CHAPTER 3

Mobile Application Architectures

Where there is no vision, the people perish.

—Proverbs 29:18

In this chapter, we discuss mobile application architectures. We start by describing some of the general concepts and terms behind client-server architectures and follow this by describing clients and servers and the connectivity between them. We then present several interesting architectural patterns and describe why they are useful as general mobile application architecture solutions. Finally, we discuss some of the tenets behind good architectural design and the considerations you need to be aware of when designing mobile applications.

3.1 CLIENT-SERVER

Application architectures are often modeled to highlight or illustrate the overall layout of the software (e.g., application code and platform) and hardware (e.g., client, server, and network devices). While there are many possible combinations of software and hardware, application architectures often fall into a series of recognizable patterns.

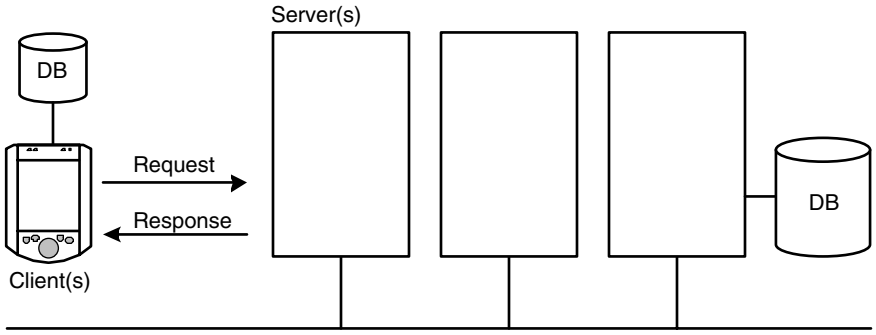
Application architectures are commonly modeled in terms of a client-server architecture wherein one or more client devices requests information from a server device. The server typically responds with the requested information (see Figure 3–1).

We can further consider client-server architectures using layers and tiers and the communication between the layers and tiers. Each of these is described in greater detail in the following sections.

3.1.1 Layers

Application code functionality is not necessarily uniform throughout an application. Certain sections of application code are better suited for handling the user interface, while other sections are developed to manage the business logic or communicate with the database or back-end systems.

Figure 3–1 Client-server architecture



Layering describes the division of labor within the application code on a single machine. Layers are often no more than code modules placed in different folders or directories on the client or server.

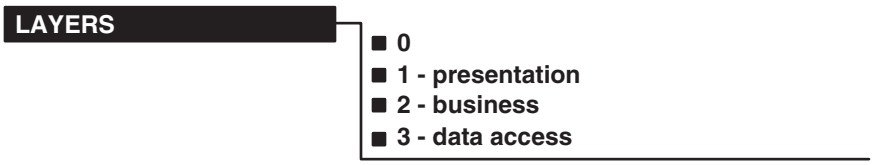
With client-side code, there are generally zero to three layers of application code. With server-side code, there are generally one to three layers of application code. This is partly a matter of good software design that helps code re-usability, partly a matter of security, and partly a matter of convenience.

A client with zero code layers essentially has no custom application code. This type of client is commonly referred to as a *thin client* and is possible in client-server architecture if the server holds all the custom application code. A client with one to three layers of application code is commonly referred to as a *fat client*.

A server can also have one to three layers of custom application code. However, you cannot have zero code layers on a server by definition.

The code layer that interacts most closely with the user is often referred to as the Presentation Layer. The second layer is often referred to as the Business Layer, as it typically handles the business logic of the code. The third layer is often referred to as the Data Access Layer. It typically handles communication with the database or data source (see Figure 3–2).

Figure 3–2 Layers



It is possible to have more than three layers on either the client or server but too many layers can become unwieldy and difficult to manage. As a result, this is not frequently encountered.

3.1.2 Tiers

While breaking up application code functionality into layers helps code re-usability, it does not automatically make the architecture scalable. In order to do so, it is important to distribute the code over multiple machines.

Tiers describes the division of labor of application code on multiple machines. Tiering generally involves placing code modules on different machines in a distributed server environment. If the application code is already in layers, this makes tiering a much simpler process.

