Redesign of Stand-Alone Applications into Thin-Client/Server Architecture

Michel van der Vlugt
AXI B.V., Breda,
The Netherlands

m.vd.vlugt@mvlugt.com

Samuel Sambasivam
Azusa Pacific University
Azusa, CA, USA

ssambasivam@apu.edu

Abstract

One of the characteristics in software development is that software systems require changes once they are deployed in an operational environment. When the software is being used by the intended audience it is almost inevitable that errors are found, requirements change or new requirements emerge because of changes in the business processes. Depending on the nature of these modifications and the life-time of the software, the impact on the existing software will vary from simple error solving to complete architectural transformation.

In this paper the architectural transformation is presented of stand-alone applications, redesigned into thin-client/server architecture to improve the application's flexibility, interoperability, performance, distribution and scalability. The strategy proposed in this paper is a decomposition of the original application in which functionality is categorized and distributed in N-tier client/server architecture. The client application only contains the user-interface while the remaining functionality is split across multiple server applications.

The software that is subject for redesign in this paper is a recently developed Point-Of-Sale application, initially designed as a stand-alone application. The goal of redesigning the application is to reduce the front-end hardware requirements, improve the application's flexibility and make the application applicable for a wider range of usage. The redesign approach is evaluated by implementation of a proto-type Point-Of-Sale application, which has proven that an N-tier client/server Point-Of-Sale application is a feasible solution and leads to a very flexible and highly scalable application.

Keywords: Client/Server architecture; Legacy Applications; Middleware; .NET Remoting; Software reengineering

Introduction

Since the early eighties, at the time that personal computers where introduced in a computer industry that was dominated by mainframe computers, the technology of client/server computing started to emerge for sharing data and applications across a network of computers. During the evolution of client/server computing many technologies are developed to support this architecture and client/server still is a dominant architecture in computer industry. As compared to stand-alone applications and mainframe systems, advantages of client/server computing are flexibility, interoperability, performance, robustness, distribution and scalability.

The aim of this paper is to evaluate possibilities and propose a strategy for redesigning a stand-alone application to client/server architecture in order to adopt the architecture's advantages. The necessity for this project has come from a recently developed Point-Of-Sale (POS) application. This application was originally designed to operate in a stand-alone environment to ensure the application's availability, even in case of infrastructure failure. Although this design choice has resulted in a robust solution, the application requires a significant amount of hardware resources in order to obtain reasonable performance. These hardware requirements limit the usage of the application to high-end desktop computers while customers in retail business demand a more flexible and wider range of usage. Examples of customer demands vary from usage of network computers in order to reduce the costs of infrastructure, to usage of Tablet PC and Pocket PC as mobile POS units for assistance in peak seasons.

The flexibility of the POS application can be accomplished when moving to thin-client/server architecture as this will reduce front-end hardware requirements and opens new opportunities. The advantages of a client/server POS application:

- Less resources of POS hardware required;
- Flexibility in client applications. For example pre-scan units where customers can scan their products themselves instead of the cashier;
- All POS units have identical product and price information because they share the same information resource;
- Easier and faster replacement of POS units because (product) information does not have to be downloaded to the local machine;
- Additional POS units can easily be added in peak seasons.

The obvious disadvantage of the client/server solution is the dependency of the network infrastructure. To achieve equal availability in the new application more effort and complexity is required in infrastructure by adding redundant components.

The goal of this paper is to examine client/server architectures and possible solutions for redesign of existing applications to this architecture, and to propose a strategy to redesign the POS application. For evaluation of the proposed strategy, the feasibility of a client/server POS and the technical aspects of the implementation, a proto-type application is developed. This proto-type application can be used in future as a model for implementation of the current application.

Redesign Approaches

Reengineering and redesign of applications has become a very important discipline in software engineering, especially in the area of legacy (mainframe) applications. Although the software application in this project can not be considered a real legacy application, it only exists for three years, the years of experience in reengineering and redesigning legacy applications is an important source of information for this project.

Reasons for reengineering and redesign techniques to become this important is because software tends to never reach a point of being finished, and the operational lifetime is quite unpredictable at time of design, implementation and deployment. Some applications will not exceed a few years of operation while other applications will still be business-critical for an organization after ten or even twenty years. During their long lifetime these applications usually are extended, modified and tuned to implement business changes and to optimize its usage in an organization. Due to years of investment these applications are specifically tailored for the business needs and therefore are of great value (Sommerville, 2001).

However, applications that exist for over ten years do have some problems that make modifications and extensions more difficult with the increased lifetime and that prevent them from keeping up with the advances that are made in computer industry. For example opportunities the World Wide Web presents, like direct access to the systems for internal and external partners or customers, are not available because of the proprietary nature of these systems (Bi et al., 2001). Other problems legacy applications faces are:

- Difficult user interfaces that prevent easy access and high productivity;
- Legacy applications usually run on dedicated and obsolete hardware which might be slow and expensive to maintain;
- Software maintenance and extension can be expensive for several reasons like high complexity, obsolete programming language (compared to other technologies the organization uses to build applications), lack of up to date documentation and unavailability of development personnel who designed the system;
- Integration with other systems can be difficult because of missing interfaces.

In order to keep using legacy applications because of their proofed functionality and to deal with the above problems several redesign approaches of legacy information systems are proposed in literature, which generally can be divided in three categories: redevelopment, wrapping and migration (Bisbal, Lawless, Bing Wu, & Grimson, 1999). Although this is a clear categorization of redesign approaches, a redesign or reengineering project usually does not use either one of the approaches. Rather, such a project uses a combination at component level; for example a migration project, in which components are reused, wrapped behind new interfaces. In this section the redesign approaches will be explained in more detail including some examples.

Redevelopment of Legacy Applications

Redevelopment of legacy applications involves the complete application to be redeveloped from scratch using a new hardware platform and modern architecture, new software development tools and new databases. Although this approach might be feasible in small scale applications, in large scale software environments there are some major disadvantages. First of all this approach obviously leads to the most changes in the system as none of the implemented functionality and data of the application is reused. Second, the legacy application needs to be shutdown at once during replacement, which makes it difficult to verify the workings of the new system. Because of these disadvantages this approach usually results in a project of high risk and high costs. Several strategies and tools that support redevelopment are available although most of these are considered too high level or have not been applied in practice (Bi, Hull, & Nicholl, 2001; Bisbal et al., 1999).

Wrapping and Migration of Legacy Applications

Wrapping and migration are two approaches that involve the redesign of legacy applications using the system's original data and functionality. Compared to redevelopment these approaches have less impact on the original system and a lower risk because proofed functionality can be reused in the new system.

