Vol 8. No. 4, December, 2015
ISSN 2006-1781

African Journal of Computing & ICT

www.ajocict.net

A Survey and Comparative Analysis of Database Software Architectures

J.R. Bunakiye
[1.] Dept. of Mathematics/Computer Science
Faculty of Science, Niger Delta University,
Wilberforce Island, Nigeria.
jbunakiye@gmail.com

E.E. Ogheneovo
[2.] Department of Computer Science, Faculty of Science
University of Port Harcourt
Port Harcourt, Nigeria.
edward_ogheneovo@yahoo.com

ABSTRACT

Database software such as Database Management Systems (DBMSs) is ubiquitous and contains thebusiness rules processing, data access, and presentation or interfaceareas of functionality. This functionality can be better understood in terms of how well structured are the components of the software. Though a lot of design for database architectures are continually considered, literature centered on database systems architecture are not as broadly known as they should be. This paper presents an architectural discussion of database software in line with application architectures to come up with a comparison relating to transactional processing and the stored program concept in a database software. It is intended to provide a common understanding around which the quality of database software can be ensured to a certain degree of performance.

Keywords:Ubiquitous Components, Degree of Performance, Transaction Processing Monitor, Stored Program Concept

African Journal of Computing & ICT Reference Format:
J.R. Bunakiye & E.E. Ogheneovo (2015): A Survey and Comparative Analysis of Database Software Architectures. Vol 8, No. 4. Pp 209-.213.

1. INTRODUCTION

All computer applications, including database software have three general areas of functionality: business rules processing, data access, and presentation or interface. Business rules are the parts of the business process that computer applications automate. Data access concerns the code that automates the storing, searching, and retrieving of data by computer applications [13]. The interface allows applications to communicate with applications and people. The ways in which these application functions are assembled in database software determines the flexibility of the applications, determines how quickly they can be modified to support changes in business and technology, and also determines how easily they interface with people and with each other [4].

Database Management Systems, which are typical database software, are ubiquitous and critical components of modern computing, and their developments have spanned many systems design techniques for scalability and reliability. While many design considerations are increasing, discussions centered on database systems architecture are not as broadly known as they should be, also the coverage in the literature of software architectures in database software that make a Database Management System work is relatively scarce.

This paper presents a discussion on database application functionality in terms of tiers of architectural design principles, transaction implementation, and characteristically shared components and utilities. Database software of today is becoming larger and more complex [7]. More powerful ways of structuring and assembling the areas of functionality are subsequently required, especially about development methodologies, structural programming, and software architecture.

This is because database software architecture is the outline of the system at the highest level of abstraction, describing the main components and their most important interactions [8]. To this end the architectural description provided in this paper willprovide a common understanding around which the quality of a database software can be ensured to a certain degree of performance. The section 2 describes the related work. In section 3, the tiers of database architectures were surveyed and program execution parameters are given. The survey findings and analysis are given in section 4. The approach for comparing the tiers are given in section 5, and the conclusions are discussed in section 6.

2. RELATED WORK

Several survey and analysisof software architecture have alreadybeen documented. Architecture of a DatabaseSystem; the authors [3] detail the critical architectural components and the multi-user potentials of database software. They realized that there has been many systems design techniques for scalability and reliability relating to database software, however database architecture coverage is scarce; in this light they presented architectural discussion of database software design principles, and parallel architecture. This complements our work in the sense of the architectural frameworks presented, the slight difference is on the areas were the current discussion focused more on the application functionality in terms of the tiers of architecture. Nicholas [5] in his dissertation presented a survey of software architecture viewpoint models.The work focused on methodology in the documentation of software architecture.

One method is to break upthe description into separate perspectives that address thedifferent concerns that stakeholders have with softwarearchitecture. These perspectives, sometimes called viewpoints,can contain multiple diagrams to describe the completesystem. Some viewpoint models were given to determinethe extent to which they cover the software architecturedomain. In this context, this paper is more interested in comparing the architectures in terms of robustness and application implementation. Rikard Land [7] also carried out a brief survey of generic software architectures. This discussion focused on software complexity, and the requirements for more powerful ways of structuring the complexity. Suitable component based architectures can be designed to handle change contribute greatly to solving software complexity elements including development methodologies, structural programming, naming conventions, and/or configuration management. Our focused on discussing the stored program concept as a bases for comparing the functionality elements in the three tiers of database software architectures presented.