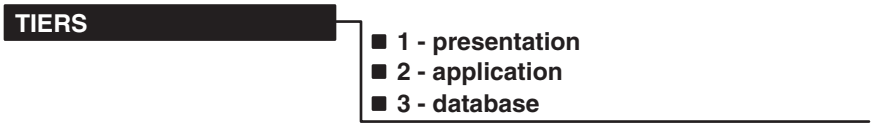
The code that interacts most closely with the user is often placed in the Presentation Tier. A second tier, which holds the application business logic and data access logic, is often referred to as the Application Tier. The third tier often houses the database or data source itself and is often referred to as the Database Tier (see Figure 3–3). This is an example of a three-tiered architecture.

The servers that make up each tier may differ both in capability and number. For example, in a large-scale distributed web application environment, there may be a large number of inexpensive web servers in the Presentation Tier, a smaller number of application servers in the Application Tier, and two expensive clustered database servers in the Database Tier. The ability to add more servers is often referred to as *horizontal scaling* or *scaling out*. The ability to add more powerful servers is often referred to as *vertical scaling* or *scaling up*. Tiering the application code in such a fashion greatly facilitates the ability to scale applications.

In large-scale distributed web applications, tiers are often bounded by firewalls. For example, a firewall may be placed in front of the Presentation Tier while a second firewall may be placed in front of the Application Tier. The Presentation Tier is thus sandwiched between firewalls in what is termed the Demilitarized Zone (DMZ), while the Application and Database Tier servers are shielded behind the second firewall in what is termed the Intranet Zone. Tiering therefore also facilitates security and allows large enterprises to shield precious internal systems from traffic originating from untrusted zones such as the Internet and DMZ. Without tiering, it becomes very difficult to secure internal systems.

Tiers generally describe server architectures, and we do not typically count client devices as a tier. While it is possible to do so, this is not a usual convention.

Figure 3–3 Tiers



3.2 CLIENT

There are many mobile device types, including RIM devices, cellular telephones, PDAs, Tablet PCs, and Laptop PCs. These mobile devices can typically operate as thin clients or fat clients, or they can be developed so that they can host web pages (see Figure 3–4). In the following sections, we describe these client types in more detail.

3.2.1 Thin Clients

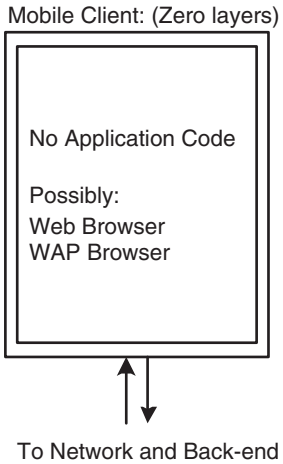
Thin clients have no custom application code and completely rely on the server for their functionality (see Figure 3–5). Thus, they do not depend as heavily on the mobile device's operating system or the mobile device type as fat clients.

Thin clients typically use widely available web and Wireless Application Protocol (WAP) browsers to display the following types of application content pages:

- Web (e.g., HTML, XML)
- WAP (e.g., WML)

Figure 3–4 Client types

Figure 3–5 Thin client–Zero layers



For example, if web pages are to be displayed, a Pocket PC can display them through Microsoft Pocket Internet Explorer, while a Tablet PC and Laptop PC can also display them through Microsoft Internet Explorer or Netscape Navigator. Similarly, a WAP browser on a cellular telephone can display WML pages.

Thin clients have several advantages over fat clients. For example, they are much easier to maintain and support since there is no application code or data on them. As a result, there is no need to consider application code release and distribution mechanisms to the client.

The problem with thin clients, however, is that they essentially must be in constant communication with the server, since that is their source for updating and obtaining data. If communications are not reliable, you may need to consider standalone fat client applications instead.

3.2.2 Fat Clients

Fat clients typically have one to three layers of application code on them and can operate independently from a server for some period of time.

Typically, fat clients are most useful in situations where communication between a client and server cannot be guaranteed. For example, a fat client application may be able to accept user input and store data in a local database until connectivity with the server is re-established and the data can be moved to the server. This allows a user to continue working even if he/she is out of contact with the server.

However, fat clients depend heavily on the operating system and mobile device type and the code can be difficult to release and distribute. You may also have to support multiple code versions over multiple devices.

