Clientele CRM.NET Architecture Whitepaper

epicor
Software Corporation
Think Ahead. Stay Ahead.™

Disclaimer

Table of Contents

Clientele CRM.NET Architecture 1

Introduction: Why talk about architecture? 1

What are Web Services? 2

Architectural Models 2
Why Web Services? 3

Technology Foundation 4

What is Microsoft .NET? 4

Why Microsoft .NET? 5

Business Requirements 5
Rich User Experience 5
Accessible 6
Reliable and Scalable 6
Affordable 9
Flexible 9
Connected 10
Focus on Building Application - Not Infrastructure 11

Clientele CRM.NET Architecture 12

The Client 12
Communications 13
The Server 14
The Customer Self-Service Portal 14
Customization 15
Integration 17

Conclusion 19

Clientele CRM.NET Architecture

Introduction: Why talk about architecture?

When a company is selecting any piece of enterprise software there is invariably someone assigned to “look under the hood” and evaluate the application’s architecture. While this is a critical step in the evaluation process, many organizations do not have a clear understanding of how their short and long-term application needs can be directly impacted by an underlying architecture. Non-functional requirements like reliability, availability and scalability (and a host of other “ilities”) need to be considered in the context of other functional requirements such as customization and integration in order to properly evaluate any architecture. Unfortunately, most software vendors would rather not discuss how their architecture addresses these key requirements and instead focus on how “cool” their technology is or how using it will enhance the evaluator’s career. Now make no mistake, we think our new CRM.NET architecture is very cool. However, like our many customers, we also share the belief that being cool is simply not enough – which is exactly why the number one requirement for the new Clientele CRM.NET architecture was that it be *practical*.

This white paper is intended to serve as a foundation for understanding how Clientele’s next generation application architecture has been designed from the ground up to provide organizations with a dynamic, flexible, and robust CRM solution that shatters traditional enterprise software boundaries. To accomplish this goal it will be necessary to first take a look at several different architectural models and examine the underlying technologies of each. This will provide the perspective needed to better appreciate why Clientele selected the resulting architecture for CRM.NET.

If you are unfamiliar with Clientele, it is a complete customer relationship management (CRM) solution for small and medium enterprises and small to medium-sized divisions of larger organizations. Clientele provides integrated sales, marketing and customer support functionality that help organizations acquire, retain and grow profitable, long-term customer relationships. It supports remote users, provides automation of repetitive tasks, and offers Internet access. Clientele enables you to be knowledgeable, responsive and proactive in communicating with your customers. Clientele works the way you do, providing powerful searching, customization and integration capabilities.

What are Web Services?

Architectural Models

When starting a discussion regarding enterprise software, one should first understand some common software architectures. There have been many architectural models in the history of computing, but three in particular are important for our discussion of business applications. These models may be known to you by various names. We apologize if the terminology you are most familiar with is not used.

The first model is that used by mainframe computer software. In this model, the entire application resides on a single machine. Input and output is accomplished through the use of terminals. The structure of this architectural model is monolithic. Monolithic means that the application is, for purposes of deployment, a single program that runs on one machine. Applications using this architecture model also run on mini-computers and it is the architecture for stand alone PC applications. Over time there have been variations on this model for multi-processor machines, but the monolithic nature of the application remains – the application is a self-contained unit with all processing happening in the application itself. While some would consider Web-based applications a new model, we consider it to be a variation of the monolithic model.

The second model is the client/server model. In this model processing is no longer monolithic. Some of the application resides on the client and other portions on the server. There are variations of the processing mix between client and server, but in all cases both the client and server are necessary for the application to function. While this model added flexibility in application design – the client and server remained tightly coupled. However, one advantage of this model is that you can create more than one client to interact with the server. Many companies did this to support multiple client platforms (i.e., Windows, Mac, UNIX, etc.)

The latest model, which has received an incredible amount of press, involves "Web services." Web services are self-describing, self-contained, modular applications that can be mixed and matched with other Web services. Web services are Internet applications that fulfill a specific task or a set of tasks and that work with many other Web services in an interoperable manner to carry out their part of a complex workflow or a business transaction. Web services interact in a language and operating system independent manner – reducing the complexities of integration. Web services based architectures are considered loosely coupled and modular. As a result, a Web service can be replaced with a new one without affecting the client or other Web services that talk to it. Web services read and write eXtensible Markup Language (XML) based on the business logic embedded in them. The Web services model encourages breaking server-side functionality into smaller pieces that exchange XML using Simple Object Access Protocol (SOAP).

Why Web Services?

As far as the world of computing goes, Web services should prove to be as significant a milestone as any other to date. Unfortunately, as important and revolutionary as Web services are, the basic concepts surrounding the technology still remain one of the most fundamentally misunderstood. Many people tend to think Web services are limited to the publishing of software services to the Internet. While this is certainly an important role that Web services will fill, it is only a small part of a much bigger picture. At the Clientele Group, we believe Web services represent a significant and fundamental evolutionary change in the way that all distributed systems will be created. In fact, we are so convinced that Web services will be the standard for the distributed systems of tomorrow that we decided to deliver on that future – today. What does this mean? It means Clientele customers will be among the first organizations anywhere to fully exploit the many benefits of a pure Web services-based application architecture. It also means that as their businesses grow and expand, they can rest assured their CRM solution can grow and expand right along with them.

Web services also provide the potential for creating interfaces for integration that were previously very difficult due to proprietary interfaces (APIs) and language specific data structures. By using XML (which is text based) to share information and SOAP to communicate in a way that is programming language independent, it is possible to create Web services that talk to other Web services, whether built by other vendors or created internally. Web services introduce the possibility of creating an application from best of breed services provided by multiple vendors. While the applications necessary have not yet been developed and the standards need to be finalized – Web services set the stage for new paradigms of software deployment and use.

One key advantage of Web services is a new level of interoperability (another important "ility"). Because Web services interact with each other or with clients via SOAP and XML, it doesn't matter what language your Web services are written in (C# or Visual Basic or Java) or what platform you run (Windows or Linux or UNIX) – they can talk to other Web services! This level of abstraction from operating systems and programming languages is critical to understanding the possibilities introduced by Web services.

Another advantage of Web services is that they do not include the user interface. Web services interact with each other (as mentioned in the previous two paragraphs) and with user interfaces. Note that we use the word "interfaces" in the plural. A single Web service can support multiple user interfaces – PC-based and other rich clients, browser clients, small form factor clients, etc. Web services do not force a particular user interface upon application vendors.

That fact that Web services function over the Internet is also important. Web services use the most ubiquitous networking transport available. If your server has access to the Internet, then the Web services that reside on it can talk to other Web servers anywhere on the Internet. This allows increased componentization of applications with minimal concern about where components are located and how to connect to them.

Finally, Web services are expected to save companies money. According to Gartner, for example, Web services will drive a 30 percent increase in the efficiency of IT development projects. According to Forrester Research, Web services "crush" the cost of business interactions by replacing manual communications to save time and money, cutting the cost of connecting to partners, making internal services available across departments and geographies, and enabling new kinds of business collaboration.

