Comp. by: MGrahalatshmiGalleys0000874184 Date:19/11/08 Time:17:15:12 Stage:First Proof
File Path://ppdys1108/Womat3/Production/PRODENV/0000000005/0000008302/0000000016/
0000874184.3D Proof by: QC by:

Multi-Tier Architecture

HEIKO SCHULDT
Database and Information Systems Group,
Department of Computer Science, University of Basel,
Basel, Switzerland

Synonyms

n-Tier architecture; Multi-layered architecture

Definition

A *Multi-tier Architecture* is a software architecture in which different software components, organized in tiers (layers), provide dedicated functionality. The most common occurrence of a multi-tier architecture is a three-tier architecture consisting of a data management tier (mostly encompassing one or several database servers), an application tier (business logic) and a client tier (interface functionality). Novel deployments come with additional tiers. Web information systems, for instance, encompass a dedicated tier (web tier) between client and application layer.

Conceptually, a multi-tier architecture results from a repeated application of the client/server paradigm. A component in one of the middle tiers is client to the next lower tier and at the same time acts as server to the next higher tier.

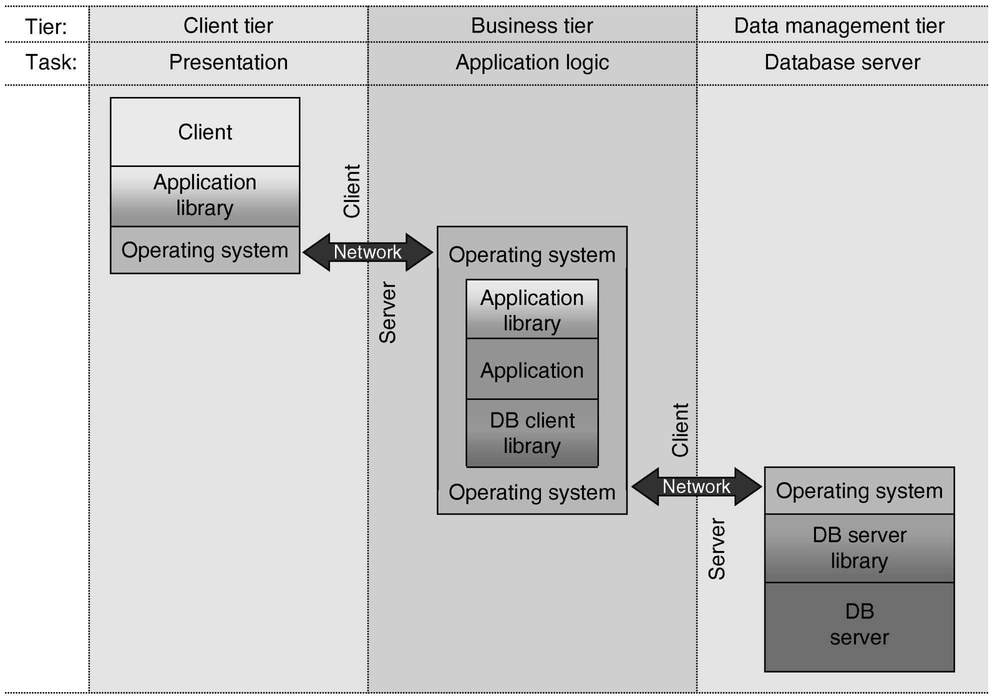
Historical Background

Early generation software systems have been built in a monolithic way. This means that all the different tasks for implementing a particular application and presenting the results to a user are provided by a single dedicated software component. With the advent of client/server architectures in the 1980s, different tasks could be separated and possibly even be distributed across network boundaries. In a client/server architecture (two tier architecture), the client is responsible for presenting the application to the user while the server is in charge of data management. For the provision of business logic, two alternatives have emerged. First, in so-called fat client/thin server architectures, the client also provides business logic, in addition to presentation and user interfaces. This can be realized by using SQL against the underlying database server in the application program run by the client, either by embedding SQL into a higher programming language or by using the database server's call level interface (e.g., JDBC, ODBC). Second, in thin client/fat server architectures, the database server also provides business logic while the client solely focuses on presentation issues. Fat servers can be realized by using persistent stored modules or stored procedures inside the database server. In the case of evolving business logic, fat client architectures, although being the most common variant of client/server systems, impose quite some challenges when new client releases need to be distributed in large deployments. In addition, a fat client architecture usually comes along with a high network load since data is completely processed at the client side. Fat servers, in contrast, impose a single point of failure and a potential performance bottleneck.

Three-tier architectures thus are the next step in the evolution of client/server architectures where both client and database server are freed from providing business logic. This task is taken over by an application layer (business tier) between client and database server. In multi-tier architectures, additional tiers are introduced, such as for instance a web tier between client and application layer.