Fat clients can be implemented using one, two, or three layers of application code. However, if you only use one layer it is extremely difficult to isolate the individual areas of functionality and reuse and distribute the code over multiple device types. Thus, it is generally better to use two or, preferably, three layers so that you can reuse as much of the application code as possible (see Figure 3–6, Figure 3–7, and Figure 3–8).

3.2.3 Web Page Hosting

It is also possible to display and service web pages on the mobile device even when the mobile client is only periodically connected to the network and back-end systems. In order to do so, we need the equivalent of a “mini” web server on the mobile device.

Figure 3–6 Fat client–One layer

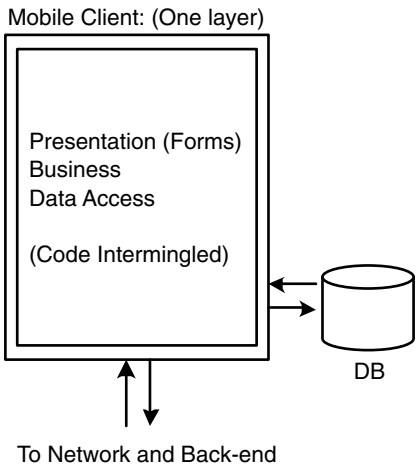


Figure 3–7 Fat client–Two layers

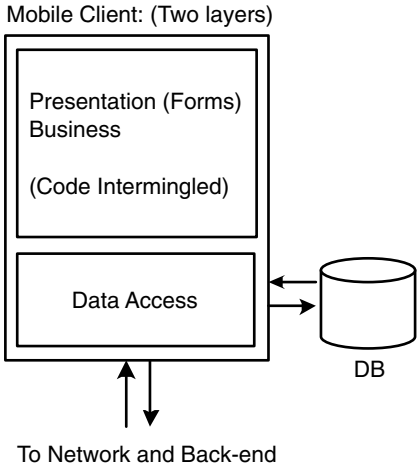
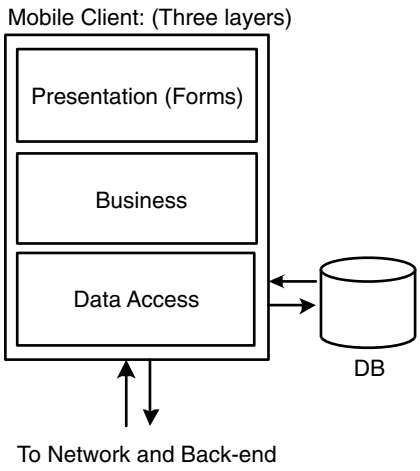


Figure 3–8 Fat client–Three layers



Microsoft has released an HTTP server that runs on a Pocket PC for just such a purpose. Data entered by a user on a web page is serviced by the HTTP server and stored in a local database until it can be uploaded to a server when connectivity has been restored.

Clients that utilize web page hosting can also have one to three layers (see Figure 3–9, Figure 3–10, and Figure 3–11). The main difference between web page hosting and the

