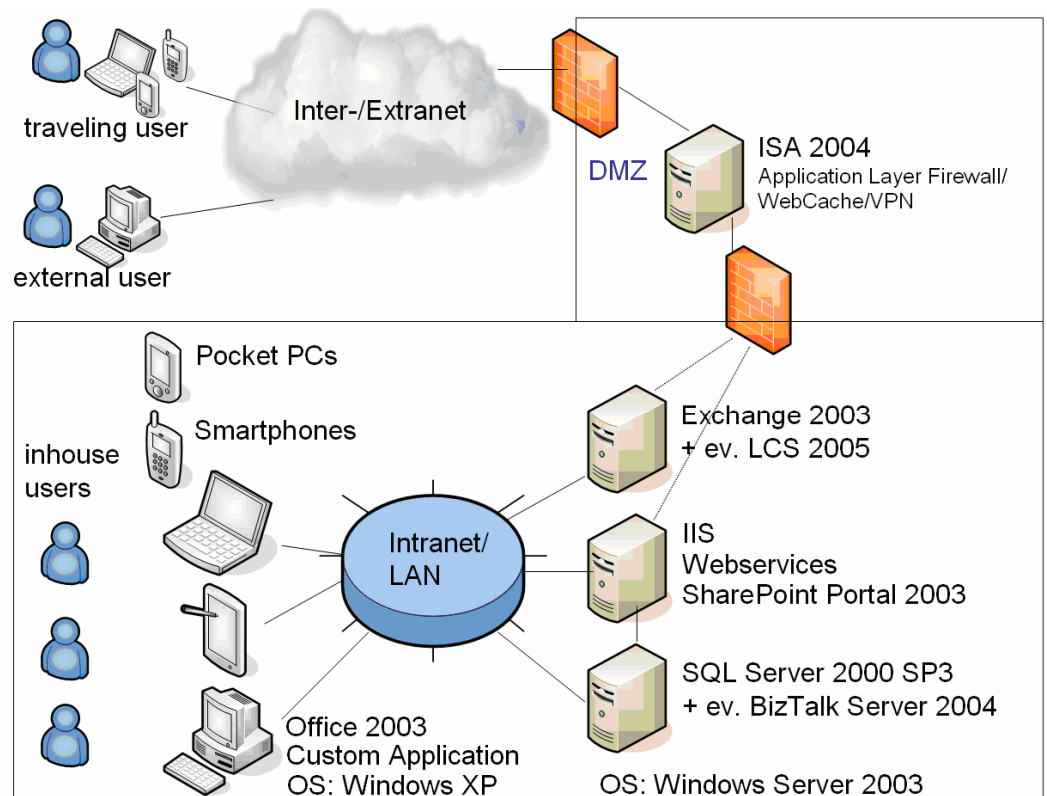
rePoint Portal 2003). It not only hosts the Webservices for the custom application, but also the whole SharePoint Portal Infrastructure.

Depending on the individual needs and the current architecture, the different Server Products can be installed on one or multiple servers. Questions like work load, scalability, security and main architecture are considered here to find an optimal deployment scenario.

The custom application in a typical scenario is a Windows Application which can be used by in-house users as well as travelling users. For external business partners, usually a Web frontend or an Office Document based approach is envisaged. Also here, the concrete needs define the chosen detail architecture.

The integration of custom applications with Office and SharePoint as well as with SQL Server and BizTalk is all done within the .NET platform. The development environment and especially the programming language C# offer exactly these huge benefits we need for professional application development.

For the development of Custom Database Applications, we use our own development Framework described in the following.



Additional .NET Productivity Framework (VDX)

At a first glance, the development of Database Applications in the Microsoft.NET environment seems to be simple enough. If your scope is the development of standardized and easy to maintain software within a team of peer developers, a lot of details have to be addressed and many concepts have to be implemented practically. Only with such an approach you can successfully implement your projects within your team. We walked through this process over the last few years and offer the result in form of an amazing new framework for the multilingual database application developer using the Microsoft Visual Studio.NET development environment.

Already within a project of only a few developers, the use of a proven framework pays off. Thanks to the complete documentation, detailed tutorials, a technical reference online help file as well as the VDX Sample Application, the knowledge can be built efficiently and be leveraged into productivity.



Visual Dotnet eXtensions (short: VDX) is our new, extensive Application Framework that makes Microsoft Visual Studio.NET Database Application Development easy! VDX is not a collection of "eye catcher controls", but a well designed, complete Database Application Development Framework. The VDX Framework consists of a Client- and a Server-Framework. Both framework components can be used together for a huge productivity gain during the development of Windows Applications. For the development of Web Applications, the server framework can be used independently from the other framework components.

The VDX Framework allows us to reach higher application implementation results in a shorter time by increasing the software quality thanks to standardization.

Conclusion

The .NET Framework is a radically more powerful and much easier way how multi tiered software has to be designed and deployed. Decisions regarding the implementation of pure HTML based clients or WinForm Clients using standard web protocols will have to be reconsidered. At least then, when a well known group of users will access the application using standard web protocols.

The fact, that .NET Applications will scale thanks to it's stateless Web Services based architecture opens the door to new applications which have not been possible to architect using standard client/server architectures.

The database independence offers an additional level of scalability, since no proprietary decisions have to be taken early during the development cycle. Also on the database level, "scale ups or outs" can occur whenever the need arises.

All clients which have the common language runtime installed, are able to run .NET Applications. No additional software components, no ActiveX Controls and no Registry Entries have to be made to deploy a .NET Application. A tremendous benefit in today's complex IT environment.

Thanks to it's modular design, a .NET Application can much more easily be adapted to severe security standards. User authentication can occur once on the middle tier server. Since no client can access the database directly, the risk related to client server solutions which explicitly expose data, does not exist in that form.

A system with lots of interfaces benefits on the long run from the XML approach of communication. This offers third applications to access the same web services in a secure and easy way. The amount of interface and infrastructure work can potentially be reduced.

As additional information regarding the benefits of our database application framework, we have more detailed information available on our web site.