Architectural Model for User interfaces of Web-based Applications

Petr Hejda

Rockwell Automation
Research Center Prague
Czech Republic
phejda@ra.rockwell.com

Abstract: Web-based applications are by far the most universally accessible. These applications evolved from simple document retrieval systems to online booking systems, online shopping malls, e-commerce websites, and collaborative systems, to name just few examples. Widely accepted software and networking standards allow for effective implementation of these applications. However, these standards do not help much with making the interface of these applications user friendly and easy to manage. In other words, there are no or little means to ensure that the user interface provides high quality in use for all users. We address this problem by introducing an architectural model for user interfaces of web-based applications. The model reflects both the physical partitioning of the application as well as the placement of user interface components. The presented model is a synthesis of two groups of models. The first group contains multi-tier architecture models currently used in the design of web-based applications. The other group of models consists of traditional models developed by the HCI community. Our model differentiates between local and remote communication lines. It can be further customized to fit concrete examples of web-based application types.

1. Introduction

Distributed applications have gained a lot of attention in the past ten years. They are held responsible for the phenomenal growth of the Internet. Among these applications, the most common and known ones are so called web-based applications. These applications evolved from simple document retrieval systems to online booking systems, e-commerce websites, online shopping malls, and collaborative systems, to name just few examples.

Web-based applications share some common features. They are **distributed**, meaning the computation takes place in different **physical locations**. Typically, the user interface runs on a different computer than logic and data storage. The client-server architecture is used as a basic distribution paradigm. User interface is implemented as a **thin, universal, and extendable client**. These clients are called browsers in the web community. Clients store persistently almost no data. The rest of the application employs re-usable parts; both **the server and data store** are universal software components. The application specific knowledge is implemented as **data, server extensions, and scripts** which are run by both client and server parts. To allow for integration of re-usable components, **open standards** are employed. The last feature, which differentiates web-based applications from general computer applications, is their **continuous run**. The application does not shut down once a user is finished with his/her task. Instead, at least some parts have to run continuously to serve other users.

While developing web-based applications, designers have the freedom to choose from a wide range of features and their possible implementations. Standard and non-standard components can be combined in many ways to achieve various effects and functionality. For example, let's say that the application has to have a "commit" option. If user chooses "commit" a certain action is triggered, if he chooses "cancel" nothing happens. This option can be implemented in many ways, for instance by two Hyper Text Markup Language (HTML) hyper links [Musciano 98], or by two buttons in an HTML form, or by two buttons

implemented with JavaScript, or even by a simple Java applet. What is the most effective way? How to decide? What are the criteria?

Architectural models can be of great advantage here. They can provide conceptualization of the architecture, define basic building blocks, and relate them to each other [Buschmann 96]. However, there are no architectural models that can be directly used for user interface issues in web-based applications.

There are two groups of architectural models coming from two communities, which seem to be related to our problem. The Human-Computer Interaction (HCI) community has produced many architectural models that address user interface issues [Dix 98]. Multi-tier models were developed to help to resolve issues regarding physical distribution of application components.

Arch [Arch 92] is a well-known and accepted architectural model dealing with user interface issues. It specifies logical components of the user interface. Arch covers the whole application, from the part interfacing users to the functional core. However, it does not specify the physical partitioning of the system into networked components. As a result, this model is not expressive enough for web-based applications where the physical partitioning plays a critical role in application design.

The other related group of models contains multi-tier models [Pattison 98]. They have been used for distributed applications consisting of multiple cooperating layers. A tier is any part of the application which can communicate remotely with other parts. The main concern of these models is communication. They define how the system is partitioned and the possible ways the parts can communicate with each other. For example, a three-tier model [RDS 97] divides the application into a client, server, and data store tier. Multi-tier models do not deal with user interface issues. They do not target mapping of tiers and user interface components.

Both Arch and three-tier models can be of help while developing web-based applications. In our model, we have combined both models into one. This results into a general model. It deals with both physical partitioning as well as user interface components. This model can be further customized to fit various implementation techniques and paradigms.

2. General model

In the previous section, we have already mentioned two main requirements an architectural model should address to be useful for web-based applications. The model should reflect both the physical distribution of the application and logical partitioning of user interface components.