Figure 3–9 Web page hosting–One layer

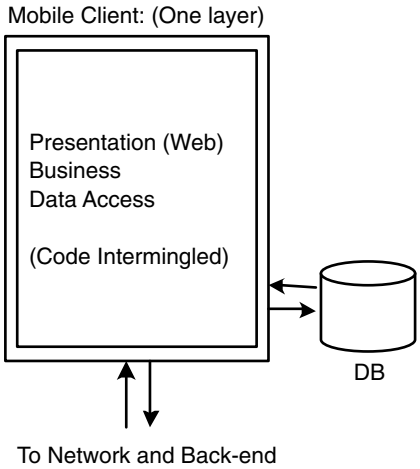


Figure 3–10 Web page hosting–Two layers

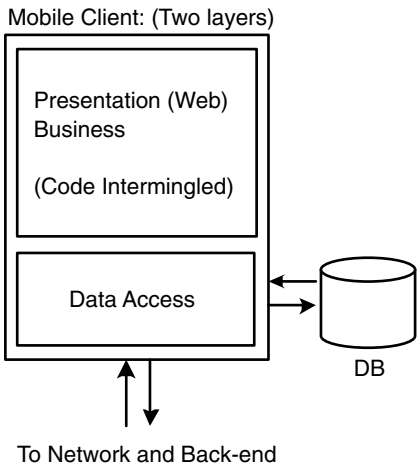
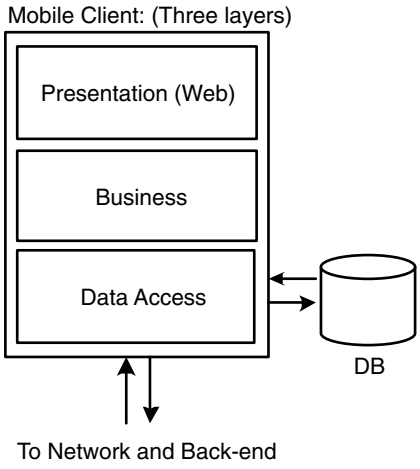


Figure 3–11 Web page hosting–Three layers



Windows Forms fat client is that the Presentation Layer displays and utilizes web pages instead of Windows Forms.

3.3 SERVER

Server architectures are commonly composed of one to three code layers implemented in one to three tiers. While the temptation is to always build three-tier architectures, there are pros and cons to doing so. For example, large-scale three-tier architectures can be expensive to implement. If the actual application is for a limited number of people, it can be overkill.

In the following sections, we will discuss one-tier, two-tier, and three-tier architectures and some of the pros and cons of implementing them.

3.3.1 One-Tier Architecture

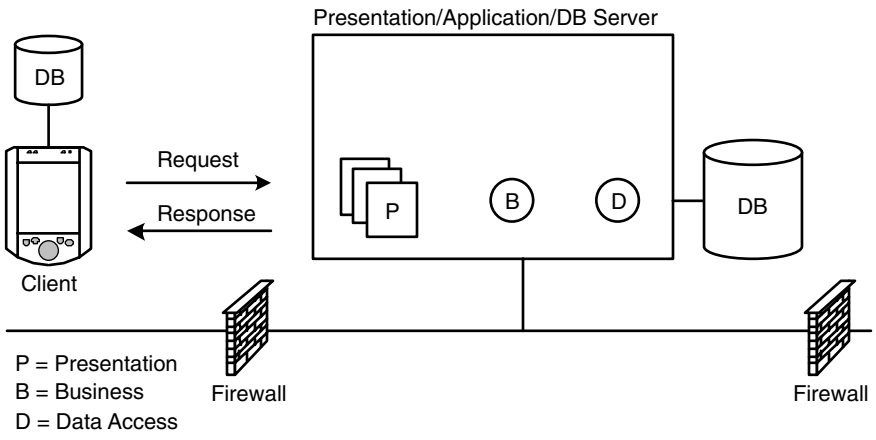
A one-tier architecture can be developed so that three code layers exist on a single server (see Figure 3–12). There are several pros and cons for doing so, as follows:

Pros
- Very convenient
- Quick to develop and deploy

Cons
- Less scalable
- Hard to secure

It is extremely convenient to be able to develop code on a single machine. On the other hand, it is extremely difficult to scale the application. For an internet application, it is also hard to shield the server using firewalls and security zones since you almost certainly have to place the server in the DMZ, which may result in exposing your database to an unacceptably high security risk.

Figure 3–12 One-tier architecture



3.3.2 Two-Tier Architecture

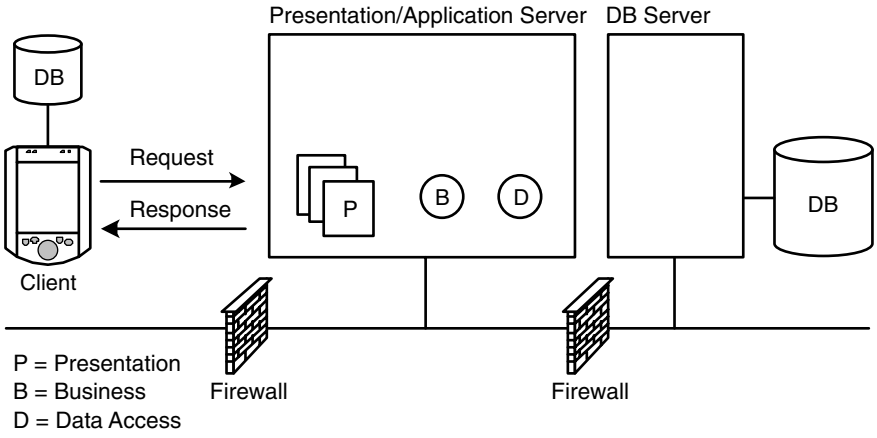
A two-tier architecture can be developed so that a database server is split off from the presentation/application server (see Figure 3–13). There are several pros and cons for doing so, as follows:

