On the Evolution of Software Architectures

VASSILIOS C. VESCOUKIS
National Technical University of Athens
Department of Electrical and Computer Engineering
15780 Zographou, Athens
GREECE

Abstract : Software is a complex technical construction that has been developed and used for the past few decades. The evolution of software applications has followed that of computer hardware and today, some sort of software application exists in most artificial systems. In their early days, software applications were usually quite simple, compared to today's giant and complex applications, and the difficulties in their construction focused on very fundamental issues. After the 80's and even more during the 90's, software applications have become very complex. Their development is supported by sophisticated tools and their runtime environments are much more advanced than in the early days. There are more than one choice on how to structure a software application and questions that a few years ago had only one answer, today have fairly more. The issue of software architecture has risen and several architectures have been proposed in order to develop better and more effective software. This paper is a review of the most important of the software architectures from the monolithic to the fully distributed one, focusing on the evolution, the advantages and the shortcomings of each one of them. IMACS/IEEE CSCC'99 Proceedings, Pages:7431-7437

Keywords: Software engineering, distributed systems, web-based software applications

1 Introduction

For the construction of any artificial item, a full description and a design has to be preceded. This design can be more or less detailed and can be drawn from many different visual angles depending on its designated purpose. After the design process has finished, a set of designs describes the different perspectives of the construction with more or less detail, depending on whom each one appeals to. These represent an abstract description of some characteristics of the artifact under construction. When the artifact is solid the perception of design can be almost straightforward, but this is not the case in invisible constructions like software. Software designs, even "simple" ones, usually can be perceived in many different ways, and have no one single interpretation.

Things become even more complicated, when software runs in a distributed runtime environment, like an intranet or the Internet. This idea is not an innovative one, as there is a tendency for software applications to function in a distributed network environment, not only in terms of deployment (that is, network connection among computers) but also in terms of functionality. In this context, the notion of “software architecture” is not any more a wishful thought. It became the centre of attention in the development of many software applications.

1.1 Software architectures

With the term “software architecture” we refer to the description of the division of an application into parts (building blocks) and to the interoperability among them, so that the tasks required from software can be accomplished [2, 8]. What exactly a building block of software is (procedure, routine, function, class, library, etc), depends on the specific implementation details. The same stands for the term "interoperability" which can be regarded as procedure or function call, message, responsibility, etc, depending again on the implementation details. The disciplinary or non-disciplinary, documented or not architectural structure of an application, is determined by the designer (actually the "designer" here describes a role instead of a single person).

Whatever the software development discipline followed is (if any), the division of an application into functional elements is made in order to "better" implement the functional and non-functional requirements in a specific implementation environment, as well as to satisfy other conditions like maintainability, extensibility, reusability, etc. The adjective "better" here, can be viewed from several angles depending on the culture of each developer. In the course of software history, there have been several interpretations of what "good" software is, however there is little doubt that the debate is still on.

In the sequence we shall refer to the mainstream software architectures as they have been introduced in the evolution of software development. We identify four architectural categories: the monolithic, the client-server, the three-tier and the multi-tier generic architecture. Before we do that, it is important to divide software functions in categories (or layers, although the layering of software is not always clear). The assignment of these functions to discrete software modules is what will be used for the definition of software architecture categories in the sequel.

1.2. Software layers

By the use of the term "function" here we do not refer to some specific functional requirement, but to a generic category of operation that has to be performed regardless of the individual requirements imposed by the application domain. We identify three such categories: business logic, data management and presentation (figure 1). Tasks like the implementation of security and access rights can be applied in any (or in all) of the aforementioned categories or directly in the operating system and without affecting the purpose of the argumentation to follow, will not be discussed.

The first such category is actually where the purpose for which we build the software is fulfilled and is called business logic. This definition is not widely accepted in academic environments where the term functional requirement is more acceptable. It is in the business logic layer, where the implementation of all the functions (requirements) of a software application is done.

The second one is the data management layer and it is where all the data-related functions, such as storage, retrieval, duplication, etc, are implemented. Obviously, not all software applications perform such functions since not all applications deal with persistent data. However, most business applications do handle persistent data and in the sequel, without affecting the generality, we shall assume that a data management layer does exist in all software applications.

Finally, we distinguish the presentation level that is, that part of software where the flow control and the user and other external interfaces are implemented. Again, without harm to the generality, we shall assume that all software applications do have such a layer.