Technology Foundation

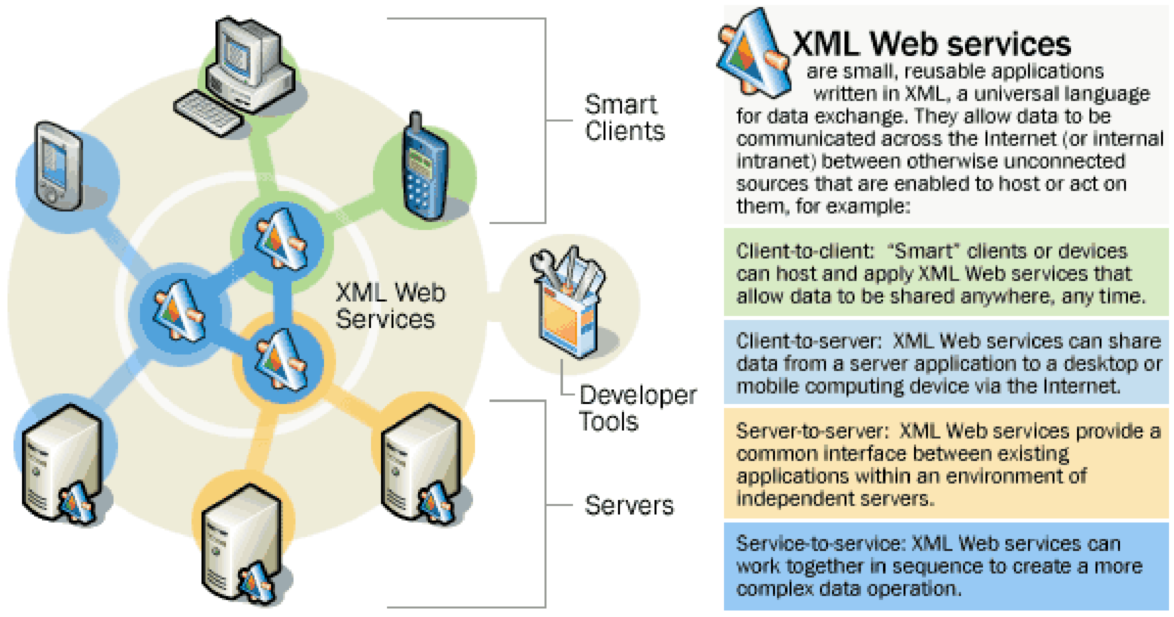
What is Microsoft .NET?

Once you have an appreciation for the power of the Web services architectural model the question becomes, "How do I get there?" There are many companies now providing development environments and servers for building Web services. Many vendors are basing their environments on Java with the goal of having a single language that runs in almost any environment. One vendor providing an environment for building Web services is Microsoft. Microsoft's .NET Platform is a complete set of products including development tools, a common language run-time (CLR), and a variety of servers (operating system - Windows, database – SQL Server, Web – Internet Information Server and integration - BizTalk).

Microsoft defines .NET in this way:

> Microsoft® .NET is a set of Microsoft software technologies for connecting your world of information, people, systems, and devices. It enables an unprecedented level of software integration through the use of XML Web services: small, discrete, building-block applications that connect to each other—as well as to other, larger applications—via the Internet. .NET connected software delivers what developers need to create XML Web services and stitch them together. The benefit to individuals is seamless, compelling experiences with information sharing. (© Microsoft Corporation 2002)

For more information on Microsoft .NET we recommend visiting http://www.microsoft.com/net/defined/default.asp. The following diagram is from Microsoft and helps describe the .NET Platform and its pieces. This diagram and further descriptions can be found at http://www.microsoft.com/net/defined/whatis.asp.



XML Web services are small, reusable applications written in XML, a universal language for data exchange. They allow data to be communicated across the Internet (or internal intranet) between otherwise unconnected sources that are enabled to host or act on them, for example:

Client-to-client: "Smart" clients or devices can host and apply XML Web services that allow data to be shared anywhere, any time.

Client-to-server: XML Web services can share data from a server application to a desktop or mobile computing device via the Internet.

Server-to-server: XML Web services provide a common interface between existing applications within an environment of independent servers.

Service-to-service: XML Web services can work together in sequence to create a more complex data operation.

Why Microsoft .NET?

Business Requirements

The Clientele Group of Epicor Software Corporation decided to use Microsoft .NET to build its next generation CRM applications based on Web services. But why?

This section attempts to address why the .NET platform was chosen by looking at what Epicor sees as the requirements when small to medium enterprises are looking at enterprise applications such as CRM, and how Microsoft .NET fulfills those requirements. Additionally, we will examine another reason for using .NET that benefits both Epicor and our customers.

Here are the main customer requirements for architecture that we will address:

- Rich user experience
- Accessible
- Reliable and Scalable
- Affordable
- Connected
- Flexible

Rich User Experience

Usually discussions of user experience and ease of use are not included with architecture. User experience and ease of use are based on the design of navigation, visual cues and other techniques which are generally independent of architecture. Yet architecture can influence user experience and ease of use. One example is that of a Web-based architecture that is designed only for a browser-based user interface. The use of a browser-based user interface places certain restrictions on the types of navigation that can be used. It also forces consideration of how standard browser navigation features (forward and back buttons) will affect the user interface and workflow.

A Web services architecture provides the powerful ability to have multiple user interfaces interact with the same Web services. Microsoft .NET reinforces this by allowing you to build both rich client and browser-based interfaces in the same development environment – Visual Studio.NET (VS.NET).

While browser-based user interfaces have gained significant popularity in recent years, the Clientele Group did not find customers demanding their CRM user interface be browser-based (vs. using Windows forms). What we found was that customers really desire robust functionality presented in a manner that supports their workflow and that is easy to navigate.

As a result, the Clientele Group chose to build a rich client – something Microsoft is now calling a “smart client” – using VS.NET. This choice makes sense for our customers who are looking for advanced functionality with intuitive workflow and simple navigation. Another compelling advantage of using a smart client is the ability to leverage client-side processing and thereby reducing the number of server “round trips.” This efficient use of bandwidth improves performance and the user experience.

This is not to say that browser-based interfaces don't have their place. For applications such as customer and partner portals, browser-based interfaces make particular sense as they can be accessed from any HTML compliant browser. Clientele is using a browser-based interface for its customer self-service portal, another native Web services application built on the .NET platform. The Clientele portal is an excellent example of how .NET allows reuse of Web services by different user interfaces, as the portal re-uses Web services developed for the core CRM application to provide customers access to data about themselves.

For more information on smart clients see http://www.windowsforms.com/whitepaper/whywindowsforms.aspx.

For an independent analysis of smart clients see this article by the Gartner Group: http://www.gartner.com/reprints/microsoft/104982.html.