Pros
- Convenient
- Allows database server specialization

Cons
- Less scalable
- Hard to secure
- More expensive

Splitting off the database server allows it to become a more specialized server. However, it is still extremely difficult to scale the application. It is also still difficult to shield the servers with firewalls and security zones, although this is markedly better than in a one-tier architecture. Nonetheless, you may still expose your application to unacceptably high security risks.

Figure 3–13 Two-tier architecture



3.3.3 Three-Tier Architecture

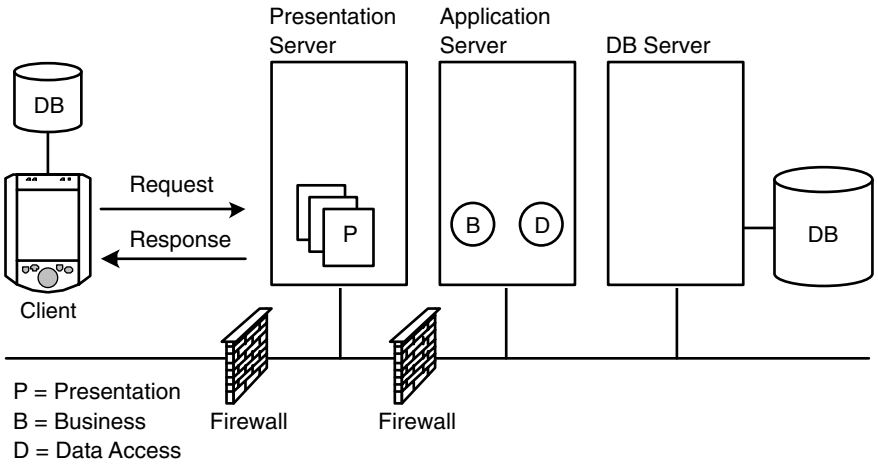
A three-tier architecture can be developed so that the database, application, and presentation servers are split off from one another (see Figure 3–14). There are several pros and cons for doing so, as follows:

Pros
- Scalable
- Secured behind firewalls and zones
- Allows database server specialization

Cons
- Overkill
- More difficult to develop
- More difficult to manage
- More expensive

Splitting off the database allows the database server to become a more specialized server. Splitting off the presentation and application servers also allows for specialization of those servers. This allows you enormous scalability potential. It is also possible to secure the application and servers behind firewalls and zones since the presentation servers can be placed in the DMZ and application and database servers can be placed in the Intranet Zone.

Figure 3–14 Three-tier architecture



3.4 CONNECTION TYPES

As discussed in Chapter 1, mobile devices typically operate in one of three modes: always connected, partially connected, and never connected (see Figure 3–15). These modes are described in more detail in the sections that follow.

3.4.1 Always Connected

A mobile device, such as a cellular telephone or RIM device, normally operates in an always connected mode. In fact, RIM coined the phrase "always on, always connected."

An enterprise might have a wireless network and set of applications and servers that allow employees to connect and use their mobile devices while on company premises. Mobile devices, such as PDAs, Tablet PCs, and Laptop PCs, then essentially become extensions of the existing applications and infrastructure, permitting users the ability to always be connected to the applications while freely moving about the office.

3.4.2 Partially Connected

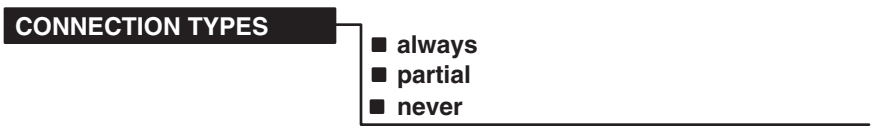
While the vision has been that mobile devices should always be connected, there are many scenarios where the mobile device is actually out of contact for extended periods of time.

