Distributed Information Systems. From A to Z. - Part I. Introduction.

Abstract: This article is a first from a set where we going to discuss several aspects of building distributed information systems using Delphi 7 and Indy.

After publication of Building a stand-alone Web service with Indy in Delphi 7 - Part I and II we got many e-mails how to use our techniques and how to build n-tier distributed systems in Delphi 7. These questions gave me an idea to extend our articles and add more information.
As a result I want to present this article. For now I planning to have 6 parts where 2 of them have been published already.
I'm going to discuss how to build simple and more complex n-tier application using Delphi 7, Indy 9, DataSnap and SOAP.
There is a list which will lead you to particular article:

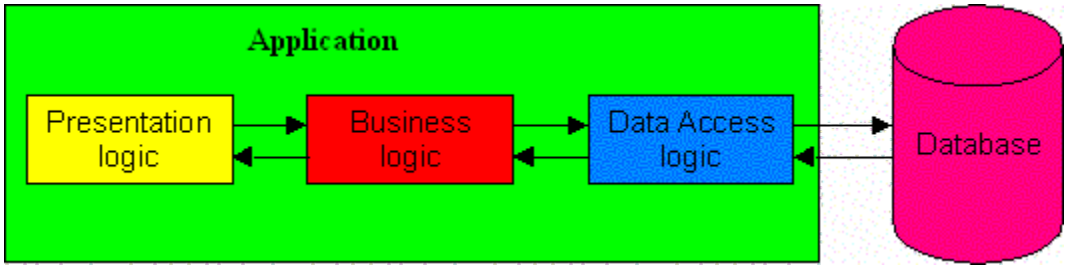
- *Part I. Introduction. Start design of n-tier application.*
- Part II. Building a stand-alone Web service with Indy in Delphi 7.
- Part III. How to use multiple SOAPDataModules in your Web service.
- Part IV. Using multiple SOAPDatamodules or multiple DataModules.
- Part V. Combining SOAP DataModules and SOAP Interfaces within your server logic.
- Part VI. Dynamic model for your business logic. Using Plug-In model with your WebServices.

Tiers. What this means?

We hear this word every day in different context: monolithic application, Client/Server, 2-tier, 3-tier, n-tier.

The 1-tier architecture or monolithic application.

Let's look at next diagram:



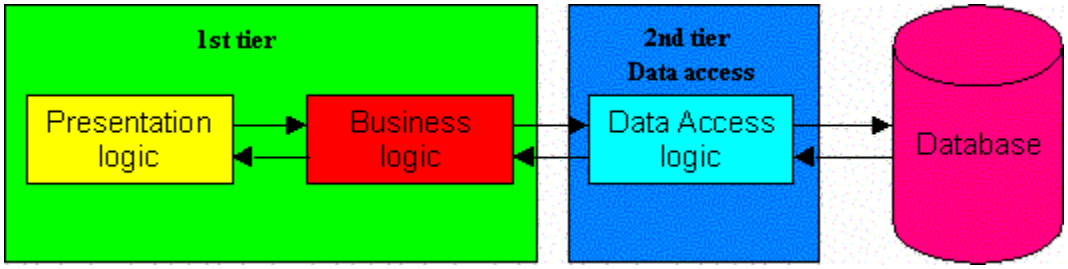
Application contains all the code necessary to deal with user interface, data processing and database communication. Logic here not separated by functionality and can lead us to a situation when we can duplicate a code over and over to access to same data objects.

Because data access and business logic is duplicated, any changes in one of them will require similar changes to be replicated in others.
To do not repeat a definition again and again, let's specify some words:

- Presentation Logic – User Interface, allows to present our data to a user and accepts input for user to change a data
- Business Logic – allows perform validation of new data and ensure what data could be applied to a database
- Data Access Logic – implements database access, sending and retrieving information from physical database. Could vary between different database systems.

The 2-tier architecture.

Here we splitting our application into two parts (tiers). Usually we will see next schema:

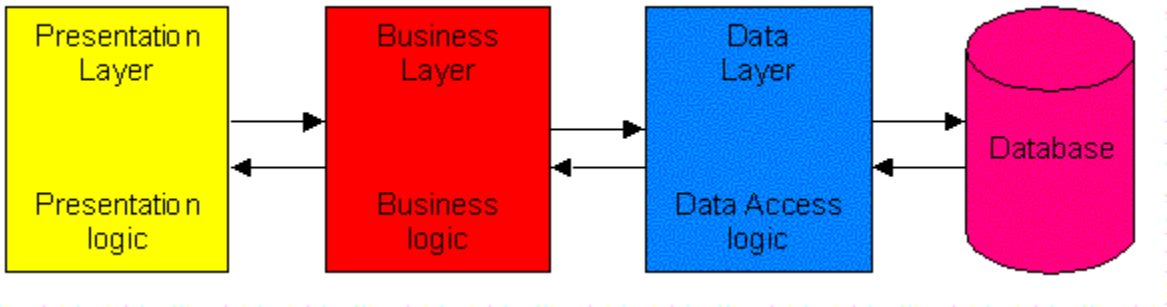


Remember a definition for Data Access Logic? Because you would like to allow your application use different Database Systems it means your access logic could be changed between them to optimize access and/or implement structures not available in some of them.