Accessible

Using a smart client to provide rich functionality does not eliminate the need companies have for easy, flexible access to CRM information. This need is often expressed as needing "Internet-based applications." What companies are saying is that they need access options to support a variety of user types. These users may be on a local area network (LAN), or they may be part of a remote office with access to the corporate network via long distance dialup lines, frame-relay, or VPN (wide area network – WAN). There are also times when a user may be remote and need to connect to the corporate network from home or a hotel room. In the case of the remote user, the Internet is usually the easiest way to connect. In all these cases, providing Internet access to the application is a solution to the real problem of providing easy access to a variety of users.

One of the benefits of the .NET platform is the ability to build applications based on Web services that have either a smart client or a browser based user interface (or both!). This capability has allowed the Clientele Group to develop a smart client application that uses Internet standards as the transport for talking to Web services that serve up data, take input, perform validations and update the database. This means that Clientele CRM.NET will work in an Intranet environment, an Internet environment or an Extranet environment (or all three at once).

The last type of access requirement is that of the disconnected mobile user. This type of user needs to take a portion of the data with them and then work while disconnected. This is particularly common for sales personnel or field service technicians. One common drawback of enterprise applications that use a browser-based interface is the simple fact that they do not function in a disconnected environment. The smart client approach provides a natural user interface for both connected and disconnected users and allows Clientele to support both modes of operation for any user of the system – using a single user interface.

Reliable and Scalable

Downtime is very expensive. All companies want applications that work and keep working over long periods of time. Reliability and availability is primarily a factor of the quality that a software vendor builds into their solution. However, some architectures lend themselves to increasing reliability and availability by allowing for redundancy and failover. The Microsoft .NET platform, which includes a family of servers and is based on Web services, provides a framework for an IT infrastructure that includes redundancy and failover of Web services to keep your CRM application running even when individual hardware failures occur.

Through the use of industry standard hardware and software redundancy and failover techniques, the CRM.NET suite of products can be implemented to the highest standards of reliability and availability with no single points of failure and maximum uptime. This insures your users will always have access to their data whenever they need it.

As we talk to potential customers, it is rare to encounter small and medium enterprises that have no intention to grow. This is not surprising given that one of the primary goals of any CRM solution is to help grow business. But the last thing companies want to do as they grow and expand their reach is to replace a critical system such as customer relationship management. This leads to questions about scalability.

Once again, Microsoft .NET and the Web services architecture provides a simple answer to scalability that also addresses another important concern - low total cost of ownership (TCO). With the CRM.NET architecture, instead of scaling up by buying bigger and much more expensive hardware, you can choose to scale in two ways. The first is called scaling vertically, which is simply upgrading the current hardware in place by adding additional CPUs, more RAM or adding additional hard disk space. The second method is called horizontal scaling, which is achieved by adding additional machines and then distributing processing requirements across all machines. The modular nature of Web services creates an environment in which you can quickly, dramatically, and cost-effectively scale up your enterprise application by upgrading existing hardware or adding additional machines and then distributing the application across multiple servers.

With Clientele CRM.NET, our goal is for the majority of initial implementations (typically 50 users of CRM or less) will be able to deploy the entire application with all optional components, including Microsoft SQL Server and the customer database, on a single server machine running Windows 2000 or, in the future, Windows .NET server. As a customer's environment grows and the number of application users increases they can choose to add an additional server and distribute Clientele's Web services, SQL Server and the database accordingly. If future application demands continue to grow, additional servers can be added and Web services distributed into a standard Web farm configuration. Three possible deployment configurations are provided here as examples. There are other deployment options available capable of supporting the most demanding of environments.