Wrapping is an approach that hides existing applications or components behind a new, more accessible layer. In case of wrapping complete applications, a new layer is added on top of the legacy application exposing the interfaces of the application to (new) remote clients. The legacy application remains as it is and the new layer makes the application look new and improved. Because the legacy application itself is not modified, proofed functionality remains the same while user interface and integration with other systems can be improved through the implementation of the new layer. On the other hand, because the legacy application itself is not changed there is little improvement in terms of performance, it still runs on the same hardware, and maintenance will be as difficult as before. Although wrapping in this form is relatively fast to implement, it is considered a short term solution (Bisbal et al., 1999).

Migration is a more complex approach in which the legacy application is moved to a more flexible environment, reusing components of the legacy system in the new environment. Migration requires a much deeper understanding of legacy application and will have a greater impact as compared to wrapping an entire application. The advantage is that it offers long term benefits like more flexibility, better system understanding, easier maintenance and reduced costs. The problem is however that because of the huge amount of different legacy applications in operation, all having their specific problems, it is unlikely that a single generic migration method will be suitable for all systems (Bisbal et al., 1999). The same authors point out that despite the existence of a generic migration method a set of comprehensive guidelines is essential to drive the migration project, although all known successful migration-like projects in literature describe ad hoc solutions to the specific problems.

The key to wrapping and migration of legacy applications is the implementation of middleware, either as a communication layer on top of the complete application in case of wrapping or between the decomposed components in a migration project. Depending on the nature of the legacy application and the environment in which the application is target to operate, different middleware products and implementation strategies are proposed in literature.

Implementations based on the Common Object Request Broker Architecture (CORBA) middleware are frequently used because different language support, available implementations for diverse computing platforms, and the complete functionality CORBA provides. Solutions using CORBA for middleware are frequently based on an approach in which only the presentation layer is converted and the legacy application is considered a distributed object, accessible in a heterogeneous computing environment (Chiang, 2001; Contronеo, Mazzocca, Romano, & Russo, 2002). Besides CORBA implementations other known implementations are based on Java Remote Method Invocation (RMI) (Bi et al., 2001) and Web Services.

Web Services is a relatively new middleware that is used for redesigning legacy applications. Because of its language independent nature Web Services is starting to become a very popular middleware technology for new solutions and for wrapping legacy applications to make them available to the World Wide Web (Litoiu, 2004; Microsoft, 2003; Zhu, 2003). Due to the advantages of platform independency by using the HyperText Transfer Protocol (HTTP) and the Simple Object Access Protocol (SOAP), the implementation of Web Services does affect the response time of the application negatively (Litoiu, 2004). Although this does not necessarily have to be a problem for every application, it needs to be considered when using Web Services.

Terminal Based Solutions

A technique that can not be classified as a redesign approach but is widely used for centralizing stand-alone applications are terminal based solutions, of which the most popular examples are Windows Terminal Server and Citrix® MetaFrame®. With these solutions the server runs specific terminal-server software and the stand-alone application is centralized. Users access the software through specific client software in which only the user interface of the application is transported over the network using respectively Microsoft’s Remote Desktop Protocol (RDP) (Microsoft, 2002a) and Citrix® Independent Computer Architecture (ICA) (Citrix, 2004).

This approach of terminal based wrapping provides several advantages like easy application deployment, performance improvement in certain situations (for example web browsing on PDA's (Lai et al., 2004)) and access possibilities from various client platforms in case of the ICA protocol. However, as with the wrapping strategy at application level, explained in the previous section the application remains the same and there are no improvements in terms of scalability (each client starts its own complete instance of the application), integration with other systems and extension to different user interfaces.

Another problem, specifically related to this project, is the control of hardware peripherals connected to the client machine. Although most terminal based solutions provide redirection features of serial ports the control of the Electronic Funds Transfer (EFT) terminal, which provides electronic payments, is currently expected by certification institutes to run on the client's computer.

Summary

Although the POS application that is subject to redesign can not really be considered a legacy application and does not face the same problems these applications do, the advances made in redesigning legacy applications do apply to this project.

The majority of solutions found in literature propose migration solutions in which the presentation layer of the legacy application is replaced by a middleware implementation, exposing the interface for use by new remote clients and enabling integration with other systems. Migration projects in which the legacy application is decomposed and moved to a new environment are less known because the legacy application is generally considered too large and complex to predict the risk and feasibility of the project.

The terminal based solutions provide an easy transition from a stand-alone application to a centralized solution. However, wrapping the application in a terminal based solution has its limitations in terms of flexibility, scalability, extension to implement different user interfaces, and control of hardware peripherals.

Given the advantages and disadvantages of the examined redesign approaches and the nature of the POS application it is expected that a combination of migration and wrapping at component level, as proposed by (Bisbal et al., 1999), will provide the most flexible solution for redesign of the stand-alone POS application to client/server architecture.

Redesign to Client/Server Architecture

The application that is subject to redesign to client/server architecture is a POS application, which provides complete functionality for a retail business to handle transactions. In this section a solution will be proposed to redesign this specific application to client/server architecture based on a migration scenario in which the application is decomposed in client and server components.

The POS is one component in a total retail automation system at which customers buy products that are offered by a company. At the POS products and payments are registered in a transaction which is send across the network to the 'Back Office' application when completed. For support of the process of selling products different hardware peripherals are attached to the POS computer like a customer display, a scanner, a receipt printer and a cash drawer.

One of the major requirements of the POS application is high-availability, being able to serve customers in almost any condition of the IT infrastructure. With this requirement the POS application is designed as a stand-alone application in a way that operation will not fail when infrastructure resources are not available.

The complete POS configuration is a combination of four components, described in the left part of Figure 1. The POS application uses a local SQL Server database for data like product and price information. This data is updated through messages from Back Office applications, which first arrive in the Message Queue of the POS and are then processed by the Message Processor, a service application. Information that has to be sent to the BackOffice (like transaction data) is composed by the POS application and send, via a queue, to a remote server.