3. SURVEY OF DATABASE ARCHITECTURES

In this section we describe different tiers of application such as one tier, two tier and three tier architectures and relate theme to a database software. One tier or monolithic applications are architectures where the code that implements the business rules, data access, and user interface are all tightly coupled together as part of a single, large computer program [6]. A monolithic application must be deployed on a single platform, usually a mainframe or midrange machine. Consider a user of a desktop computer who uses Microsoft Access to load up a list of personal addresses and phone numbers that he or she has saved in MS Windows' "My Documents" folder. This is an example of a one-tier database architecture.

The application Microsoft Access runs on the user's local machine, and references a file that is stored on that machine's hard drive, thus using a single physical resource to access and process information. Another example of a one-tier architecture is a file server architecture. In this scenario, a workgroup database is stored in a shared location on a single machine. Workgroup members use a software package such as Microsoft Access to load the data and then process it on their local machine [2]. In this case, the data may be shared among different users, but all of the processing occurs on the local machine.

Essentially, the file-server is just an extra hard drive from which to retrieve files. One-tier architectures can be beneficial when we are dealing with data that is relevant to a single user or small number of users and we have a relatively small amount of data. They are somewhat inexpensive to deploy and maintain. Monolithic applications are costly and time consuming to modify. It is more difficult to integrate applications to share services and data [9]. There is little reuse of redundant code between applications, making it more expensive to build and maintain applications. It is more difficult to have applications communicate with other applications. Deployment alternatives and interface flexibility is limited.

3.1Two Tier Client/Server Architectures

In a two-tier client-server architecture, application functionality is partitioned into two executable parts, or "tiers." One tier contains both the code that implements a graphical user interface (GUI) and the code that implements the business rules. This tier executes on PCs or workstations and requests data from the second application tier, which usually executes on the machine where the application's data is stored. Two-tier client-server applications suffer from many of the same drawbacks as monolithic applications and they aremore difficult and expensive to modify when business requirements change. They are more difficult to manage than monolithic applications. In two-tier client/server architecture, the client solely handles the user system interface [1]. The client communicates directly with the database server. In contemporary two-tier architectures, the processing logic either resides on the client or the database server in form of stored procedures.

Two-tier architecture is one that is familiar to many of today's computer users. A common implementation of this type of system is that of a Microsoft Windows based client program that accesses a server database such as Oracle or SQL Server (see fig. 1). Users interact through a GUI to communicate with the database server across a network via SQL (Structured Query Language). In two-tier architectures it is important to note that two configurations exist. A thin-client (fat-server) configuration exists when most of the processing occurs on the server tier [8]. Conversely, a fat-client (thin-server) configuration exists when most of the processing occurs on the client machine.

Vol 8. No. 4, December, 2015
ISSN 2006-1781

www.ajocict.net

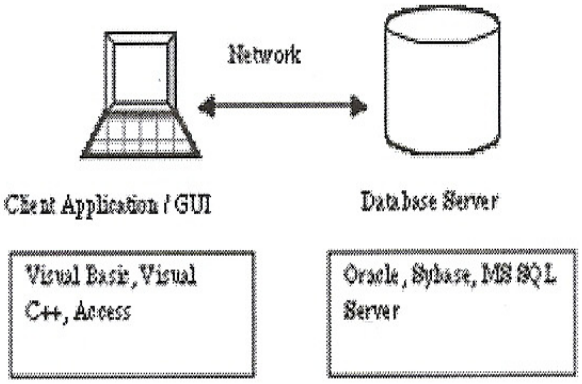

Figure 1 Two-Tier Client-Server Architecture

Another example of a two-tier architecture can be seen in web-based database applications. In this case, users interact with the database through applications that are hosted on a web-server and displayed through a web-browser such as Internet Explorer [11]. The web server processes the web application, which can be written in a language such as PHP or ASP. The web application connects to a database server to pass along SQL statements which in turn are used to access, view, and modify data (see fig. 2). The DB server then passes back the requested data which is then formatted by the web server for the user.Although this appears to be a three-tier system because of the number of machines required to complete the process, it is not.