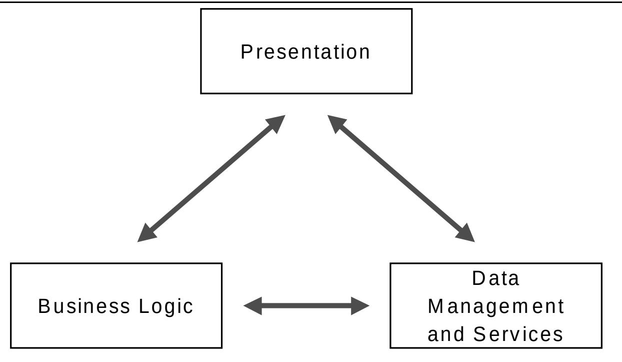

Figure 1. Layering of software functions

Not all of the aforementioned layers can be distinguished in all software applications [5]. In not a few cases all of these functions are mixed-up in software elements, so that it is not possible to determine which layer or category a software element belongs in.

2 Software architectures

2.1 Monolithic architecture

The simplest of all software architectural schemas is the monolithic one, which dominated in the early days of computing [2, 3]. Monolithic software applications run in their entirety in one single main computer, as shown in figure 2. By this we mean that all the application-related tasks run in the same computer and make use of its resources such as memory and CPU.

In the early days of computing the single monolithic structuring was the only available way for structuring software applications: there was one single specific computer where the application was intended to run. Interoperability and distribution where unknown notions and there was one single purpose: that the application runs. In this category we classify the legacy applications of mainframes that communicate with the user through dump terminals, custom-made applications that ran on PCs, off-the-shelf shrink-wrapped applications. Such applications have been made very "popular" in the late eighties, when the PC has entered in the small business offices.

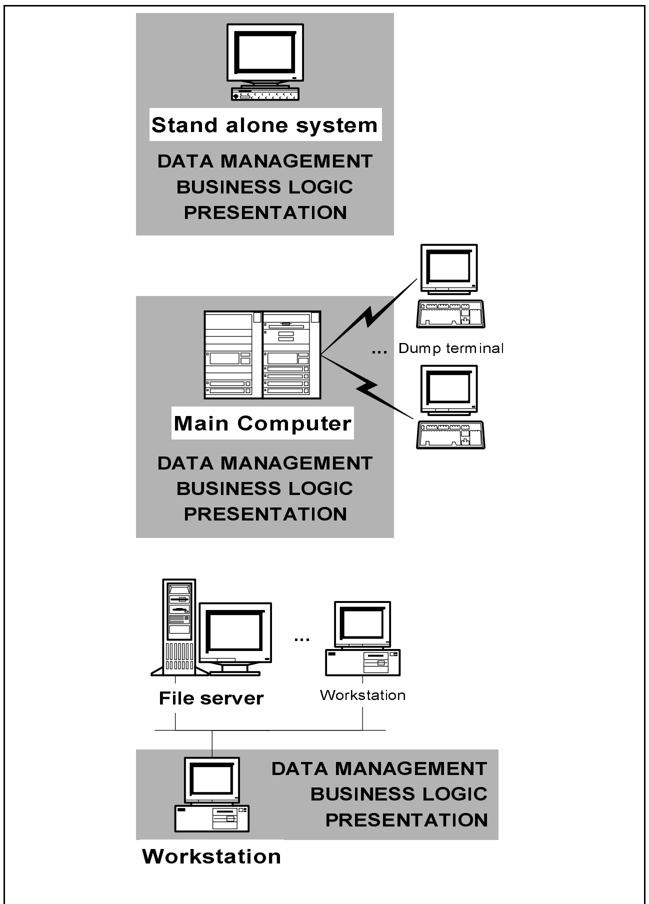

Figure 2. Monolithic applications

We also classify here applications that make use of some remote storage - not computing power - which is available through a network file server. All the operations of monolithic applications are performed on the same CPU and memory space, as shown in figure 2. In the late seventies and even more during the eighties, the evolution of computer networks and the appearance of the first data base management systems, have defined the framework in which the first distributed software architecture has been developed.

2.2 Client-server schemas

The early monolithic giant applications have been replaced by more numerous but smaller ones that co-operate with each other accessing the same data, which is stored in a database server. This scheme was named client-server and has been popular for quite a long time. In fact, most of today's corporate applications still follow the client-server paradigm. In this schema, the data management and perhaps some of the business logic is responsibility of the database server. The presentation and the (rest of) business logic are the client's responsibility (figure 3).