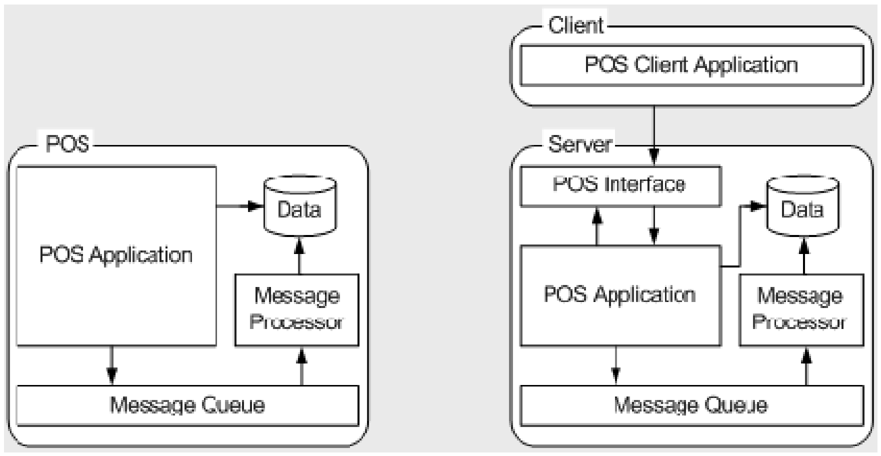


Figure 1: Overview of Redesign POS Application

The Message Queue implementation is an important component in the POS system as it provides a complete loosely coupled (asynchronous) integration between POS and BackOffice. Using the Message Queue the POS application is independent and unaware of the infrastructure's status and messages are guaranteed to arrive at their destination.

When the POS configuration is redesigned to client/server architecture the initial requirement of high-availability will be compromised because the application becomes fully dependent of the IT infrastructure. Moreover, when the infrastructure is unavailable all POS applications will fail to operate. The risk of infrastructure failure is inevitable in client/server architecture and has to be mitigated by use of redundant components in the infrastructure.

For redesigning the stand-alone POS configuration into client/server architecture the following goal is defined:

Redesign of the stand-alone POS application to client/server architecture in which the client side application is designed as thin as possible to lower the hardware requirements at the POS computer. After redesign the application is able to operate in different deployment scenarios in a way that different client/server configurations are possible, as well as a stand-alone configuration when necessary.

Considering this project's goal the current stand-alone POS configuration can be redesigned to client/server architecture in two scenarios:

- Two-tier architecture: Redesign and centralize the SQL Server database and Message Processor for use by multiple POS applications and leave the POS application unchanged; and
- Three-tier / N-tier architecture: In addition to the first step, redesign the POS application to client/server architecture (configuration at the right side of Figure 1).

Each of these scenarios, including the stand-alone POS solution, has their own advantages and disadvantages, which are summarized in Table 1.

Table 1: Differences in Point-Of-Sale architectures

	Stand-alone POS	Centralized Database	Redesigned POS
Impact of redesign	□	-	++
Complexity of application	-	-	+
Hardware requirements front-end	++	+	-/--
Hardware requirements server(s)	□	+/○	+/○
Availability	++	+	+
Front-end replacement impact	++	--	--
Server replacement impact	□	+	+
Scalability	□	-	++
Data consistency	-	++	++
Browser based client application	□	□	yes

++ very high, + high, - low, -- very low, □ none/not available, ○ depended of number of clients

From the results of Table 1 can be concluded that the centralized database solution provides certain advantages over the stand-alone POS, mostly related to easy front-end replacement in an operational environment and database consistency. However, more flexibility and lower front-end hardware requirements are expected from the redesigned POS application. Despite the higher impact of the redesign and the higher complexity of the final application the redesigned POS application will be chosen for further investigation because this design will lead to the most flexible solution.

Redesign of Application

The stand-alone POS application is designed using an Object-Oriented development method and classes are grouped depending on their functionality in separate packages. Logically these packages can be grouped into three layers: a business tier, a middle tier that provides access to the business tier from the user interface, the third tier. The packages used in the stand-alone POS application are presented in Figure 2, including the relationships between the packages. For simplicity reasons the model shows only the main packages, for example the package 'Interface Back-Office' in reality consists of three sub packages.

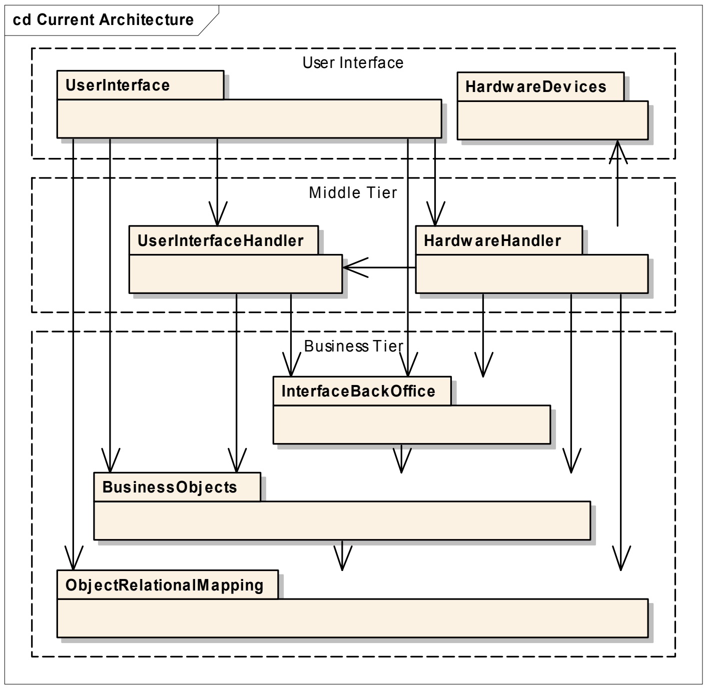


Figure 2: Stand-alone POS Architecture

In order to decompose this application into client and server components and to make the decision whether specific functionality should operate at the client-side or the server-side a clear understanding is required of the entire functionality of the application. The functionality of each package in the stand-alone POS application is analyzed and a distinction is made between client-side, server-side and interface functionality in the new architecture.

Based on a functional decomposition it is clear what functionality will operate at the client-side and the server-side of the redesigned application. The client-side application will contain the user interface and the handling of hardware peripherals and all other functionality will be moved to the server-side. Moving the functionality to the server-side can be accomplished using two different scenarios:

- Three-tier architecture: one server application that contains all functionality; and
- N-tier architecture: decomposition of the server functionality into more (smaller) server applications.

The first scenario has the advantage that it will be less complex to develop and maintain and the client only needs to access one server for all remote functionality. On the other hand, the second scenario will be more flexible in terms of scalability and load balancing because of multiple smaller server applications. For example, with a small number of clients all server applications can run on one server in order to achieve sufficient performance, while in cases of many clients each server application might run on separate servers or even multiple servers running the same server application.