Ironically, to a mobile application developer, this is where things are most interesting. For example, a mobile office worker might periodically connect to a server at the office to obtain email, contact information, or tasks to be done. The worker then disconnects the mobile device and carries out his/her normal tasks away from the office, during which time he/she might refer to the downloaded information. The user might also update the information locally on his/her mobile device before reconnecting at a later time to resynchronize the mobile device with the server.

3.4.3 Never Connected

There are also several mobile devices that never connect to back-end systems, such as certain gaming devices. While we have included this section for completeness, we will not be discussing these devices in this book.

Figure 3–15 Connection types



3.5 SYNCHRONIZATION

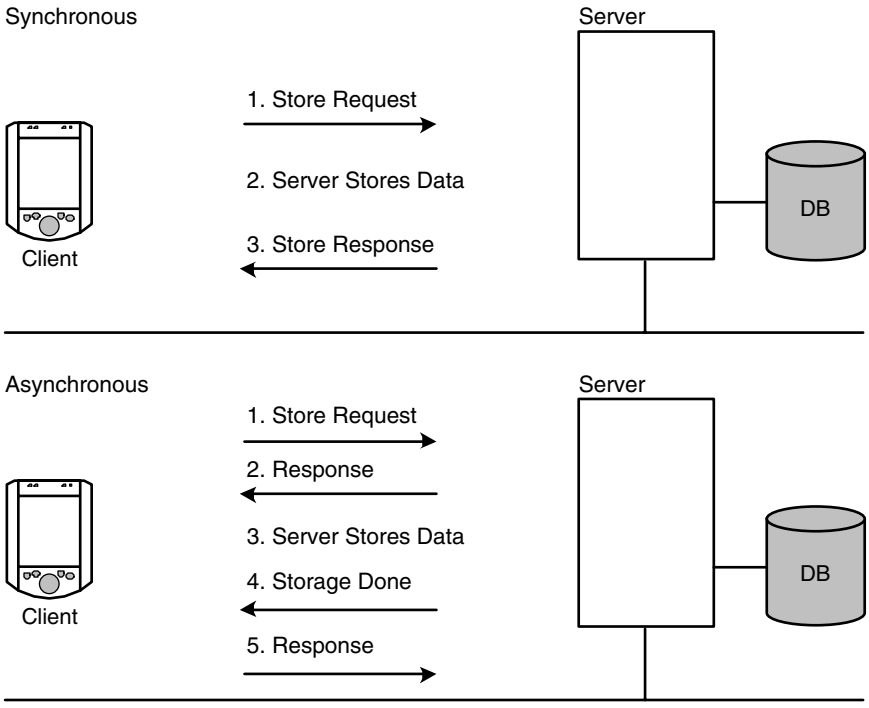
The connection type affects the way in which you can synchronize data between the mobile device and back-end systems. Synchronization is possible in two ways: continuously or through a store-and-forward method (see Figure 3–16). In the following sections, we will discuss these methods in more detail.

3.5.1 Continuous Communication

When the connectivity between the client and server is continuous, the synchronization of data between client and server is continuous and can be achieved through synchronous or asynchronous means (see Figure 3–17).

Figure 3–16 Synchronization methods

Figure 3–17 Synchronous and asynchronous communication



Synchronous communication occurs when a request to store data is sent to the server followed by the data to be stored. The data is then placed in a storage area, such as a database, on the server. In synchronous communication, all data is completely stored before the server acknowledges receipt of the data and frees up the client user interface.