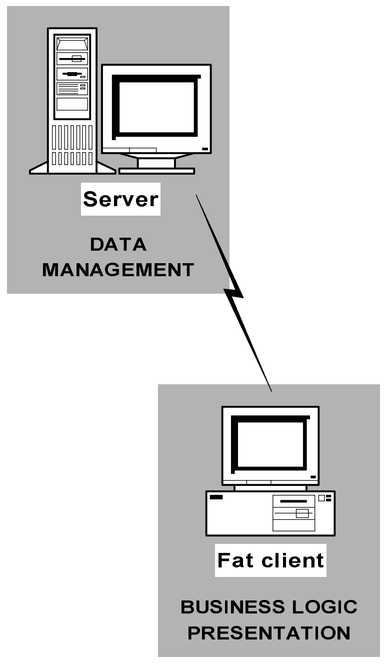

Figure 3. The client-server paradigm

In the client-server paradigm, each application has two discrete and functionally independent parts: the client and the server. Each one runs on a separate computer or as a discrete task in the same computer. They do not require each other in order to exist, and although from the user's point of view they are not of use unless they work together, technically they are completely independent pieces of software that may run in different operating systems.

The client-server paradigm has dominated for quite a long time, and in many cases is still dominant. One of its major shortcomings is the cost of maintenance of numerous clients with every change in the business logic or in the presentation layer implementation, as requirements evolve over time. Another one, is the size of the client itself, which constantly increases as applications become more and more complex and feature-loaded. This kind of client, the "traditional" one, is referred to as "fat client".

2.3 Three-tier architectures

Lately a more refined instance of this schema has been introduced, based on the different allocation of the fat client responsibilities (in fact on the split of these responsibilities). The client-server model has evolved to a three-tier schema as shown in figure 4. In this case, the servers are two: the database server, that operates as in the classic client-server model and the application server on which runs a big part or all the software functions in the business logic level. The responsibility of the presentation still is on the client side, which is connected either only to the application server or both to the application and database servers. This kind of client is referred to as “thin client”.

In a more generic version of this schema, more than one application server may exist, forming an even more distributed computing paradigm. This is the case of the multi-tier architecture using thin clients and should not be confused with the multi-tier web-based architecture whose description is to follow.

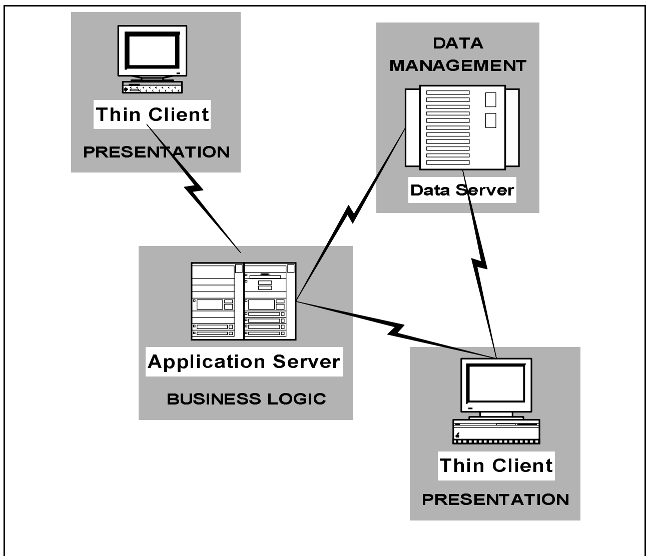

Figure 4. The three-tier thin client paradigm.

This schema eliminates several shortcomings of the traditional client-server model. The biggest part of the maintenance is done more in the database and/or application servers and less in the client, which results in the decrease of the maintenance cost. This is because (a) the frequency of interventions in the client is generally lower and (b) the client's fixed cost is less since the computational and storage load have been transferred elsewhere (to the application and the database server, respectively). However, the client still needs maintenance, especially whenever changes to the presentation layer need to be implemented.