The web-server does not normally house any of the business rules and therefore should be considered part of the client tier in partnership with the web-browser. Two-tier architectures can prove to be beneficial when we have a relatively small number of users on the system and when an increased level of scalability is desired.

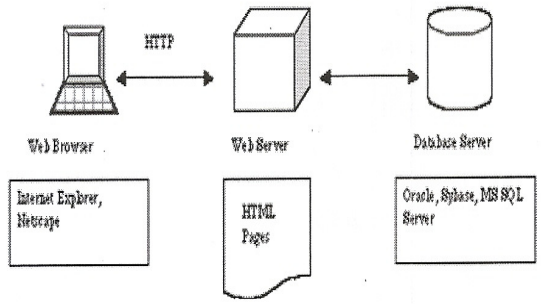

Figure 2 Web-Based, Two-Tier Client Architecture

3.2 Three Tier N- Tier Architectures

Most n-tier database architectures exist in a three-tier configuration. In this architecture the client/server model expands to include a middle tier (business tier), which is an application server that houses the business logic [12].

This middle tier as shown in figure 3 relieves the client application(s) and database server of some of their processing duties by translating client calls into database queries and translating data from the database into client data in return. Consequently, the client and server never talk directly to one-another.

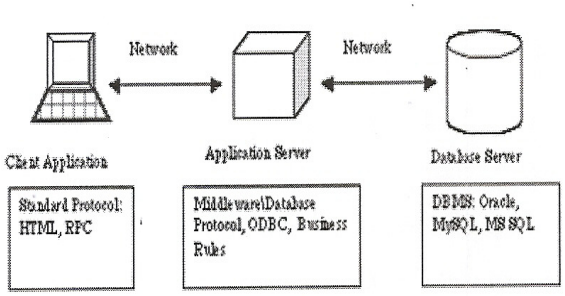

Figure 3 Three-Tier Client-Server Architecture

A variation of the n-tier architecture is the web-based n-tier application. These systems as illustrated in figure 4 combine the scalability benefits of n-tier client/server systems with the rich user interface of web-based systems. Because the middle tier in three-tier architecture contains the business logic, there is greatly increased scalability and isolation of the business logic, as well as added flexibility in the choice of database vendors.

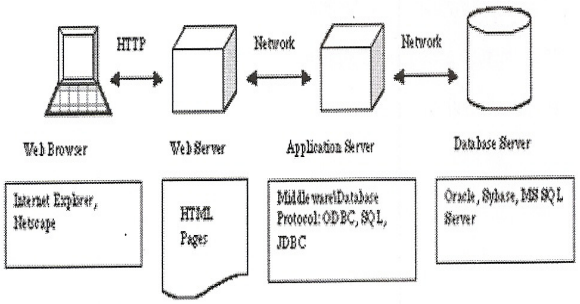

Figure 4 Web-Based, Three-Tier Client Server Architecture

The third tier contains database management functions. Its purpose is to optimize data and file services without having to resort to the usage of proprietary database management system languages. This component makes sure that the data is consistent throughout the environment. In order to do so, it utilizes such features as data locking, replication, and consistency. The connectivity among tiers can be changed dynamically, but of course this depends on the user's request for services and data. The middle tier on the above model provides process management services which will be shared by multiple applications. These services may include process enactment, process resourcing, process development, and process monitoring [11].

This tier also serves so as to improve performance. It is also called the application server. It improves scalability, reusability, flexibility, and maintainability via the centralization of process logic.