Because of the increased flexibility of the second scenario this scenario will be used for the redesigned POS application. The key components that are identified for design as separate server applications are presented in Figure 3 and described in Table 2.

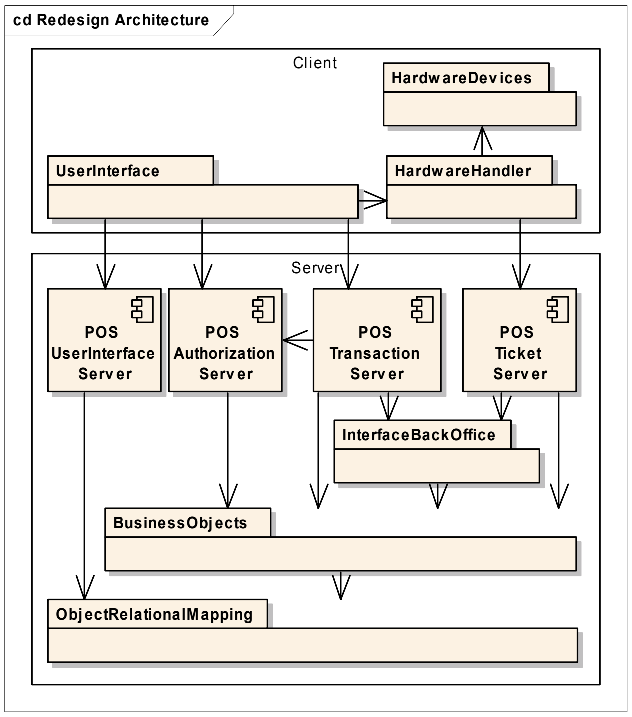


Figure 3: Redesigned C/S POS Architecture

Table 2: Functional description of server applications

Server component	Description
POS Authorization Server	Authentication of clients and cashiers to grant access for use of the Pos Transaction Server.
POS Transaction Server	Registration of POS transactions. For verification of clients requesting a transaction registration the POS Authorization Server is used
POS Ticket Server	Composer of tickets to be printed directly on the receipt printer
POS User Interface Server	Information provider for dynamic user interface components like lay-out of the customer display and functions of on-screen buttons and keyboard

The interaction between the client application and the server applications is presented in the sequence diagram of Figure 4. The sequence diagram describes the application’s start up, a cashier who logs in at the client and performs a registration of one transaction after which the session is finished by logging off.

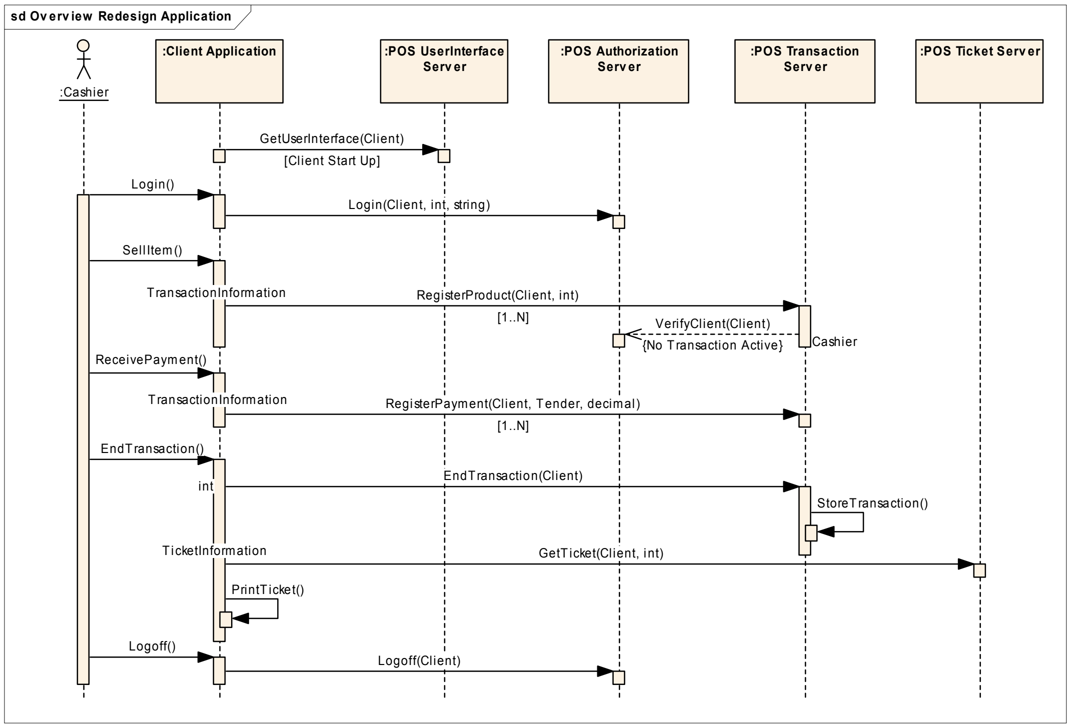


Figure 4: Sequence Diagram 'Operation of C/S POS Application'

At start up of the client application a `Client` object is created for identification of the client application which is passed in all calls to the server applications. Also the ‘Pos User Interface Server’ is accessed to retrieve information about the dynamic parts of the user interface. Next step is for the cashier to login to the application. The cashier will be verified and stored by the ‘POS Authorization Server’ and after successful authorization the cashier can start to register products and payments using the ‘POS Transaction Server’. At the first registration of a product the request will be verified at the ‘POS Authorization Server’ to prevent unauthorized use of the server and when successful an active transaction will be created and registered at the server. All products and payments that are registered afterwards will be added to the client’s active transaction. When all products and payments are registered the cashier can end the transaction. This will cause the transaction to be stored and the ‘POS Ticket Server’ can use this transaction to compose a receipt to be printed by the client’s hardware.

Transition Strategy

An important final step in this redesign approach to a client/server POS is replacement of the current POS application in a production environment. Three different transition strategies are possible (Bisbal et al., 1999):

- *Cut-and-Run:* Replace the stand-alone POS application for the complete redesigned application;
- *Parallel Operations:* Running the complete redesigned POS application in parallel of existing POS application;
- *Phased Interoperability:* Implement the redesigned POS application in ‘small’ iterations and deploy each step in a production scenario.

The *Cut-and-Run* and *Parallel Operations* strategies are both situations in which the complete application is implemented at once. Of these strategies the *Parallel Operations* approach is the more realistic situation because the new application can be tested in parallel with the original application, verifying the results of both applications.