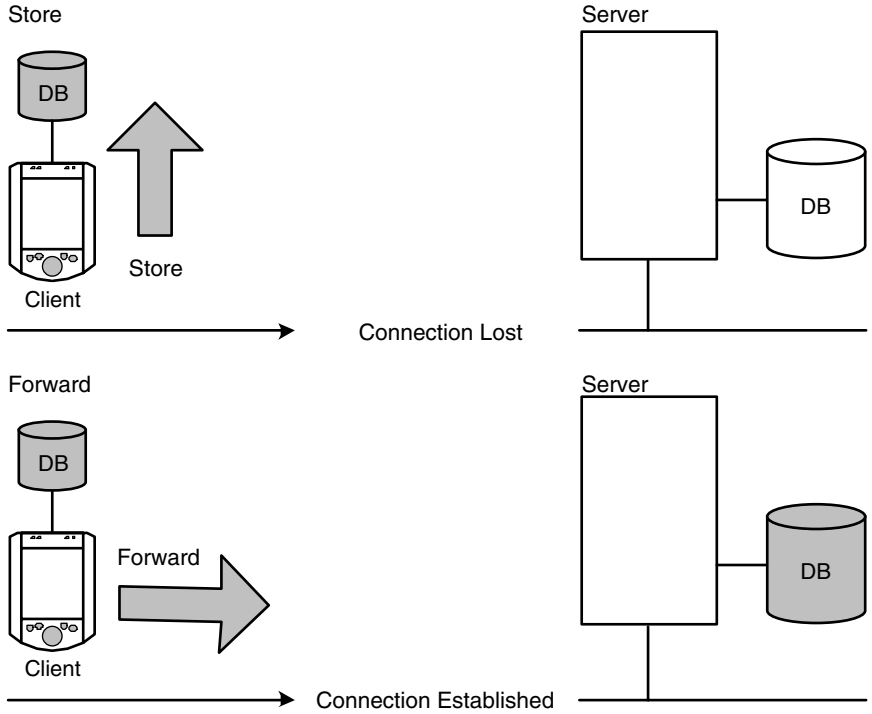
Asynchronous communication occurs when a request to store data is sent to the server followed by the data to be stored. The data is then placed in a storage area, such as a database, on the server. In asynchronous communication, however, the data does not have to be completely stored before the server acknowledges the client. Indeed, the server typically acknowledges the request immediately and only subsequently carries out the store request. Subsequently, when the store request is actually complete, the server will initiate a conversation to tell the client it is done.

3.5.2 Store-and-Forward Synchronization

When connectivity between a client and server cannot be guaranteed, it is still possible to store and transmit information safely using a method called "store-and-forward."

Suppose, for example, that a mobile user wishes to enter data while his/her mobile device is not connected to a server. A mobile client application can initially store the data in a local data-

Figure 3–18 Store and forward



base. Later, when a connection has been established, the mobile application will forward the data from the local database to the database on the server (see Figure 3–18).

Store-and-forward is a powerful method that allows mobile users the ability to work even when they are not connected to a server. It is important to note, however, that if you permit mobile users to store data in a local database in this manner, you must also ensure data integrity when the data is synchronized with the server database, since other users may be adding or modifying possibly conflicting data on their mobile devices.

3.6 INTERESTING ARCHITECTURAL PATTERNS

In the following sections, we describe some of the patterns of small-, medium-, and large-scale mobile application architectures.

3.6.1 Pattern Matrix

If we assume there are four possible client layers, three possible server tiers, and three connectivity types, there are 36 possible combinations in total. This can be represented in tabular form (see Table 3–1).

However, not all of these combinations are particularly useful or viable. For example, if a mobile client device is never connected to a back-end server, this is not a useful architecture in our case. Thus, 12 of the 36 combinations are not viable, leaving just 24 patterns.

Currently, the "partially connected" patterns are probably the most prevalent since connectivity cannot always be guaranteed. In the future, the "always connected" patterns can be expected to be much more prevalent.

3.6.2 Zero-Layer, Three-Tier, Always Connected Architecture

In Figure 3–19, we present a simple mobile architecture. The mobile client has zero application code layers on it, which means it is a thin client. The server holds all the application code and it is organized in a three-tier architecture.

The Presentation Tier has application code that is able to render pages to a mobile device such as a Pocket PC. The pages are ordinary web pages (e.g., ASP.NET, JSP, HTML) and are viewable through the use of a web browser such as Microsoft Pocket Internet Explorer.