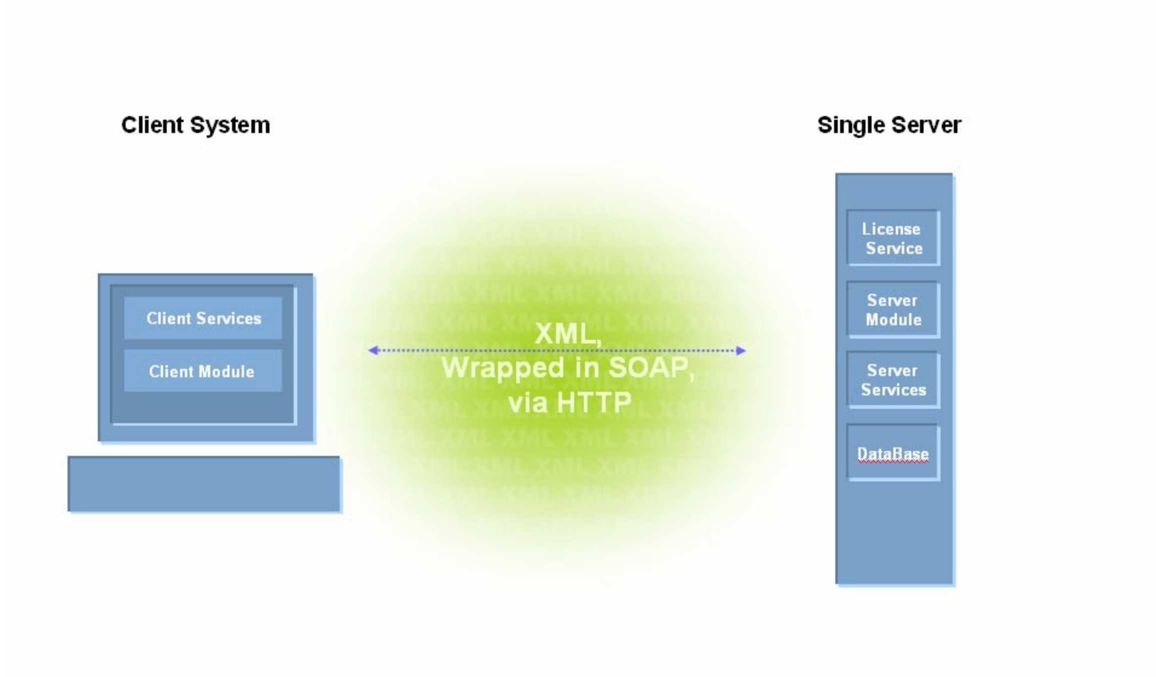


Figure 1 - Single Server Deployment of CRM.NET

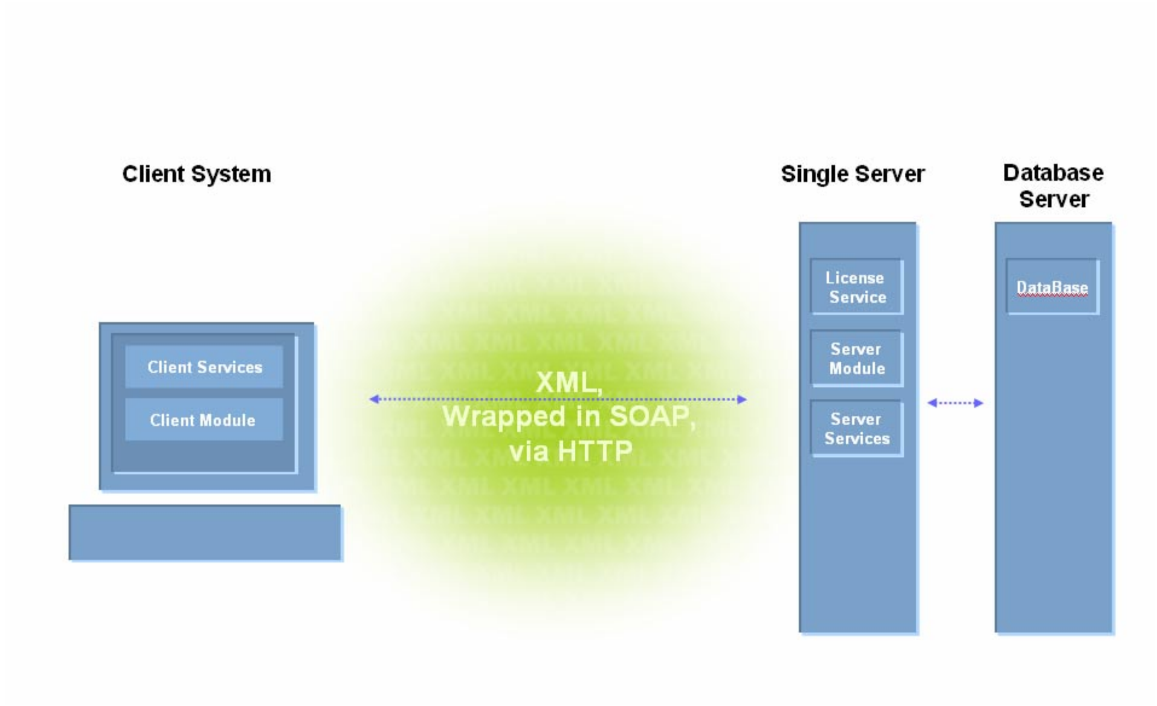


Figure 2 - Two Server Deployment of CRM.NET

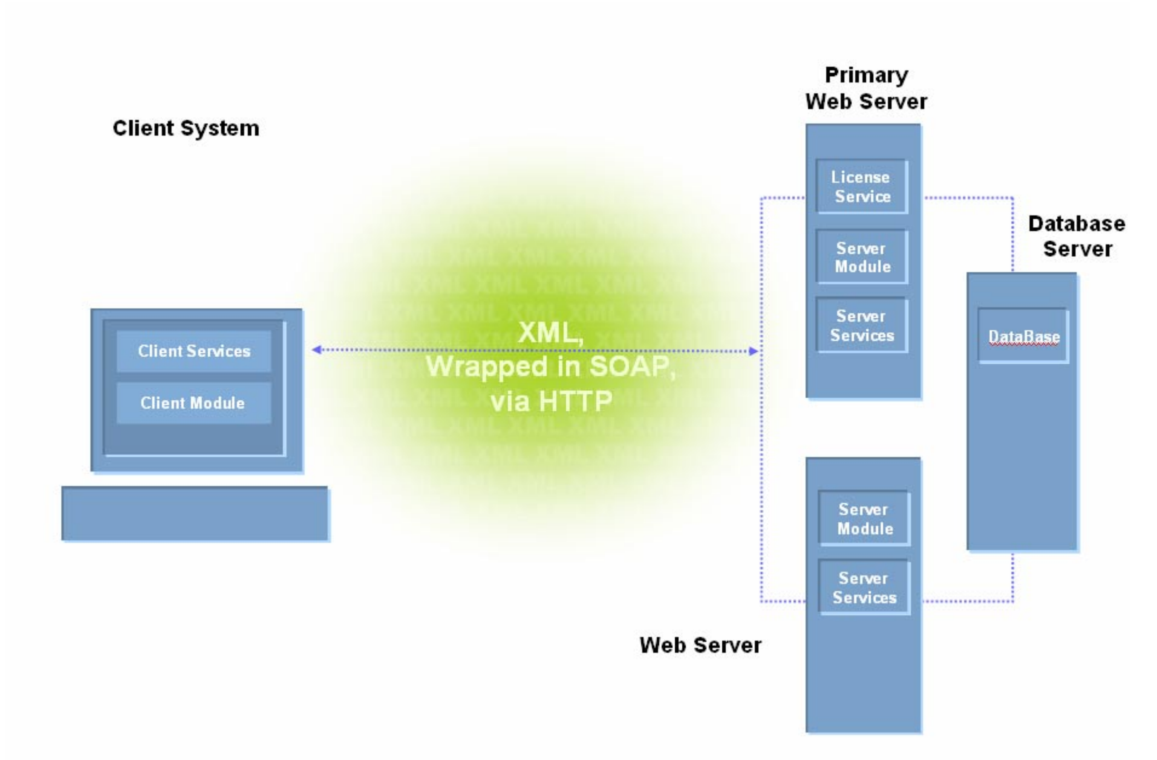


Figure 3 - Multiple Server Deployment of CRM.NET

Affordable

There are many variables in the total cost of ownership equation. Microsoft's .NET platform helps companies lower the total cost of ownership for enterprise software by providing several benefits that keeps costs in check.

First, as discussed in the previous section on scalability, enterprise applications such as Clientele CRM.NET will be able to initially deploy on a single server and then scale by adding inexpensive hardware – rather than replacing the initial server with one that is more expensive.

Second, the use of standard Microsoft servers will make finding skilled and experienced administrators easier and less expensive. Microsoft helps by providing certification programs – so you can seek out certified professionals. Applications based on the .NET platform will use Microsoft servers such as Windows Server, Internet Information Server, and SQL Server.

Third, the use of Microsoft's VS.NET development environment as the standard customization tool means finding developers with the appropriate skill set for customizing your enterprise application will be much easier compared to other enterprise applications using proprietary toolsets. Thanks to the Common Language Runtime, part of the .NET platform, organizations can leverage their existing investment in development skills by allowing developers to use any .NET compatible programming language. This means that a software vendor can now develop their solution in one language (like C#) and allow customers to customize the solution using another language (like VB.NET). In addition, vendors other than Microsoft have introduced .NET versions of their languages – making it even easier to find qualified developers.

For more information on other .NET compatible languages see:
http://msdn.microsoft.com/vstudio/partners/language/default.asp.

Flexible

In the previous section we talked about how customization using standard tools reduces the total cost of ownership for an enterprise application. This section focuses on other advantages of using the standard tool for Microsoft .NET development – Visual Studio.NET.

There are two important types of customization for enterprise applications such as customer relationship management. The first is the ability to change existing capabilities – adding or subtracting fields, re-arra nging fields to support your workflow, or changing the business rules to ensure your processes are followed. With VS.NET as the customization environment, application vendors can build applications that allow for the above changes without exposing the original source code or creating unnecessary difficulties with future upgrades.

All VS.NET languages are object oriented. One of the central characteristics of object-oriented programming is inheritance. Simply stated, applications are built as code objects. These code objects are called classes. Code classes are built in layers with the foundation layers referred to as base classes. Each succeeding layer can inherit the functionality of the layer below it. The advantage of this is that the customer and vendor can work on different layers simultaneously. As a software vendor, we can continue to add functionality on the base classes without interfering with customizations executed on successive layers. Because .NET has a "multilingual" compiler, these simultaneous customizations need not even be in the same language.