The main disadvantage of both these approaches is the time it takes to implement the new application. During the implementation it is not unlikely that the original application is subject to modification and needs to be deployed quickly. This might be impossible because of the redesign implementation in progress. An alternative is to maintain the two applications in parallel. In this case changes have to be implemented in both applications, which is less desirable because of the risks it involves.

In case of the redesigned application based on the proposed design of using multiple server applications the best transition strategy to use is *Phased Interoperability*. In this strategy the application is implemented in small manageable iterations. The application can be tested and deployed with each iteration and required application modifications can be included if necessary. A possible sequence of implementation iterations can be:

1. Centralize database and Message Processor;
2. Implementation of the ‘POS User Interface Server’;
3. Implementation of the ‘POS Authorization Server’;
4. Implementation of the ‘POS Ticket Server’;
5. Implementation of the ‘POS Transaction Server’.

Design of Application

Based on the proposed design of the client/server POS a proto-type client/server POS will be designed and implemented. The intention of this proto-type application is to evaluate the feasibility of a client/server POS application, the proposed design and the technical aspects of the implementation.

For this proto-type application only the core functionality of the original POS application will be implemented and to avoid interference with the continuous development of the stand-alone application no original source code will be used. However, the design of the proto-type will match with the original design so the proto-type application can be used as an implementation model for the migration of the original application.

From the proposed design the parts that will be designed and implemented are the ‘Pos Authorization Server’, the ‘Pos Transaction Server’ and two variants of client applications, a Windows Desktop client application and a Web browser client application.

The defined functionality of the proto-type application involves the design of two server applications, the ‘POS Authorization Server’ and the ‘POS Transaction Server’, which are identified in the architectural design of Figure 3.

For evaluation of different type of client-side applications the proto-type application will be designed with two different client applications: the ‘POS Client Application’ and the ‘POS Web Client’. The ‘POS Client Application’ is a Windows desktop application of which the user-interface will be equal to the current POS application. The ‘POS Web Client’ is a web application that runs in a web browser and therefore the user interface will have a slightly different look and feel.

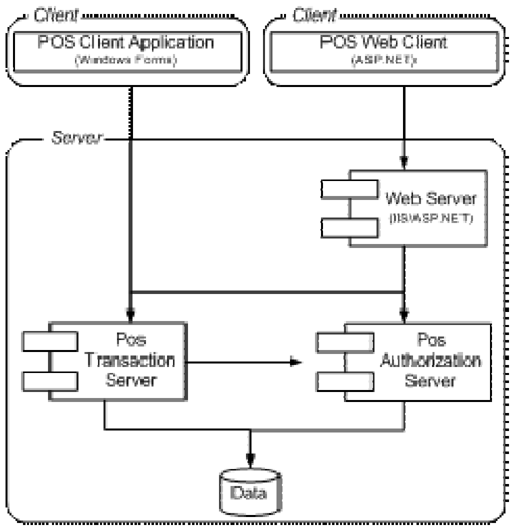


Figure 5: Overview Client/Server Application Design

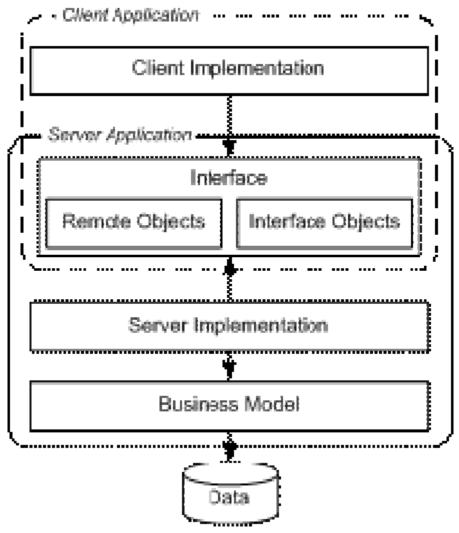


Figure 6: Server Application Design

Both client applications will be equal in functionality and operate using the same server applications. The web application however does require a Web Server at the server-side in front of the server applications to create the web pages for the browser. In Figure 5 the design of the complete proto-type application is presented.

General Design Server Applications

The server applications identified for the prototype application are both designed using the same design model, which is shown in Figure 6.

In this model the server application is divided in three layers (packages) to provide a clear distinction in the functionality can be accessed by clients, the functionality that is specifically for operation of the server and the business model. Furthermore, this design hides the server's implementation from the client, making it possible to change and deploy the server's implementation without modification of the client applications.

Interface

The interface layer is a collection of class definitions that describe the services the server application offers. The interface layer is a separate package that can be included in the client application for access to these services. In this interface layer two types of objects are defined:

Remote Objects
Objects that run on the server and clients can obtain a reference to these objects for method invocation. In the interface layer only the interface classes (or the abstract classes) of the remote objects are defined to hide the object's implementation.

Interface Objects
Objects that can be created by the client or the server which are used to interchange information between the client and the server (passed as method parameters or return values).

Server Implementation

The Server Implementation contains the classes that realize the server application. In this layer the Remote Objects are implemented which are defined in the Interface. Classes can be added in this layer to support the server's functionality.

Business Model

The Business Model contains the classes that define the business logic of the application.

Considering the original POS application of Figure 2 the Server Implementation will typically contain the functionality that is extracted from the User Interface and the Middle Tier. The Business Model contains the packages from the Business Tier. In case of this redesign approach the packages of the original application can be used without modification, shared across the different server applications.

Implementation of Application

The original POS application was developed from scratch nearly three years ago with Microsoft Visual Studio .NET (now version 2003), using C# as the programming language. This means that the application is built with the .NET Framework and is managed and executed by the Common Language Runtime (CLR), which is similar to the Java Virtual Machine.

The fact that the application is already developed in an up-to-date development environment brings a huge advantage in the redesign approach as the source code does not have to be ported to a new development environment. Moreover, in comparison to COM/DCOM Microsoft now supports greatly improved possibilities for distributed computing which makes middleware from other manufactures less desirable to use in this redesign implementation.

Considering all features of the available frameworks the choice for this project's implementation is the .NET Remoting framework. The reason for this choice is that the interoperability features of Web Services and the services of Enterprise Services are not required and .NET Remoting provides the highest performance in communication between client and server.

For this project the server applications will be hosted in a Windows Service and the communication parameters will be TCP/Binary. However, because of the flexibility of .NET Remoting the hosting and communication parameters can easily be changed when for example security services are required in specific deployment scenarios.