And this schema allows you to do this. You could have the same presentation and just by changing content of data access tier you could switch different DBS.

The 3-tier architecture.

One more split and we got a system we looking for:



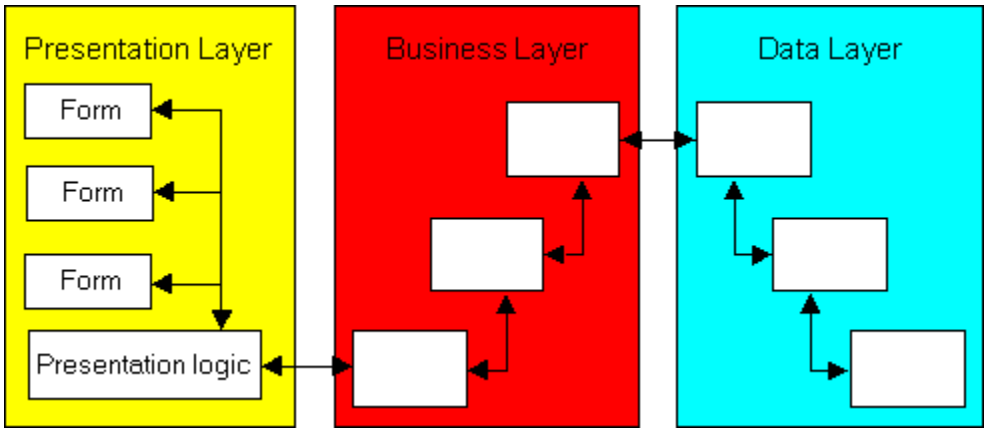
We have an each piece of original schema as a separate layer.

Main advantage of this structure over 2-tier is a business logic encapsulated as a separate

object and could be shared among many different components on presentation level. Any changes on business level can be made in one place and instantly available throughout the whole application.
For this structure we have to remember what stability of whole system defined by stability and how accurate interfaces between different tiers are defined. This will allow making any changes in logic of a tier without affecting a code in others. Example of this could be switching between Windows-based and web-based clients.

N-tier?

At this point we got all we need: we redesign our application to have all pieces of our logic split up in different layers, we can work on different code platform and with different DB systems. We achieve our goals! But application still most likely will work on one computer. It still requires all resources and could perform one task at the time (until you run multiple instances or implement multi-threading, we not going to discuss this method here). What could we do?
Use resources available on your network! And here n-tier comes in play.
Web-resource searchWebServices.com Definitions gives us next definition for N-tier:
"An n-tier application program is one that is distributed among three or more separate computers in a distributed network. The most common form of n-tier (meaning 'some number of tiers') is the 3-tier application, in which user interface programming is in the user's computer, business logic is in a more centralized computer, and needed data is in a computer that manages a database. N-tier application structure implies the client/server program model. Where there are more than three distribution levels or tiers involved, the additional tiers in the application are usually associated with the business logic tier. In addition to the advantages of distributing programming and data throughout a network, n-tier applications have the advantages that any one tier can run on an appropriate processor or operating system platform and can be updated independently of the other tiers. Communication between the program tiers uses special program interfaces such as those provided by the Common Object Request Broker Architecture (CORBA). "
N-tier structure could be presented like this:



You can see what out Business Layer and Data Layer now are not mono blocks. They contain more then one piece and then could be moved to different computers on your network or even around world via Internet.