The second type of customization is when the purchaser of enterprise software chooses to extend the functionality of an application. Again, the object-oriented nature of VS.NET provides advantages in this area. In particular, customers can inherit from base classes for common functionality and user interfaces. These base classes provide a dramatic head start in creating new application functionality that integrates tightly with the rest of the application being extended. New forms or Web services follow the same model and know how to interact with existing Web services in the application. This inheritance model will significantly reduce both the amount of time and resources needed to build additional functionality.

Additionally, the Clientele CRM.NET model of publishing "events" allows companies to create new services that listen in on application events and perform additional processing based on those events. For instance, a company could add a Web service to perform notification to another application or start processing in another application when a specific event occurs in the CRM system.

The Clientele CRM.NET Framework exposes a consistent event model for easy integrations to external systems. This decoupling also allows for isolation of software vendor code from integrations, making it much easier to integrate services into Clientele by listening in on application events and performing additional processing based on those events. For instance, a company could add a notification to a Shipping System when a customer's address is changed or start processing an invoice in a financial application when a billable support call is created.

Connected

One of the most important parts of the Web services architectural model is the potential for integration between different applications. By using XML as the format for sharing data, and SOAP to communicate in a way that is programming language independent, it is possible to create Web services that talk to Web services built by other vendors, or that have been created internally.

Integration between Web services is not automatic. XML – the text-based language for exchanging data - still needs standards. Like HTML, XML uses "tags" to designate what a piece of data means. Today, organizations like WS-I.org[1] (Web Services Interoperability Organization) are working together so that the tag for "Customer Identifier" is used consistently across applications. Creation of standards will allow vendors to create applications that use standard XML tags – making integration even easier.

XML is only part of the integration equation. The second part is the "methods" exposed for interaction. Today there are no standards for naming these methods and what functionality should be exposed to other Web services. For instance, if two Web services are to talk to each other directly, they need to know the names of the methods available and what they do.

While integration servers are not required to link to Web services, they can help reduce the amount of work necessary to integrate applications. These integration servers, such as Microsoft's BizTalk, allow a vendor to create a "connector" to the integration server. The connector contains the definition of the module interfaces and XML streams. Using the integration server, it is possible to map two XML streams together and create rules for what to do when the connector is activated.

[1] Epicor Software Corporation is a founding member of WS-I.org.

So far we have discussed integration within a single business. But Web services have the potential for integrating with commercially published Web services. A technology called Universal Discovery Description and Integration (UDDI) provides a Web-based distributed directory that enables Web services to publish information on the Internet and discover each other, similar to a traditional phone book. Companies can use a commercial UDDI directory or build their own internal directory. The goal of UDDI is to allow businesses to connect to Web services provided by external business partners.

Focus on Building Application - Not Infrastructure

The last advantage of Microsoft's .NET platform we will discuss is an indirect benefit for buyers of enterprise software. It is not a standard "requirement" for architecture that most customers would name – but it does have potential for significant impact on our customers. The advantage is that the vendor building the application can focus on the application – not the platform. Many enterprise software vendors build their own application platforms. Building a platform takes many resources and must be supported by fixes, patches, and improvements over time. These resources are expensive and while they make functionality possible – rarely provide capabilities directly experienced by the end user.

By using the .NET platform, software vendors can focus on what they do best – building application functionality for their target audience. By relying on the hundreds of software engineers working on the .NET platform to provide a base from which to build functionality, enterprise software vendors are able to focus on capabilities that improve and automate important business processes. Building on this platform frees up resources to focus on the customer experience and improving user interfaces and capabilities.

As Microsoft and other vendors continue to extend the .NET platform, additional functionality will automatically become available to Clientele and our customers. Customers will be able to take advantage of these enhancements through VS.NET. For example, third party development tool companies will create new controls to enhance the user experience; legacy applications can be wrapped up and exposed in a standard way for the .NET platform to extend their lifespan; security software companies provide encryption techniques to ensure data security; network systems vendors provide load balancing hardware and software to increase reliability of .NET based systems.

Clientele CRM.NET Architecture

The Clientele CRM.NET suite is built on the Microsoft .NET platform. This section will explain the components of the architecture and how it is built using the .NET platform. A deeper discussion of customization is included as it is considered an important aspect of the architecture.

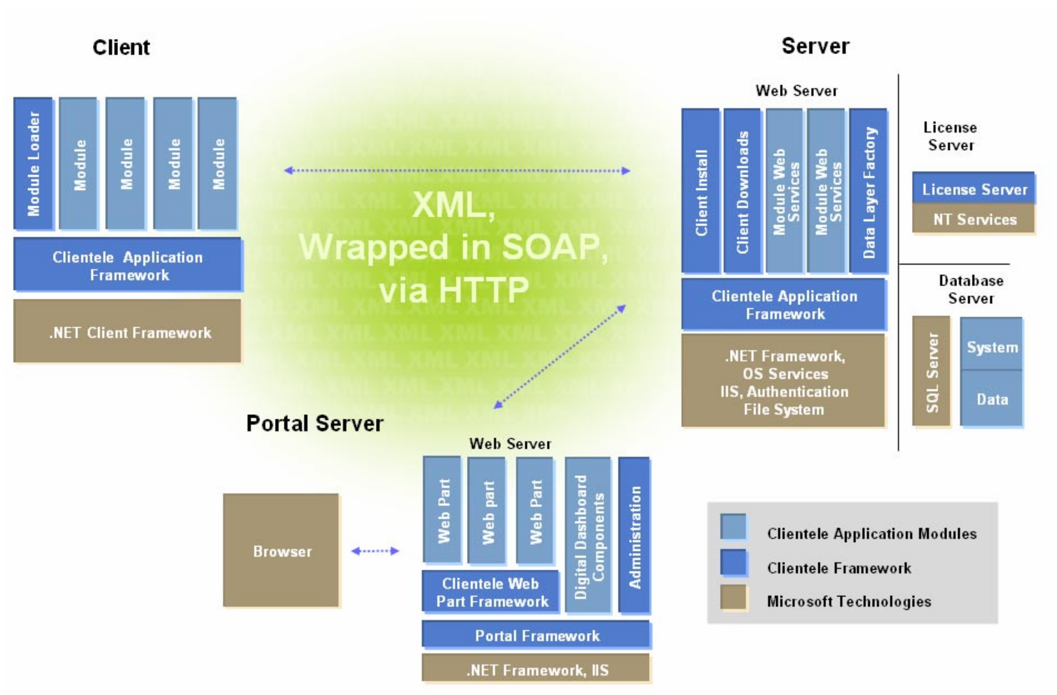


Figure 4 - Clientele CRM.NET Architecture

The above diagram shows the overall Clientele CRM.NET architecture. The diagram is color coded to show the portions of the architecture that are from Microsoft. The next three sections look at each of the major sections of the diagram – the client, the communication layer, the server and the customer portal.