Deployment of Application

The implemented redesign approach of decomposing the POS application into multiple distinct server applications introduces possibilities for flexible deployment of the application. The deployment scenarios can be based on the customer's specific needs, without modification of the software. This section discusses different scenarios of deployment for the POS application.

Single Computer Scenario

The single computer scenario is basically no different than the current POS application. All server applications and the client application run on each POS's computer. In this scenario there is no additional value for using the Web Browser Client application and it is therefore recommended to use the Windows Desktop Client application. This deployment scenario can be used in companies with a single POS or situations of multiple POS's that require utmost elimination of any possibility in infrastructure failure. The disadvantage of this scenario is the requirement of the hardware resources of the POS computer, which will as high as the current POS application.

Single Server Scenario

In a single server scenario, in which the server applications as well as the database run on a single server, the advantages of the client/server application start to emerge. The hardware requirements of the POS computer can be significantly reduced and adding new clients (or replacing clients) will be quick and easy. The obvious disadvantage in this scenario is the single point of failure that is introduced by the single server. If this server fails to operate all client applications in the configuration will fail. Therefore this scenario is less recommended to be used in a production environment; at least one server should be added in the configuration to reduce the risk of unavailability.

Multiple Server Scenario

To mitigate the risk of single point of failure of the previous single server scenario multiple servers can be introduced in the deployment configuration. In Figure 7 one of many possible scenarios of multi server deployment is presented.

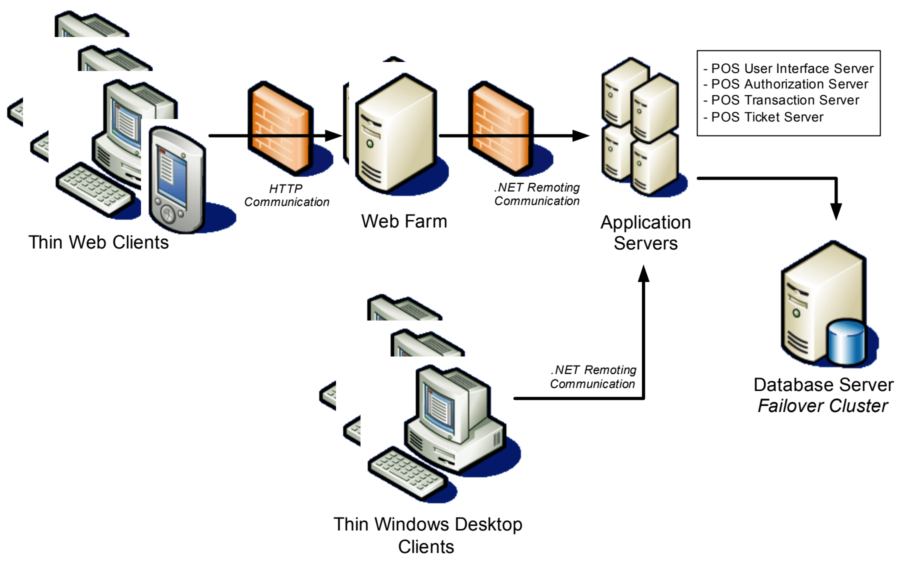

Thin Web Clients
HTTP Communication
Web Farm
.NET Remoting Communication
Application Servers
- POS User Interface Server
- POS Authorization Server
- POS Transaction Server
- POS Ticket Server
Database Server
Failover Cluster
.NET Remoting Communication
Thin Windows Desktop Clients

Figure 7: Deployment of Point-Of-Sale

In this multi server scenario the server applications are deployed across multiple servers. The Web Server, optionally placed in a demilitarized zone (DMZ) using firewalls, serves the web clients by providing the web pages. The Database Server runs the SQL Server database engine. Again these servers introduce single points of failure in the configuration, which can be eliminated if necessary using respectively a web farm and a database *Failover Cluster* (Microsoft, 2004a). These solutions basically add redundant servers that take over operation seamlessly if the primary server fails.

The Application Servers in the configuration are a series of servers that run the server applications of the POS application. The number of servers used for the Application Servers depend on the number of clients that are defined in the configuration although at least two servers are necessary to mitigate the risk of unavailability. When more clients are added to the configuration and the response time of the server applications increases, the performance of the Application Servers can be upgraded by adding more servers to the configuration.

For implementation of the Application Servers that provide the described functionality of redundancy and scaling out (adding more servers to improve performance) two different solutions are possible: Manual Load Balancing and Automatic Load Balancing.

Manual load balancing

With manual load balancing the POS application is configured manually at time of deployment. The server applications are all installed at multiple server computers and the client applications are assigned to use the server application from a specific server. Additionally the clients are assigned to a second server that can be used in case the primary server is unavailable.

Most important advantage in this solution is its simplicity of implementation, although the client application has to be programmed to connect to the redundancy server in case of failure of the primary server. The downside is that the workload is not evenly balanced among the servers, especially when some of the client applications are temporarily not used. And, when servers are added the clients have to be redistributed manually across the servers.

Automatic load balancing

The problems that are identified with Manual Load Balancing can be solved by the technique of Automatic Load Balancing. With Automatic Load Balancing the clients are not assigned to specific servers at deployment but automatically at run-time using a central located Load Balancer. The Load Balancer receives all requests from the clients and based on a specific algorithm each request is redirected to one of the application servers.

The advantage of Automatic Load Balancing is that the server configuration is transparent from a client's point of view, only one (virtual) server exists, and the workload is optimal balanced over the existing servers. However, Automatic Load Balancing is a complex technology and many diverse implementations exist. These solutions can be distinguished in hardware and software solutions, and static and dynamic solutions.

It is expected that Automatic Load Balancing can play and important role in the future deployment of the client/server POS application, especially when the configuration involves a large number of client applications. When Automatic Load Balancing is considered for implementation further detailed research is required in this area, which is beyond the scope of the project at this stage.

Reflection on Design Decisions

In this section the proposed design and implementation aspects of redesigning the POS application to client/server architecture will be evaluated using the experience of building the proto-type client/server application.

Redesign Approach

The redesign approach conducted for this project's POS application is based on the techniques of wrapping and migration. The application is first decomposed in client-side and server-side functionality with the goal to create a client application that is as thin as possible to limit the front end hardware requirements. The result of the decomposition is a client application that only contains the user interface and the control of hardware peripherals, while the remaining functionality is moved to the server-side. In addition, the server-side functionality is further decomposed in different components to create smaller units of functionality.