The physical distribution can be borrowed from the three-tier model. We have used the term **tier** for distinctive parts of the application, which communicate remotely. In a similar way, the term **component** is used for a logical part of the user interface of the application as defined by Arch. This tier/component dichotomy needs to be integrated into one model. Unfortunately, there is no mapping which would assign some user interface components to some tiers, or vice-versa.

Instead of a mapping, we have used cross product of tiers and user interface components as shown in Figure 1. Three tiers form a vertical layering of the general model. Each tier contains several user interface components. The components of a tier are organized in a horizontal fashion. Not all tiers contain all available user interface components, meaning only a subset of the cross product was used. The lines in the model illustrate which components are allowed to communicate between themselves.

Our three tiers correspond to the tiers of the three-tier model. There are client, server, and data store tiers. The client tier is the part of the application that runs on the machine operated by user. Usually, this tier is implemented as a browser and its various extensions, such as

Java and JavaScript interpreters, plug-ins, etc. The server tier corresponds to a so-called "middle" tier. It is the portion of a distributed application that contains the business logic and performs most of the computation. This tier is typically located on a shared machine for optimum resource usage. The data store tier is the portion of a distributed application that manages access to persistent data and its storage mechanisms, such as relational databases.

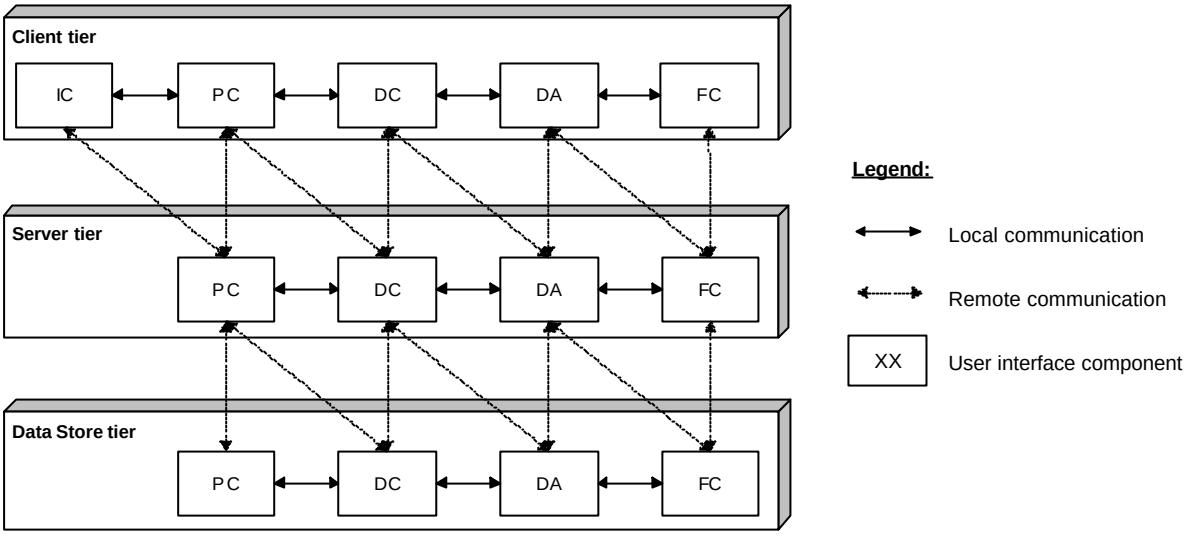

Figure 1: General architectural model

Each tier is populated by up to five user interface components. We have used the following abbreviations. For more detailed definitions see [Nigay 93].

- IC – interaction component
- PC – presentation component
- DC – dialog controller
- DA – domain adaptor
- FC – functional core

User interface components are connected by two kinds of lines. They represent the possible communication channels between components. The horizontal ones (full and thick) depict communication within a tier. The vertical lines (dotted and thin) show information transmission between tiers.

3. Features of the general model

This section highlights features of the general model. The tiers, interaction components and their combinations are discussed in detail. Some rules for use of the model are derived.