The Client

Today many enterprise applications are being rewritten to support browser-based clients. These clients are generally considered "zero-footprint" or "lite" clients. One of the advantages of this model has been the ease of deploying new functionality. Updates are made to the server and afterward all pages delivered to the browser client automatically receive the changes. For Clientele we chose to provide the ease of deployment and accessibility of a browser-based model while simultaneously providing the extra capabilities and enhanced usability of a rich Windows client. In addition to the fact that a rich client can dramatically increase overall performance by performing a significant amount of processing on the client machine and reduce server round trips. Also, there are many user interface paradigms that are simply not possible in a browser-based application.

The Clientele Group has created a new rich client for Clientele CRM.NET using Microsoft's VS.NET and C#. The client requires the Microsoft .NET Client Framework which is available as a free download from Microsoft (now part of the standard Windows Update). Because the .NET Client Framework uses the CLR, or Common Language Runtime from Microsoft, it is possible to customize Clientele using a language other than C#, such as Visual Basic or C++. On top of the Microsoft .NET Framework we have built a lightweight layer we call the Clientele Application Framework. The Clientele Application Framework provides base classes for the client modules and forms to standardize the look and feel of Clientele CRM.NET and speed development by the Clientele Group, partners and customers.

Client modules contain both forms and code. Inheriting from base classes and forms provides our customers with the ability to create new functionality that, in general, behaves like functionality already in the system. By placing those shared capabilities in base classes and forms, those doing customization do not need to recreate that functionality each time they want to use it. This also improves Clientele's ability to change or add functionality quickly.

The client modules are designed to function independently but have the ability to interact. This loosely coupled model means that the absence of a module will not necessarily crash the system. However, certain modules are required for the system to function – for instance, the module for managing Customer identities is always required.

The client framework and module loader components are the only portions installed by the installer. Together they are less than one megabyte in size. When installed on the server, Clientele CRM.NET creates a Web page with download and installation instructions. When ready to deploy Clientele, the administrator can send out an email with a link to the Web page and tell users to follow the instructions. The page checks to make sure the user has the correct version of the Microsoft .NET Client Framework installed and directs them to the appropriate Web page if they do not have it. After running a small installation program, users are able to login. At login time the module loader interacts with the client download component on the server to provide automated deployment of module updates and customizations when each user logs in. After initial installation the module loader will download all needed modules. After that only updates, added modules, or customizations will be downloaded. Providing automated deployment of updates greatly reduces the cost of making changes and dramatically speeds up the process of distributing changes to users, both local and remote.

Communications

Communication between clients and the server in the Clientele CRM.NET architecture is accomplished using HTTP. This means that all clients talk to the server via the standard port used for all Web page traffic. Because of this model, the rich client is able to talk to the Web services on the server utilizing the Internet just like a browser-based application. Additionally, it deploys in a local area network or wide area network environment (intranet) and can be used in conjunction with a virtual private network (VPN) for added security.

Data is transferred between client and server modules as XML datasets and uses the simple object access protocol (SOAP) to make Web services module calls and send the XML data stream. Data in XML format is clear text and can be read by others on the Internet. For this reason we strongly recommend using secure sockets layer encryption (SSL) or a VPN or one of the emerging SOAP security techniques when having users connect via the public Internet. Many companies may choose to use these techniques even in an intranet environment for additional security.

While most companies will find the security of SSL and VPNs adequate for their needs, new standards are being developed that will increase the protection of customer data. A standards body with the backing of Microsoft, IBM and Sun Microsystems is working on Web services security standards. These security standards should enhance security beyond the use of SSL and VPNs mentioned above. Clientele will support these standards as they come available.

The Server

It is something of a misnomer to refer to "the server" in this architecture as there can be one or more servers in your Clientele deployment depending on your specific needs. However, one of the key benefits of this architecture is that all components can reside on a single server – including SQL Server and the actual databases used. As we discussed in the section on scalability, there are a number of ways to quickly and dramatically scale the application to meet even the most demanding environments. Our goal is to have the entire system running flawlessly on a single server supporting 50 concurrent users. This will allow many businesses to perform their initial deployment on a single server.

The Clientele Group has also written the server for Clientele CRM.NET using Microsoft's VS.NET and C#. The server is made up of native Web services that run on Microsoft's Internet Information Server. Just as it does on the client side – the Clientele Server Framework provides base classes to standardize the functionality of Clientele CRM.NET and speed development by the Clientele Group, partners and customers. By inheriting from these base classes customers can add or change business logic or Web services to Clientele. By listening for events occurring in the server framework, customers can easily integrate into other systems.

In keeping with the focus on building application functionality and not infrastructure, the server leverages Windows NT authentication. Clientele does not keep passwords in its database. Instead, when a user attempts to login, Clientele simply asks the network operating system to authenticate the user.

The Customer Self-Service Portal

With the Clientele Customer Self-Service Portal, your customers can quickly find answers to their problems, submit new incidents or check the status of an existing one, drill into service level agreement details, check the status of RMAs, or view their product information. In addition, customers can be allowed to personalize their online support experience by choosing exactly what they want to see and how they want to see it.

Unlike other self-service offerings, the Clientele solution was built on top of our new enterprise-class portal application server that is also built entirely on the Microsoft .NET framework (see architecture diagram for more details). Why did we take this approach? Because we understand that there is no such thing as a "one size fits all" approach to customer self-service. By basing our solution on a robust portal framework, our customers are free to extend and customize the self-service experience to fits the unique needs of their business partners and customers.

Our portal application server uses industry-standard web part technology, thereby supporting thousands of commercially available web parts. For example, Microsoft SharePoint Portal web parts will render natively in our environment. In addition, the Clientele portal framework has been extended to use ASP.NET user controls as web parts. The result is the capability to quickly create custom web parts that provide a seamless, integrated end-user experience.

On top of our enterprise portal server, Clientele has created a lightweight framework that provides native integration with Clientele CRM.NET applications. For example, the customer self-service functionality that is available for Customer Support 8.0 is a collection of web parts that take advantage of advanced capabilities provided by the Clientele web part framework. This means that your customers will now be able to interact with live data from the same Clientele server used by customer service representatives. If a change is made to a call record, that change is immediately available to an authorized customer online.

Because the portal application server is also a native web services-based architecture, integration with external applications is extremely easy. For example, no changes to code were necessary to integrate the portal with the new Customer Support 8.0 product. Communication between the two applications leverages the same web service interfaces that the Windows client uses.

Customization

Customization is a key requirement of enterprise software –regardless of the size of the customer. We built Clientele CRM.NET to be highly customizable. The customization model is object-oriented using Visual Studio.NET as the development environment. In this section we will outline the various ways to customize Clientele. All customization methods require VS.NET as the development environment.

The first type of customization in Clientele CRM.NET is the extension of existing client forms. In this case the developer will inherit from an existing form in Clientele and make modifications. The developer can change labels, hide fields and add fields to the form. VS.NET uses visual cues to show inherited fields on a form. The three screenshots below demonstrates first how the Clientele Group builds forms using a base form, and then how a customer could modify the form by inheriting from the published version in Clientele CRM.NET.