Scientific Fundamentals

Multi-tier systems follow an architectural paradigm that is based on separation of concerns. The architecture considers a vertical decomposition of functionality into a stack of dedicated software layers. Between each pair of consecutive layers, a client/server style of interaction is applied, i.e., the lower layer acts as server for the next higher layer (see Fig. 1). Typical tiers in a three-tier architecture are *data management*, *business* and *client* tier. Multi-tier architectures consider



Multi-Tier Architecture. Figure 1. Structure of a three tier architecture.

additional layers, such as a web tier which hosts servlet containers and a web server and which is located between client tier and application tier.

In addition to vertical decomposition and distribution across tiers, in many cases multi-tier architectures also leverage horizontal distribution within tiers. For the data management tier, this means that several distributed database servers can be used. Most commonly, horizontal distribution is applied at the business tier, i.e., providing several application server instances [7].

The main benefit of multi-tier applications is that each tier can be deployed on different heterogeneous and distributed platforms. Load balancing within tiers, especially for the application tier, is supported by distributing requests across the different application server instances. This can be implemented by a dispatcher which accepts calls from the next higher layer and distributes them accordingly (this is done, for instance, in TP Monitors which allow to distribute requests among application processes at the middle tier in a three-tier architecture).

When multi-tier architectures are used in a business context, they have to support transactional interactions. Due to the inherent distribution of software components across layers and potentially even within layers, distributed transactions are needed. This is usually implemented by a two-phase commit protocol (2PC) [5] (depending on the application server and the middleware used, this can be done, for instance, via CORBA OTS, the Java Transaction Service JTS, etc.). While 2PC provides support for atomicity in distributed transactions, it does not take into account the layered architecture where transactions at one layer are implemented by using services and operations of the next lower layer. Multi-level transactions [11] take this structure into account. SAP ERP [4], for instance, applies multi-level transactions by jointly considering the application server and data management tier. Asynchronous interactions between components in a multi-tier architecture require message-oriented middleware (MOM). In this case, transactional semantics can be supported by persistent queues and queued transactions [1].

In order to increase the performance of multi-tier systems and to improve response times, caching is used at the application tier. For this, different database technologies such as replication, materialized views, etc. can be applied outside the DBMS [6].

Key Applications

Due to the proliferation of both commercial and open source application servers, multi-tier architectures can

Comp. by: MGrahalatshmiGalleys0000874184 Date:19/11/08 Time:17:15:13 Stage:First Proof
File Path://ppdys1108/Womat3/Production/PRODENV/0000000005/0000008302/0000000016/
0000874184.3D Proof by: QC by:

be found in a very large variety of different domains. Applications include, but are not limited to, distributed information systems, Web information systems, e-Commerce, etc.

Experimental Results

The Transaction Processing Performance Council (TPC) has defined a benchmark, *TPC-App*, for evaluating the business tier and in particular the performance of application servers in a three- or multi-tier architecture [10]. It includes Web Service interactions, distributed transactions, and asynchronous interactions via message-oriented middleware (reliable messaging and persistent queues).

Cross-references

► Application Server
► Client/Server Architecture
► Distributed Transactions
► Message-Oriented Middleware
► Middleware
► Persistent Queues
► Three-Tier Architecture
► Web Services
► Web Transactions

Recommended Reading

1. Bernstein P. and Newcomer E. Principles of Transaction Processing. Morgan Kaufmann, Los Altos, CA, 1997.
2. Birman K. Reliable Distributed Systems: Technologies, Web Services, and Applications. Springer, Berlin, 2005.
3. Britton C. IT Architectures and Middleware. Addison Wesley, Reading, MA, USA, 2001.
4. Buck-Emden R. and Galimow J. SAP R/3 System: A Client/Server Technology. Addison-Wesley, Reading, MA, USA, 1996.
5. Lindsay B., Selinger P., Galtieri C., Gray J., Lorie R., Price T., Putzolu F., and Wade B. Notes on Distributed Databases. IBM Research Report RJ2571, San Jose, CA, USA, 1979.
6. Mohan C. Tutorial: ACaching Technologies for Web Applications. In Proceedings of the 27th International Conference on Very Large Data Bases (VLDB'01), Rome, Italy, 2001, http://www.almaden.ibm.com/u/mohan/Caching_VLDB2001.pdf.
7. Mohan C. Tutorial: Application Servers and Associated Technologies. In Proceedings of the 28th International Conference on Very Large Data Bases (VLDB'02), Hong Kong, China, 2002, http://www.almaden.ibm.com/u/mohan/AppServersTutorial_VLDB2002_Slides.pdf.
8. Myerson J. The Complete Book of Middleware. Auerbach, Philadelphia, PA, 2002.
9. Orfali R., Harkey D., and Edwards J. Client/Server Survival Guide. Wiley, 3rd edn., 1999.
10. Transaction Processing Performance Council.TPC-App. http://www.tpc.org/tpc_app/default.asp, 2008.
11. Weikum G. and Schek H.J. Concepts and Applications of Multilevel Transactions and Open Nested Transactions. In Database Transaction Models for Advanced Applications. Morgan Kaufmann, Los Altos, CA, 1992, pp. 515–553.
12. Weikum G. and Vossen G. Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control. Morgan Kaufmann, Los Altos, CA, 2001.