This centralization makes change management and administration a lot simpler by localizing the functionality of the system so that changes only have to be written once. They are then placed on the central tier and made available throughout the systems. With other architectural designs, it would be necessary to write the change into each and every application. The central process management tier also serves as a controller of asynchronous queuing and transactions. This thus ensures that transactions will be completed in a reliable fashion [13]. The middle tier successfully manages to distribute database integrity through a commit process that occurs in two phases. Access to resources based on names, rather than locations, are provided. Thus, an improvement of flexibility and scalability results as the components of a system are either moved or added.

4. SURVEY FINDINGS

The most basic type of three tier architecture has a middle layer consisting of Transaction Processing (TP) monitor technology. The TP monitor technology is a type of message queuing, transaction scheduling, and prioritization service where the client connects to the TP monitor (middle tier) instead of the database server. The transaction is accepted by the monitor, which queues it and then takes responsibility for managing it to completion, thus freeing up the client. When the capability is provided by third party middleware vendors it is referred to as "TP Heavy" because it can service thousands of users. When it is embedded in the DBMS (and could be considered a two tier architecture), it is referred to as "TP Light"; because usually, there is performance degradation whenever a large number of clients up to 100 clients are connected. TP monitor technology provides the ability to update multiple different DBMSs in a single transaction, provides connectivity to a variety of data sources including flat files, non-relational DBMS, and the mainframe.It also provides the ability to attach priorities to transactions, and robust security [8].

Using three tier client/server architecture with TP monitor technology results in an environment that is considerably more scalable than a two tier architecture with direct client to server connection. For systems with thousands of users, TP monitor technology (not embedded in the DBMS) has been reported as one of the most effective solutions. The three tier application server architecture allocates the main body of an application to run on a shared host rather than in the user system interface client environment [6]. The application server does not drive the GUIs; rather it shares business logic, computations, and a data retrieval engine. Advantages are that with less software on the client there is less security to worry about, applications are more scalable.

Support and installation costs are less on a single server than maintaining each on a desktop client. This application server design is a very necessary consideration when security, scalability, and cost are major concerns

4.1 The Message Server and ORB Architecture

Messaging is another way to implement three tier architectures. Messages are prioritized and processed asynchronously. Messages consist of headers that contain priority information, and the address and identification number. The message server connects to the relational DBMS and other data sources. The difference between TP monitor technology and message server is that the message server architecture focuses on intelligent messages, whereas the TP Monitor environment has the intelligence inthe monitor, and treats transactions as dumb data packets. Currently industry is working on developing standards to improve interoperability. Developing client/server systems using technologies that support distributed objects holds great promise, as these technologies support interoperability across languages and platforms, as well as enhancing maintainability and adaptability of the system [5].

There are currently two prominent distributed object technologies: Common Object Request Broker Architecture (CORBA), Component Object Model (COM), and Related Capabilities). Industry is working on standards to improve interoperability between CORBA and COM. Three-tier applications are much more difficult to build than two-tier applications. The biggest obstacle is that the software tools' integrated development environments are not aware of the three-tier model. As a result, much more hand-coding is required to write a three-tier application [7]. Three-tier applications are also harder to design, because they are somewhat abstract compared with their more direct two-tier counterparts. Software tool vendors are starting to release new versions for three-tier or n-tier development support.

5. THE APPROACH FOR COMPARING DATABASE ARCHITECTURES

The comparison approach used in this paper is based on the stored program concept. Stored procedures are user-written structured query language (SQL) programs that are stored at the data base server and can be invoked by client applications. A stored procedure can contain most statements that an application program usually contains. Stored procedures can execute SQL statements at the server as well as application logic for a specific function. A stored procedure can be written in many different languages, and the language in which stored procedures are written depends on the platform where the data base server is installed. Local client applications, remote Distributed Relational Database Architecture (DRDA), or remote data services can invoke the stored procedure by issuing the SQL CALL statement. The client program can pass parameters to the stored procedure and receive parameters from the stored procedure [10].

The client program and the stored procedure do not have to be written in the same programming language. For example, a C client program can invoke a COBOL stored procedure. In previous releases of DRDA, the client system performed all application logic.
The server was responsible only for SQL processing on behalf of the client. In such an environment, all database accesses must go across the network, resulting in poor performance in some cases.