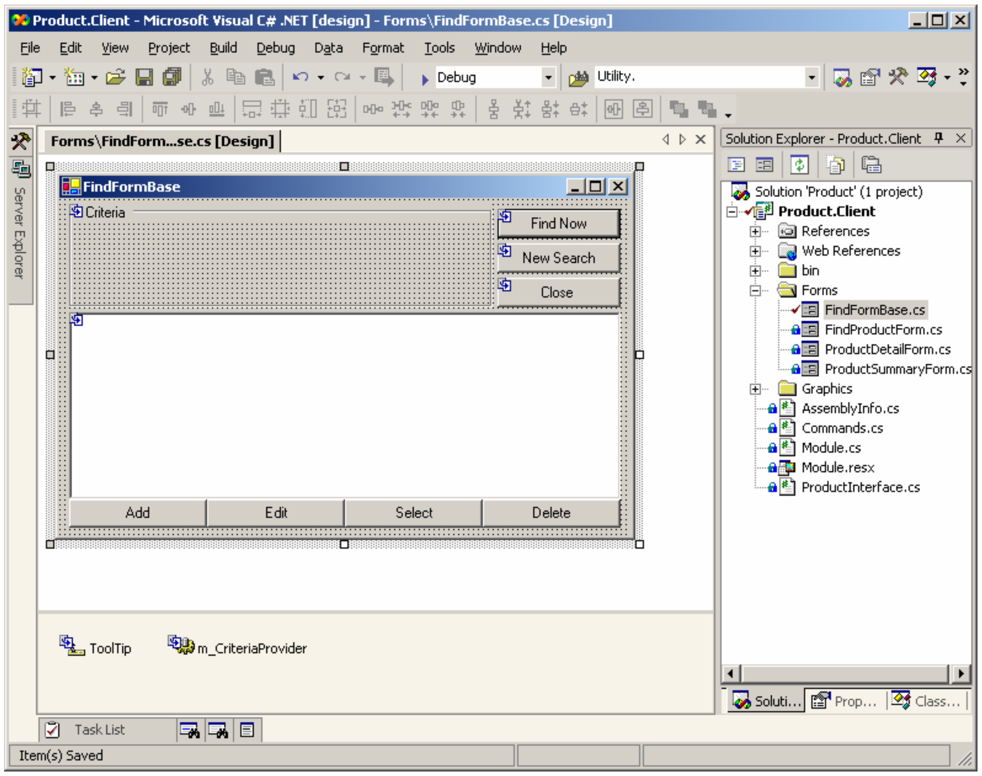


Figure 5 - FindFormBase from Clientele Application Framework

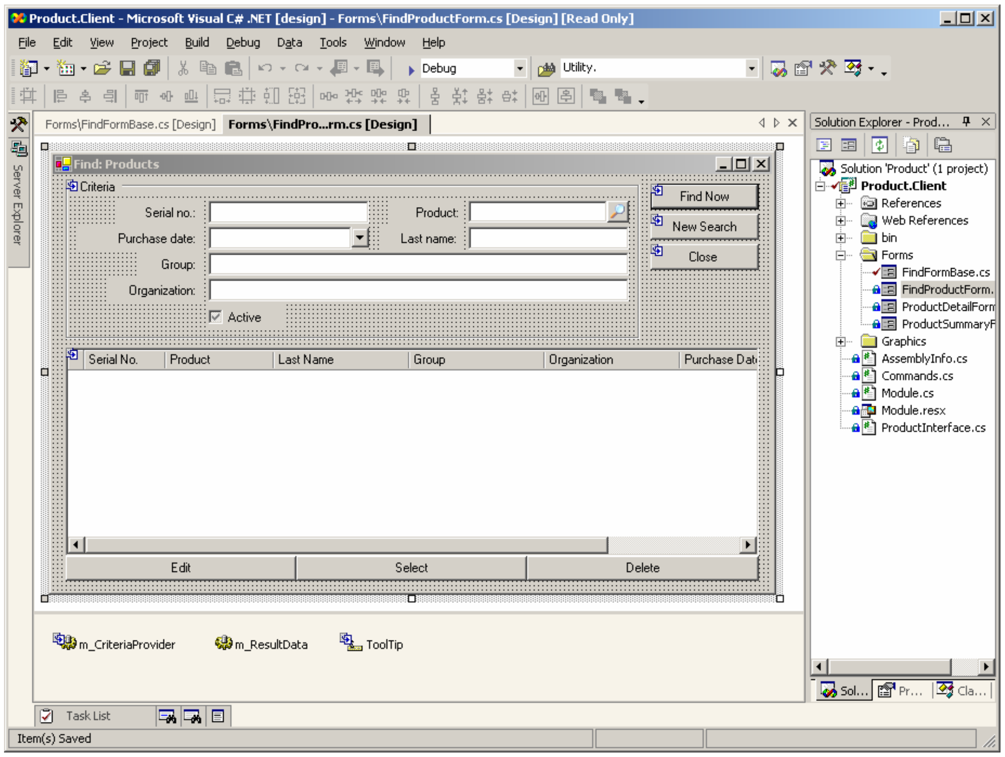
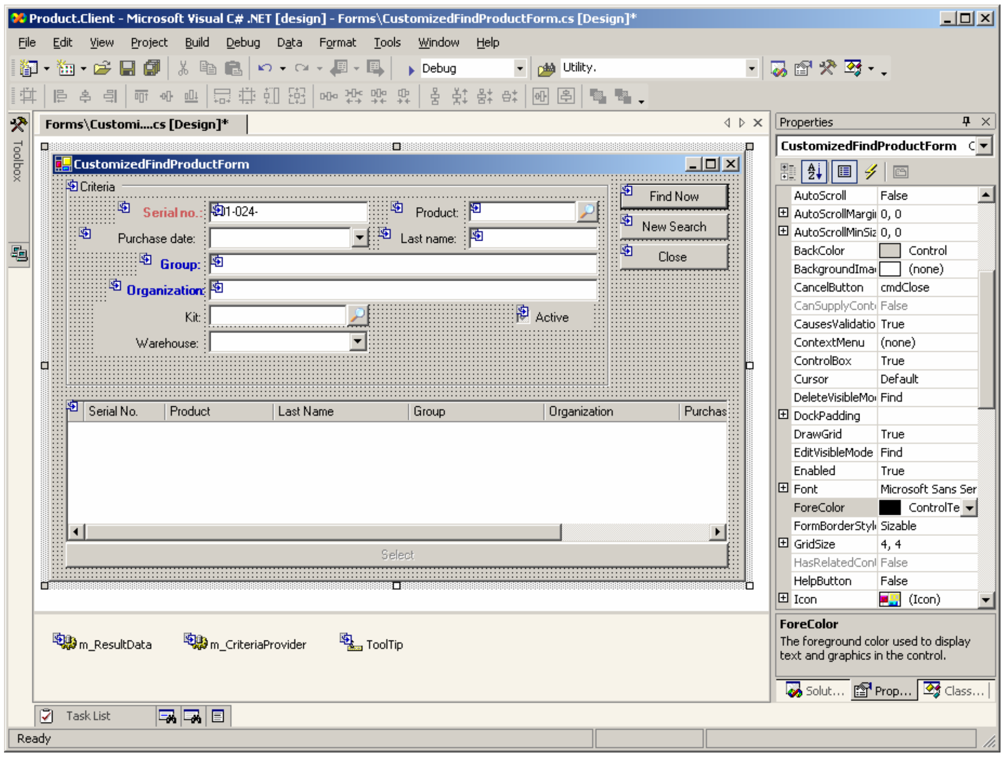


Figure 6 - Clientele Application Form. The blue boxes on the form indicate inherited controls.