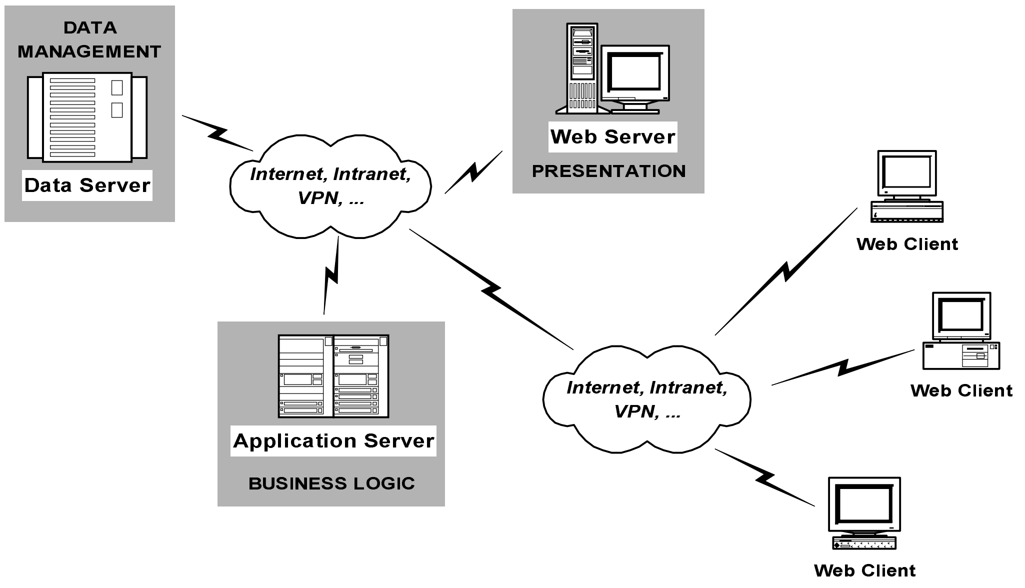

Figure 5. A multi-tier web-based software architecture

2.4 A multi-tier web-based schema

We are already in the nineties and the networks are in the era of the big explosion that led to today's global domination of the Internet. The web technologies have appeared and the www, which was introduced as a simple distributed presentation framework, evolved to become a mature, interactive and complete application environment. The main difference from the multi-tier thin client schema mentioned above, is that here no client (thin or fat) exists. A web browser takes its place, which carries no application-specific responsibility at all. The presentation responsibility is now assigned to a web server to which the web browser is connected. This means that no application-specific software needs to be installed in the client's side: a suitably tuned generic web browser can do the job.

The idea is called web client (figure 5) and the main innovation it carries along is the transfer of the presentation services from a custom (no matter whether thin or fat) client to a suitably tuned web server. This is shown in figure 5, where in the same company environment exist more than one server from each category. The intranet/internet connections can be any suitable for the company needs (for example a LAN or VPN).

The maintenance of the software applications can be done totally in the premises where the servers are located, that is in the corporate computer room. More than one servers can be active in a corporate computing environment, depending on the general architecture of the company's intranet, on the available machines, on the distribution of the services, on the security requirements, etc.

3 A new computing paradigm

Most of current business software applications follow the client-server paradigm. Personal desktop applications are still monolithic, although they are modular, that is, there are several application modules that execute at the same time, making use of the current multi-tasking OSs. Today there is an increasing interest in the development and deployment of web-based applications. Indeed, many software applications classified as ERP (Enterprise Resource Planning) begin to adopt the web client concept.

Furthermore, what is actually rising is a completely new computing paradigm that will be fully distributed and quite flexible. The web client concept is not applicable only when all the distributed software fragments are constructed by the same developer. Co-operating distributed software modules can be of several developers or even application service providers. An integrator can collect application and data services from all around the Internet, add some value in terms of business logic and presentation services and provide a new service to his own clients. The clients will pay for the service the integrator, who in turn will pay for the application and database services he has used. This concept is shown in figure 6.

As an example, one can imagine the sociological analysis of sales data using many different methods. One choice for the structuring of such an application is to embed the related data and all the different analysis algorithms, providing a complete solution. Instead of doing this, a developer can buy the data services as well as the analysis services from independent sources, build a communication framework and a web-based user interface and rapidly deliver a new service to the market. This way is not only faster, but also much more flexible. Each part does exactly what specialises in, no unneeded data redundancy costs exist, and the computational resources available to the customer are more than they would be if only one developer had to create software for all the possible data analysis methods.