Disadvantages? Yes, there are some…

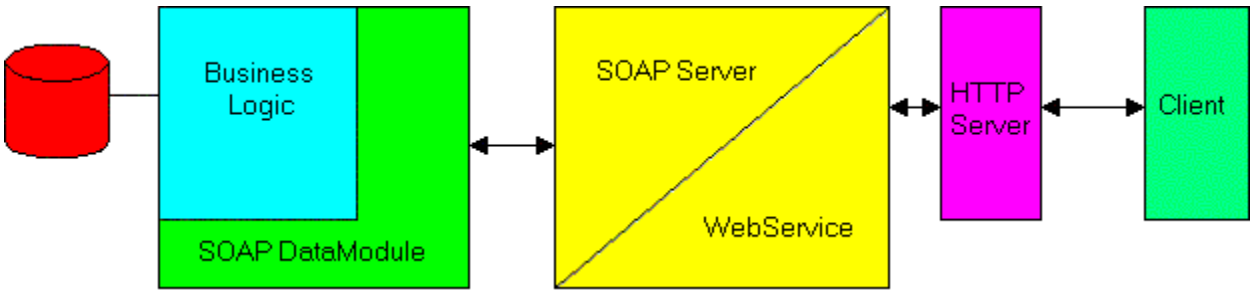
You build your first n-tier application. You could use different tools (Delphi, C++Builder, Visual Basic, etc.) with different DB platforms (MSSQL, Oracle, Interbase) – result could vary but something you will see in common… Using this model you have to remember what design is very important. With n-tier structure interfaces between tiers have to be static. If you start to change it every day, you will shoot your self in a foot and return to a problem we trying to avoid from the beginning – you will sit and patch your code all days.
Another point people missing very often is what n-tier doesn't have to give you increase in speed of your application, especially when you looking just on one process thread. Why? When we redesigned our application we add some "overcode" – intermodule communication and object encapsulation. Two simple examples:

- code written in Assembler will always faster then in any 3GL language – we add translation level (object representation)
- Program distributed between 3 computers will work slower then program executed on 1 computer - we add a communication limitation of our network resources.

So, are you disappointed? Please don't be. It is very unusual these days when we have only one person working with a program in one area. You will most likely see your program act as client run from different computers to access to central data storage. So you will have similar processes run over and over. And here n-tier structure will work perfectly, allowing you balance your process load and resources to get best performance.
Please check our reference section for more information about n-tier.

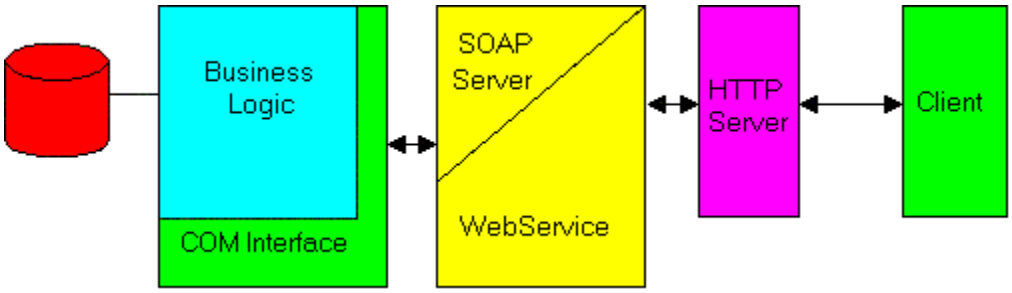
Some implementation model of n-tier application.

In this part I will present some of the ideas we look at before we choose our structure. I'm not going today discuss all variety of possible implementations. You could always find most suitable for you. Remember our 3-tier model? A little transformation and we add couple more layers to be able to publish our application via HTTP server. As HTTP server you could choose one of the standard one like IIS, Apache or build own using Indy components available in Delphi.



In next article we will discuss how to build such server, but for now let just look how else we could build our system. There will be no significant changes, but it will just show you what there is always a freedom of choice. Because of specifics of how WebServices works usually a model build on it means your server will be stateless.
From The Free On-line Dictionary of Computing (09 FEB 02) :
Stateless - A stateless server is one which treats each request as an independent transaction, unrelated to any previous request. This simplifies the server design because it does not need to allocate storage to deal with conversations in progress or worry about freeing it if a client dies in mid-transaction. A disadvantage is that it may be necessary to

include more information in each request and this extra information will need to be interpreted by the server each time.
An example of a stateless server is a World-Wide Web server. These take in requests ({URLs) which completely specify the required document and do not require any context or memory of previous requests.
Contrast this with a traditional FTP server which conducts an interactive session with the user. A request to the server for a file can assume that the user has been authenticated and that the current directory and transfer mode have been set.
As you can see it is not always could be acceptable or just introduces some overhead logic you could probable avoid. What could you do here? One of the solutions could be to use COM object as storage of your business logic. Then we will get next schema:



You could see what a changes isn't so significant. But this could make logic of your application simpler because server became stateless so you don't have to worry to remember a prior state. This could work when information which you publish not filtered based on user request or handled in some different way, such a session unique for each user. So when you have not so many users you could allow system have COM object for each of your clients.
There is not so many changes you will need to make in what we will discuss, so I'm not going to discuss it here. If you have any problem with how to adjust a code, please contact me.

References:

1. Application Architecture: An N-Tier Approach
2. N-tier
3. N-tier related links
4. 3- and n-Tier Architectures

About the Author

Serge Dosyukov is a founder of Dragon Software, a consulting company which provides a consulting and training services for Delphi, MSSQL Server, Wise Installation System and Wise for Windows Installer since 1996.
For information about on-site training and consulting you can contact author via email Serge or visit his Web site at www.dragonsoft.us.