AutoScroll	False
⊞ AutoScrollMargi	0, 0
⊞ AutoScrollMinSiz	0, 0
BackColor	▭ Control
BackgroundIma	▭ (none)
CancelButton	cmdClose
CanSupplyConte	False
CausesValidatio	True
ContextMenu	(none)
ControlBox	True
Cursor	Default
DeleteVisibleMo	Find
⊞ DockPadding	
DrawGrid	True
EditVisibleMode	Find
Enabled	True
⊞ Font	Microsoft Sans Ser
ForeColor	■ ControlTe
FormBorderStyle	Sizable
⊞ GridSize	4, 4
HasRelatedConl	False
HelpButton	False
⊞ Icon	(Icon)

ForeColor
The foreground color used to display text and graphics in the control.

Figure 7 - Customer Customized Form. The blue boxes on the form indicate inherited controls. Note how properties on inherited controls can be overridden.

The second type of customization is where the developer fully replaces (overrides) the original form with a new one he/she developed without inheriting from the original. In this case, the developer inherits from a Clientele base form (figure 5) and begins to build functionality (skipping figure 6 and moving straight to figure 7).

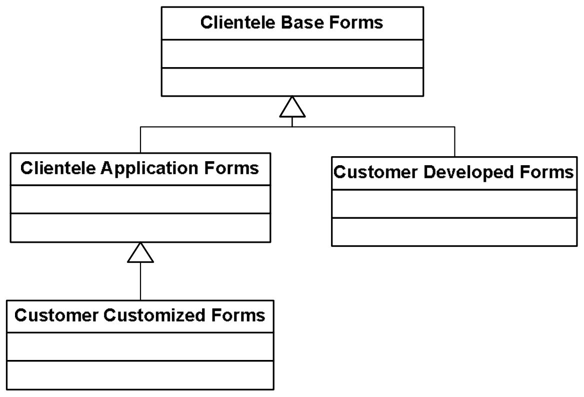


Figure 8 – The left branch of this diagram represents the first type of customization and the right branch represents the second type of customization.

The third type of customization is changing client logic. This is similar to the first type, but is based on inheriting code instead of forms. The developer can change or extend the logic in a module that inherits from that provided in Clientele.

The fourth type of customization involves modifying the datasets. Datasets define what data is made available to the client. If the developer adds fields to a table and to a form, the corresponding dataset must be updated for the new data to reach the client. The modifications are made using VS.NET. The definitions of the datasets are stored as text.

The fifth type of customization is modifying Web services. The developer can add new methods to the existing Web service or inherit from the original and modify existing methods. This is similar to the second type of customization, but the code runs on the server.

The last type of customization is to change the server business logic. This would include changing business rules to call external business processes (other Web services). This type of customization could be used to integrate Clientele with other applications.

Integration

While customization is about changing existing functionality or building new functionality within an existing system, integration is about linking functionality and data between two different applications or systems. Clientele CRM.NET leverages the power of Web services for integrating Clientele with financial, distribution, or manufacturing systems. There will also be opportunity to integrate Clientele with Web services that provide very specific functions such as tax calculations or address verification and standardization.

Just as we described several types of customization for Clientele CRM.NET, there are also several models of integration. These models are primarily based on the capabilities provided by a Web Services based architecture.

The first type of integration is not based on Web services. It is commonly used today as a way to link the data between two systems. In this type of integration an application is written or purchased that will take data from one system and copy it to another. There is work involved in doing this as most applications will not use identical data types and field sizes for the same data. This method has common because it is the simplest method. But it has risks. When data is inserted directly into the database, that data does not go through the same validation and business rules as data entered via the application. This increases the risk of having bad data. This method also tightly couples applications and increases the difficulty of upgrading either system.

Another type of integration relies on server events in the Clientele Server Framework. The Server Framework publishes Clientele events with pertinent information about what is being changed in the application. Subscriber applications can be developed to take action when particular server events occur. Clientele will rely on this capability to provide workflow functionality. Another example of this model is use of an integration server, such as Microsoft's BizTalk, to listen for Clientele server events. The integration server can be configured to talk to other applications via multiple methods such as Web services, APIs, etc.

The third type of integration is the one most commonly thought of when discussing Web services. Web services have the natural ability to talk to other Web services. Customers can build integration between applications by having a Web service in Clientele call a Web service in another application. The integration is made bi-directional by having Web services in the other application call Clientele Web services. We believe that most enterprise software will soon provide Web services as interfaces for building integration with other applications.

The fourth type of integration is to use the Clientele client to provide a user interface to legacy systems. Most commonly this will occur through the creation of Web services interfaces for legacy applications. Alternatively, the Clientele client can interact directly with the desktop for other integrations such as Computer Telephony Integration and integration to desktop email and calendaring applications.

The last type of integration is similar to the fourth, but the user interface is a Web part in the Clientele Portal. This could be any type of Web part including Microsoft Digital Dashboard parts, HTML Web parts, or XML data. Portal interfaces have the advantage of being able to bring together data from many different applications and systems (because each Web part is independent) and displaying them together in a single user interface.

Conclusion

Clientele CRM.NET is the next generation of Clientele, with Clientele's rich functionality moved to the .NET platform. Clientele embraces Microsoft .NET by using VS.NET to build an object-oriented client and Web services to create a complete CRM solution for small and medium enterprises and small to medium-sized divisions of larger organizations. Clientele brings the power of Web services to customer relationship management.

In the introduction we told you that an architecture needed to be *practical*. To us, practical means that we meet our target customers' key requirements. Clientele CRM.NET, built upon the Microsoft .NET platform, meets these key architectural requirements:

- A rich user experience through a smart client interface that provides the familiarity and ease of use customers need.
- Accessibility over a LAN, WAN, Extranet, Internet or Intranet – plus support for disconnected users.
- Reliability via server redundancy and failover.
- Scalable to hundreds of users through the addition of inexpensive hardware.
- Affordable to install, configure, customize and maintain.
- Easy integration with other applications using Web services.
- Highly customizable using an industry standard toolset.

Clientele Customer Support 8.0 is the first release of the Clientele CRM.NET suite, scheduled for release in the third quarter of 2002. The Customer Self-Service Portal 8.0 will ship shortly after. The Clientele Group will continue to move Clientele to the .NET platform, releasing Sales functionality in Q1, 2003 and Marketing in Q2, 2003.

If you are interested in learning more about the power of Web services and Clientele CRM.NET please contact us at 800-365-0912 or check us out on the Web at clientele.epicor.com.