This approach has resulted in a client/server application existing of four server applications that cooperate to provide the complete POS functionality to the thin clients. In comparison to a three-tier client/server solution, having only one server application, this N-tier design of multiple applications has the disadvantage of being more complex to monitor and maintain in an operational environment. However, the N-tier solution provides important benefits in flexibility and overall performance that outweighs this disadvantage. Because each server application is dedicated to provide only a small part of the total functionality, multiple client applications that are in different stages of the transaction-process can be served in parallel. Moreover, duplicates of each server application deployed across multiple servers allow parallel access to the same functionality. This flexibility in scalability and load-balancing leads to quicker response times and overall performance which cannot be achieved in a three-tier design.

Implementation

The implementation of moving to a client/server POS mainly involves the development of the server applications. These server applications wrap the packages of the business tier and are implemented using extracted source code from the user-interface-tier and the middle tier. Considering the total impact of the implementation the highest impact will be extraction of the source code, moving it from the middle-tier packages and the user-interface to the specific server applications. The original packages of the business tier are used without modification.

Using the transition strategy of *Phased Interoperability* the redesign project can be implemented in small iterations, implementing one specific server application per iteration. This transition strategy makes the project considerably more manageable and less of risk because each iteration can be fully tested against the original application and deployed if necessary.

Another aspect in the implementation phase is the middleware that is used for communication between client and server application. Different middleware technologies are explored and given the development environment of the application, a technology supported by the .NET Framework is preferred because of easy integration and future development of the application. The middleware technology of choice is .NET Remoting as this provides synchronous communication, flexible configuration in communication and security, and the highest performance in communication. The fact that this technology requires the client application to be developed with the .NET Framework is for this project not a problem because interoperability with other (third-party) systems is not a requirement. With the implementation and testing of the proto-type application .NET Remoting showed to be a flexible, high-performance, easy to implement and reliable middleware technology for this type of application.

Client Applications

The POS application is redesigned with the requirement to create a thin-client solution. Only two functional components of the application are implemented in the client application, the user interface and the control of the hardware peripherals. In addition to the Windows Desktop Client, which is the result of the implemented redesign, a new Web Browser Client application is developed for support of a browser based application.

Both client applications provide the same functionality although the user interfaces are slightly different because of the different technologies. The Windows Desktop Client is a ‘normal’ .NET application in which all functionality of the .NET Framework is available, as well as direct access to all resources of the client’s computer.

The Web Browser Client is different in this context. The application is developed with ASP.NET and runs in a web server, Internet Information Server (IIS). From the web server the POS server applications are accessed using .NET Remoting and the client is only presented an HTML page in its browser. Using of the browser based client application does have its advantages, especially in deployment and updating the application because there is only one location to update and all clients automatically use the new version. However, this approach introduces some complexities because of the browser’s limitations, in this case specifically related to the control of the hardware peripherals.

Accessing the hardware peripherals from a browser is possible using the same standardized technique as the Windows Desktop Client, OPOS (OLE for Retail POS (Monroe, 2004)). Because this technique is based on ActiveX controls running on the client’s computer these controls are accessible from an ActiveX enabled browser using client-side scripting (IBM, 2004). For all hardware peripherals there are OPOS controls available, except for the EFT (Electronic Funds Transfer) terminal. Currently no standard OPOS implementation exists for EFT terminals and this makes usage difficult as controls have to be specifically developed. However, some vendors of EFT terminals are currently working on TCP/IP terminals (instead of serial communication) that can be accessed from anywhere in the network. This makes other (centralized) solutions possible but this will not be available until next year.

Considering both client applications the Windows Desktop Client is less complex to develop and provides the fastest communication because of direct access to the server applications using .NET Remoting. The browser based solution provides a ‘thinner’ client application, only a browser and the OPOS controls are required, and is easier to deploy in an operational environment. However, this concept introduces additional complexities with hardware peripherals and performance will be reduced because of the additional web server.

Performance and Reliability

Performance and reliability are two of the most important aspects of the POS application. In the stand-alone POS application reliability is not dependent of the network infrastructure and performance is directly related to the available resources of the POS’s computer. With the client/server POS application (single/multiple server deployment) both reliability and performance become dependent of the network infrastructure. Therefore the reliability of the network infrastructure is very important and single points of failure in the infrastructure should be eliminated using redundant components. Of these redundant components the servers can play an additional role to improve the overall performance, distributing the workload across the available servers (load-balancing). Adding redundant components obviously increases the costs of the network infrastructure and maintenance, although these costs are compensated by low-cost POS computers that are possible because of this client/server concept.

During the test phase the proto-type application is analyzed in different deployment scenarios and in none of the scenarios problems are encountered in the application’s performance. However, the proto-type application only consists of a sub-set of functionality from the original application. This makes it difficult to predict the performance issues related to a total client/server POS application, for example the number of clients per server. These issues have to be evaluated once the complete application is redesigned. More servers can be added in the configuration when performance turns out to be insufficient, which is possible because of scalability of the application.

Future Work

In this project most important aspects of redesign and implementation of the client/server POS application are explored and evaluated. However, some issues require further investigation in the future of this project:

- Research of changes required for the new implementation of .NET Remoting in .NET Framework 2005, codename Indigo (Microsoft, 2004b);
- Development of a management tool to maintain/monitor the server applications;
- Research of possibilities in automatic load balancing for transparent scalability in large scale implementations;
- EFT implementation for the Web Browser Client.

Conclusions

Redesigning applications into client/server architectures is a frequently addressed topic in literature. Many papers exist but unfortunately there appears to be no detailed generic method or solution that is suitable for all systems. The reason for this is that applications all have different problems and specific solutions are used to solve them. The redesign approaches that are explored describe different solutions, based on wrapping and migration techniques, to provide new user-interfaces and interoperability with other systems.

The redesign approach proposed for the Point-Of-Sale application is similar to the solutions found in literature. The application is decomposed in a client and server functionality and middleware is implemented to provide the communication. Additionally the server's functionality is further decomposed into four distinct server components, which results in a flexible and highly scalable Point-Of-Sale application.

The implementation of the redesigned application involves the creation of the server applications, which wraps business tier of the original application and contains the source code extracted from the user interface and the middle-tier. The functionality of the server application is exposed by the interface of the server. The interface also includes communication objects that represent a simplified model of the business logic, only containing information that is required in the user interface of the client application. For communication between client and server applications the middleware product .NET Remoting, provided by the .NET Framework, has proven to be a flexible, easy to implement and reliable middleware technology in the implementation and tests of the proto-type application.