This is a relatively simple model, which makes the application program easy to design and implement. Because all application code resides at the client, a single application programmer can take responsibility for the entire application [3]. However, there are some disadvantages to using this approach. Because the application logic runs only on the client workstations, additional network input/output (I/O) operations are required for most SQL requests. These additional operations can result in poor performance. This approach also requires the client program to have detailed knowledge of the server's database design. Thus, every change in the database design at the server requires a corresponding change in all client programs accessing the database. Also, because the programs run at the client workstations, it is often complicated to manage and maintain the copies there. Stored procedures enable you to encapsulate many of your application's SQL statements into a program that is stored at the data base server.

The client can invoke the stored procedure by using only one SQL statement, thus reducing the network traffic to a single send and receive operation for a series of SQL statements [5]. It is also easier to manage and maintain programs that run at the server than it is to manage and maintain many copies at the client machines. Stored procedures enable you to split the application logic between the client and the server. This technique can be used to prevent the client application from manipulating the contents of sensitive server data. It is therefore important to note that three tier architectures have considerable advantage of flexibility over other architectures.

6. CONCLUSION

Typical database software are ubiquitous and contains thebusiness rules processing, data access, and presentation or interfaceareas of functionality. For proper database application functionality, it is becoming more important to structure the inter mechanisms for optimal performance. This structuring is here discussed in line with application architectures to come up with a comparison relating to transactional processing and the stored program concept in a database software. It is intended to provide a common understanding around which the quality of a database software can be ensured to a certain degree of performance.

REFERENCES

[1] Elliott, R. &Powers, N., (2002) (Intellex), "One -Tier, Two-Tier, Three-Tier, A Server: Using Technology to Solve Business Problems", http://www.pacific-electric.comlPacificEleclProduct/whtpap04.htm
[2] Groff, J.R. and Weinberg, P.N., (Osborne McGraw Hill, 1990), Using SQL, pp. 277
[3] J. M. Hellerstein, M. Stonebraker and J. Hamilton (2007) Architecture of a Database System Foundations and Trendsin Databases Vol. 1, No. 2 141–259 2007 DOI: 10.1561/1900000002
[4] Paulsell, K. and Deering, B., (Sybase, Inc., 1992), Commands Reference Manual for Sybase SQL Server, pp. 2-76.
[5] Nicholas May (2004) A Survey of Software Architecture Viewpoint Models dissertation expansion of Master of Technology (Information Technology) at RMIT University Melbourne, Australia nick may@netlink.com.au
[6] Hemmer, F.M. (1993) "RHIC Electronic Data Collection and survey &Alignment Database" Proceedings of the Third International Workshop On Accelerator Alignment, Annecy, pp197-230
[7] Rikard Land (2002), A Brief Survey of Software Architecture Mälardalen Real-Time Research Center (MRTC) Report Department of Computer Engineering, Mälardalen University, Västerås, Sweden, February 2002 rikard.land@mdh.se
[8] M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin, K. P. and V.Watson, "System R: Relationalapproach to database management," *ACM Transactions on Database Systems(TODS)*, vol. 1, pp. 97–137, 1976.
[9] P. A. Bernstein and N. Goodman, "Concurrency control in distributed databasesystems," *ACM Computing Surveys*, vol. 13, 1981.
[10] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows,T. Chandra, Fikes, and R. E. Gruber, "Bigtable: A distributed storagesystem for structured data," in *Symposium on Operating System Design andImplementation (OSDI)*, 2006.
[11] S. Chaudhuri and U. Dayal, "An overview of data warehousing and OLAP technology," *ACM SIGMOD Record*, March 1997.
[12] M.-S. Chen, J. Hun, and P. S. Yu, "Data mining: An overview from a database perspective," *IEEE Transactions on Knowledge and Data Engineering*, vol. 8,1996.
[13] H.-T. Chou and D. J. DeWitt, "An evaluation of buffer management strategiesfor relational database systems," in *Proceedings of 11th International Conferenceon Very Large Data Bases (VLDB)*, pp. 127–141, Stockholm, Sweden,August 1985.