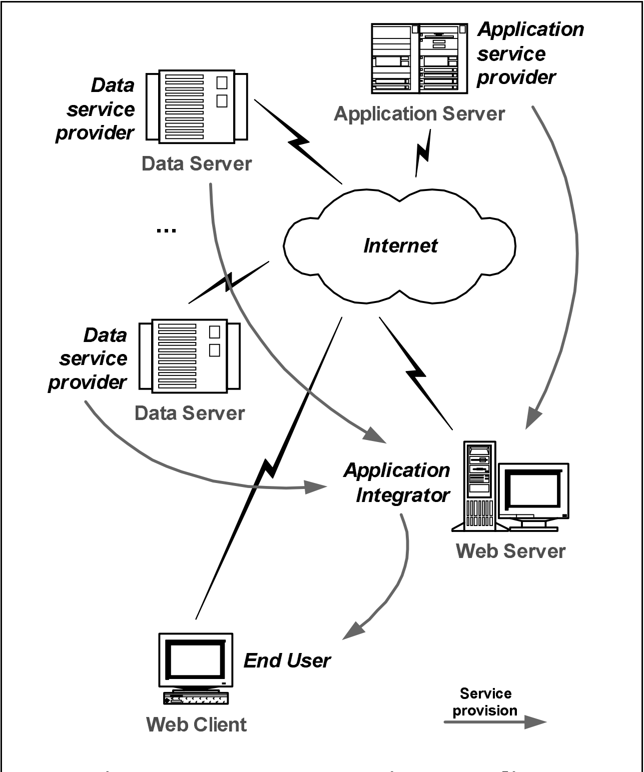

Figure 6. A new computing paradigm

Of course, much work has still to be done before such a computing paradigm can dominate or even exist in a measurable scale. The network availability, reliability, speed and security are the obvious technical problems the solution of which will undoubtedly push things ahead. More serious problems exist in the software domain where things are still somehow confused in terms of standards, APIs, services and even development environments. Technologies like COM/DCOM, Java and scripting strive to gain acceptance from developers. The large installed basis of legacy systems and the non necessarily standard middleware that is used, do not allow the easy interoperability that the web-based distributed application schema requires.

The description of software architectures had not been standard for a long time and several different notations have been used to denote more or less the same things. Lately, the Unified Modelling Language (UML) has been accepted by the OMG as the standard notation schema for object-oriented software systems (in fact UML can be used for the modelling of non-software artefacts as well). In figure 7, the equivalent of the notation-free diagram shown in figure 5, is presented using the UML notation [7]. In the UML terminology [6], this is called a deployment diagram.

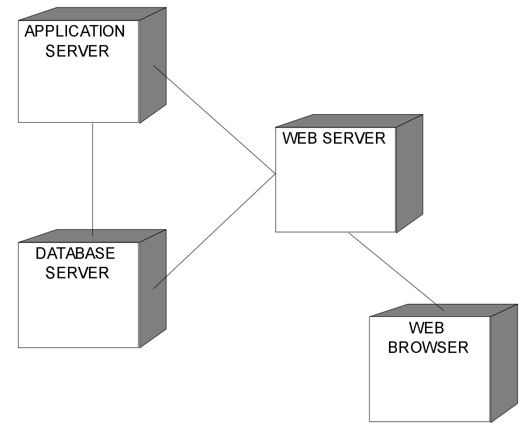

Figure 7. A UML deployment diagram.

5 Concluding remarks

Software engineering has gone a long way from the early days up to our days. The monolithic applications which became giant legacy applications can no longer satisfy the requirements of computing today. On the way, several cooperative paradigms have been introduced, all under the generic name "client-server". The evolution of global networks, basically of the Internet, has created the basis for a completely new, truly distributed computing paradigm.

In the future, more and more existing software applications will migrate to this new paradigm. What is even more impressive, is that this paradigm will allow the conception of new ideas for new software applications and services. In fact, several new-generation applications are currently under design [5] and development and one can only say that "the best is yet to come"._

References

[1] Agresti, W.W., *"New Paradigms for Software Development"*, IEEE Computer Society order number 707, IEEE Computer Society Press, 1986.

[2] Fairley, R.E., *"Software Engineering Concepts"*, McGraw-Hill, 1985

[3] Sommervile, Ian, *"Software Engineering"*, Fourth Edition, Addison-Wesley, 1995.

[4] S.Retalis, V.C.Vescoukis and E.Skordalakis, *"An Object-Oriented Data Model for Web-*

Based Courseware Design", in "Software and Hardware Engineering for the 21st Century", N.Mastorakis (editor), World Scientific, 1999.

[5] De Paoli, Flavio and Sosio, Andrea, *"Requirements for a layered software architecture supporting cooperative multi-user interaction"*, in proceedings of the 18th International Conference on Software Engineering (ICSE), IEEE Press, 1996.

[6] G. Booch, J.Rumbaugh and I.Jacobson, *"The Unified Modelling Language User Guide"*, Addison-Wesley, 1999.

[7] J.Rumbaugh, I.Jacobson and G. Booch, *"The Unified Modelling Language Reference Manual"*, Addison-Wesley, 1999.

[8] Roger Pressman, *"Software Engineering. A practitioner's approach"*, Mc Graw Hill, 1997.