Because the business tier of the original application is not modified, the highest impact in the implementation will be rewriting the middle-tier and the user interface packages, moving functionality to the server applications. To make the implementation a manageable project of minimal risk, a transition strategy named *phased interoperability* is the most appropriate strategy for this project. Using this strategy the implementation is separated in small iterations of one server application that is fully tested and deployed before moving to the next iteration.

In addition to the Windows Desktop Client, which is the result of the redesign, a second Web Browser Client application is developed. The browser based client provides advantages specifically in deployment of the client application but introduces complexities related to the control of the hardware peripherals. Accessing the hardware peripherals is possible using client-side scripting and the OPOS ActiveX controls, although an OPOS control is currently not available for the EFT terminals.

The redesign approach and implementation of the proto-type application has shown the possibilities of redesigning the Point-Of-Sale to client/server architecture. The flexibility and usability of the application are greatly improved as different operational configurations are possible by changing the configuration files only. In the future of this project a few aspects have to be explored in more depth. These aspects involve performance issues, which could not be evaluated completely because of the reduced proto-type functionality, and automatic load-balancing in case of large scale implementations. However, due to the flexibility and the ability to scale out the application, possible future problems in performance can be avoided at any time.

Based on the research, the proposed design, the evaluation of the proto-type application and the sponsor's evaluation, this project has now reached a stage at which it is recommended to start the implementation in the current Point-Of-Sale application, creating new opportunities for use of the future Point-Of-Sale.

References

Bi, Y., Hull, M. E. C. & Nicholl P. N. (2001). An XML approach for legacy code reuse. *Journal of Systems and Software, 61* (2), 77-89.

Bisbal, J., Lawless, D., Bing Wu., & Grimson, J., (1999). Legacy information systems: Issues and directions. *IEEE Software, 16* (5), 103-111.

Chiang, C., (2001). Wrapping legacy systems for use in heterogeneous computing environments. *Information and Software Technology, 43* (8), 497-507

Citrix. (2004). Understanding Citrix ICA. Retrieved August 17, 2004 from http://www.citrix.com/site/PS/products/QA.asp?familyID=19&productID=1449&faqID=5638&featureID=QAP

Cotroneo, D., Mazzocca, N., Romano, L., & Russo, S. (2002). Building a dependable system from a legacy application with CORBA. *Journal of Systems Architecture, 48* (1), p 81-98.

IBM Support. (2004). OPOS Checkup. Retrieved July 30, 2004 from http://www2.clearlake.ibm.com/store/support/html/oposcheckup.html

Lai, A. M., Nieh, J., Bohra, B., Nandikonda, V., Surana, A. P. & Varshneya, S. (2004). Mobility: Improving web browsing performance on wireless PDAs using thin-client computing. *Proceedings of the 13th conference on World Wide Web*, May 2004

Litoiu, M. (2004). Migrating to Web services: A performance engineering approach. *Journal of Software Maintenance and Evolution: Research and Practice, 16* (1-2), 51-70.

Microsoft. (2002a). Technical overview of windows server 2003 terminal services. Retrieved August 17, 2004 from http://www.microsoft.com/windowsserver2003/techinfo/overview/termserv.mspx

Microsoft. (2002b). ASP.NET web services or .NET remoting: How to choose. Retrieved April 14, 2004 from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/bdadotnetarch16.asp

Microsoft. (2002c). Performance comparison: .NET remoting vs. ASP.NET web services. Retrieved April 14, 2004 from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/bdadotnetarch14.asp>

Microsoft. (2003). Web service facade for legacy applications. Retrieved April 6, 2004 from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/wsfacadelegacyapp.asp

Microsoft. (2004a). Performance and reliability patters – Failover clusters. Retrieved August 7, 2004 from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dn-patterns/html/DesLoadBalancedCluster.asp

Microsoft. (2004b). Code name Indigo: A guide to developing and running connected systems with Indigo. Retrieved May 15, 2004 from http://msdn.microsoft.com/msdnmag/issues/04/01/Indigo

Monroe Consulting Service. (2004). OPOS back-ground. Retrieved August 26, 2004 from http://monroecs.com/oposbackground.htm

Orfali, R., Harkey, D. & Edwards, J. (1999). *Client/Server survival guide* (3rd ed.). New York: Wiley Computer Publishing.

Rammer, I. (2002). *Advanced .NET remoting* (C# Edition). Apress.

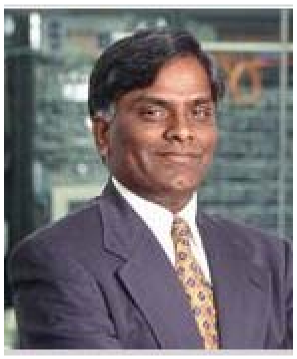
Rammer, I. (2004). .NET remoting use-cases and best practices. Retrieved August 18, 2004 from http://www.thinktecture.com/Resources/RemotingFAQ/RemotingUseCases.html

Sommerville, I. (2001). *Software engineering* (6th ed.). Addison-Wesley.

Zhu, J. (2003). Web services provide the power to integrate. *IEEE Power and Energy Magazine, 1* (6), 40-49

Biography

Michel van der Vlugt is living in The Netherlands. He got his Master of Science degree from the Liverpool University. He has been working as an IT developer, software specialist and consultant for more then seven years and is currently working as a Project Engineer for AXI B.V., Breda, The Netherlands. He has been working mainly for the Windows platforms with different developer tools and developed applications by using different programming languages such as C, C++, Visual Basic, C#, Java and .NET. The fields of his special interests are User Requirements Gathering, Project Organization, Integrated Development Environments (IDE) & Development Tools, Web Applications & Web Services, Testing and Software Development Automation. He is using different .NET technologies in current project developments.



Dr. Samuel Sambasivam is the chairman of the Department of Computer Science of Azusa Pacific University. Professor Sambasivam has done extensive research, publications, and presentations in both computer science and mathematics. His research interests include optimization methods, expert systems, Fuzzy Logic, client/server, Databases, and genetic algorithms. He has taught computer science and mathematics courses for over 20 years. Professor Sambasivam has run the regional Association for Computing Machinery (ACM) Programming Contest for six years. He has developed and introduced several new courses for computer science majors. Professor Sambasivam teaches Database Management Systems, Information Structures and Algorithm Design, Microcomputer Programming with C++, Discrete Structures, Client/Server Applications, Advanced Database Applications, Applied Artificial Intelligence, JAVA and others courses. Professor Sambasivam coordinates the Client/Server Technology emphasis for the Department of Computer Science at Azusa Pacific University.