Tiers capture the physical distribution of the application. Each tier represents a part of the application that can reside on a separate device. One tier is not supposed to be split in two or more distributed parts. However, it is possible that two or more tiers run on the same device. A practical example can be a server and data store running on one computer.

Interaction components are functional blocks of the user interface implementation. Similarly to Arch, our model states that the application contains five types of such functional blocks: functional core, domain adaptor, dialog control, presentation component, and the interaction component.

Model can be customized. The general model shows all possible placements of user interface components. The model can be specialized by removing some components to

reflect either existing or desired configuration of the application. For example, the model of an application implemented as an applet has exactly one of each PC, DC, and DA components, all placed in the client tier. The server and data tiers contain only functional core. In general, the model remains functional when there is at least one path that connects functional core with the interaction component. In any case, no components are to be added to the general model.

Multiple components are allowed. Although some components can be removed during customization, there can be several copies of the same component. For example, the customized model can contain two dialog controllers; the first one is in the client tier, and the second one in the server tier.

User interface is on the client. The user interface of the whole web-based application runs on the client tier. The general model expresses this feature by the fact that the client tier is the only tier that contains the interaction component. Neither server nor data store tier can contain the interaction component.

Data are not stored on the client. The client tier cannot store the gross of the data. Although the client tier can implement part of the functional core, the client itself can store only a limited amount of data related to the user interface. This feature of the general model reflects the thin client definition [Client].

The client cannot communicate with the data store. The client and data store tiers cannot communicate directly. Any data transfer between them has to be mediated by the server tier. This provides better application security and stability.

Model differentiates between local and remote communication. There are two kinds of lines used in the model shown in Figure 1. The thin lines indicate that remote communication takes place. This implies these communication lines are slower and more delayed. They should be used as sparsely as possible, transmitting as few data as possible. In contrary, the thick lines mean local communication. In this case, no networking is involved and usually much faster and reliable communication can be expected.

4. User Interface for All

Building a web based application for all is still a non-traditional challenge. Two groups of requirements are opposed to each other. On the one hand, the application should have specialized user interfaces for people with special needs, e.g. for deaf or visually impaired people. On the other hand, the application should use standard components and standard protocols. Implementation of special requirements needs careful reasoning about which components of the system should be extended and how. This section shows how our model can assist while making these decisions.

As an example, let's take a hotel reservation system. It should be accessed via either a Windows Icon Mouse Pointer (WIMP) interface [Martin 96] or a touch-tone phone. The functionality has to be the same: to place a reservation for a given hotel for given time. The task of the developer is to decide how to split the application into parts, what the functionality of each part is, and what standard software can be used to implement some of the parts. Two customizations of the general model can be used to assist the development.

Table 1 proposes one of the many solutions, how the tiers can be populated by user interface components to achieve reasonable quality of service. The rows of the table speak about the tiers of the application. The columns relate to versions of the user interface. The description of the decisions that led to this particular assignment is out of the scope of this short paper.

Table 1 Example of tier / component assignment

	WIMP interface	Touch-tone interface
Client tier	IC, PC, DC	IC
Server tier	DA	PC, DC, DA
Data store tier	FC	FC

Both customizations can be then combined into one customized architectural model, as shown in Figure 2. The combination is done to minimize the number of components used in the actual implementation.

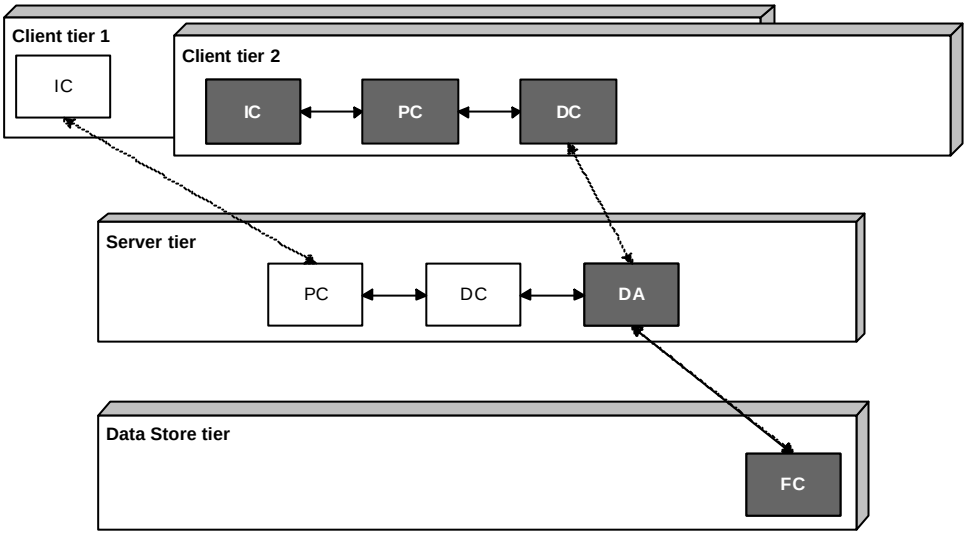

Figure 2 Architecture for WIMP / phone application

There are two versions of the client tier. Each version models architecture of different client devices. Client tier version 1 depicts the phone, containing nothing but an interaction component. Version 2 models the proposed setup for the WIMP interface.

The server and data store tiers are shared by both versions of the client tier. This provides common functionality of both kinds of the user interface. The server tier contains the domain adaptor, dialog control and presentation component. The phone interface (version 1) uses different dialog control and presentation component than the WIMP interface (version 2). This allows for additional flexibility in tailoring the structure and form of a dialog to different needs of different media.

Some interface components of the model are drawn with a dark background. They represent those components that can be implemented using standard software. Common web browsers can be used to implement version 2 of the client tier. Domain adaptor and functional core can be realized as a common web server. In this way, development costs can be minimized while providing high quality in use.

5. Summary

The development of web-based applications is a complex and error-prone process. We address this problem by introducing a general architectural model for web-based applications. The model was constructed to reflect both the physical partitioning of the application, as well as to provide a tool while reasoning about a user interface, its components, and their implementation.

The general architectural model is derived from a cross product between a multi-tier model and a user interface model. The physical partitioning was expressed using three tiers: a client, server, and data store tier. The tiers reflect physical components of the application, such as a client device, server computer, and database system. The user interface components refer to user interface creation process. Each user interface component can exist in several copies. There is one exception from this principle; the interaction component can reside only in the client tier, thus reflecting the fact, that the user interface is visible solely on the client device. The model specifies that the client tier cannot communicate directly with the data store tier. Moreover, all communication lines between various user interface components are listed. The model differentiates between local and remote communication lines.

Among other things, the model can assist to develop web-based applications with multiple user interfaces covering the same functionality. An example is presented to demonstrate use of the model in development of an application with a dual user interface.

Bibliography

[Arch 92] The UIMS Tool Developers Workshop, A Metamodel for the Runtime Architecture of an Interactive System, SIGCHI Bulletin, 24, 1 (Jan. 1992), pp. 32-37

[Buschmann 96] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. "Patterns Oriented Software Architecture: A System of Patterns", John Wiley & Son Ltd, 1996

[Client] Thin client definition, http://isp.webopedia.com/TERM/t/thin_client.html

[Dix 98] Dix, A., Finlay, J. E., Abowd, G. D., Beale, R. "Human Computer Interaction", Prentice Hall, 1998, pp. 341-373

[Musciano 98] Musciano, C., Kennedy, B. Loukides, M "HTML: The Definitive Guide", O'Reilly & Associates, 1998

[Martin 96] Martin, A., Eastman, D. "The User Interface Design Book for the Applications Programmer", John Wiley & Sons, 1996, p.116

[Nigay 93] Nigay, L., Coutaz, J. "A Design Space For Multimodal Systems: Concurrent Processing and Data Fusion", in Proc. Interchi'93 Human Factor in Computing Systems (Amsterdam, April 24-29, 1993), ACM Press, pp. 172-178

[Pattison 98] Pattison, T. "Programming Distributed Applications with COM and Microsoft Visual Basic 6.0", Microsoft Press, November 1998

[RDS 97] RDS 1.5 Documentation: Understanding Remote Data Service Applications, http://www.hexagon.net/iishelp/Msadc/docs/adcdg01_1.htm, Microsoft Corp., 1997