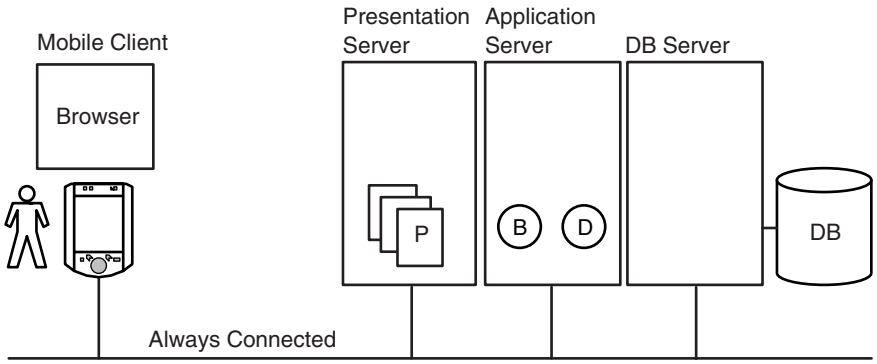
The Presentation Tier also communicates with business and data access objects on the Application and Database Tiers. Typically, data may be read from the database and written back to it during an update.

This architecture is very simple because the mobile client is assumed to always be connected to the server. Thus, there is no provision for storing application data on the mobile device. If the mobile device becomes disconnected, it will not be able to obtain up-to-date information until the connection is re-established.

Table 3–1 Pattern Matrix

Client	Server	Connectivity	Comments
0	1	Always	
0	1	Partial	
0	1	Never	Not Viable
0	2	Always	
0	2	Partial	
0	2	Never	Not Viable
0	3	Always	
0	3	Partial	
0	3	Never	Not Viable
1	1	Always	
1	1	Partial	
1	1	Never	Not Viable
1	2	Always	
1	2	Partial	
1	2	Never	Not Viable
1	3	Always	
1	3	Partial	
1	3	Never	Not Viable
2	1	Always	
2	1	Partial	
2	1	Never	Not Viable
2	2	Always	
2	2	Partial	
2	2	Never	Not Viable
2	3	Always	
2	3	Partial	
2	3	Never	Not Viable
3	1	Always	
3	1	Partial	
3	1	Never	Not Viable
3	2	Always	
3	2	Partial	
3	2	Never	Not Viable
3	3	Always	
3	3	Partial	
3	3	Never	Not Viable

Figure 3–19 Zero-layer, three-tier, always connected architecture

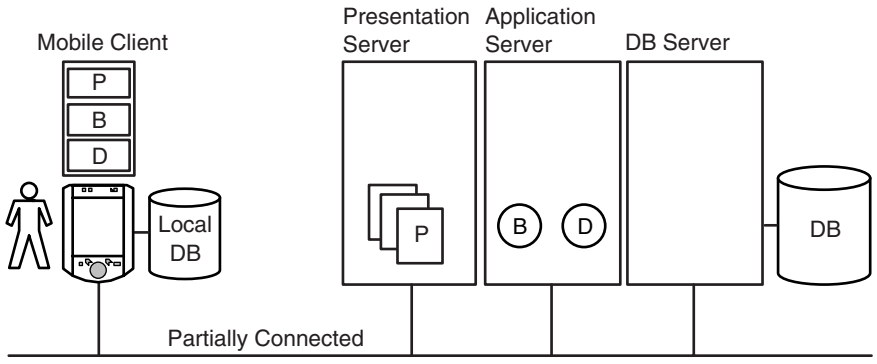


3.6.3 Three-Layer, Three-Tier, Partially Connected Architecture

In Figure 3–20, we present a more complex mobile architecture. The mobile client has three application code layers, which means it is a fat client. The server also holds application code and is organized in a three-tier architecture.

The mobile client has a complete standalone application that is able to read and write user-entered data to a local database during periods when it is not connected to the server. When connectivity has been re-established, the data can be retrieved from the local database and uploaded to the server using a store-and-forward mechanism.

Figure 3–20 Three layer, three-tier, partially connected architecture



3.7 GOOD ARCHITECTURAL DESIGN TENETS

In the following sections, we describe some of the tenets of good architectural design. In practice, it may not be possible to achieve all of the tenets. Nonetheless, you will find that many of the best mobile application architectures meet many of these tenets.

3.7.1 Requirements

The architectural design must address the business, functional, and user requirements. Without conforming to the requirements, you will not be able to satisfy financial, functional, or usability success criteria.

3.7.2 Technology Independence

In an ideal world, you should develop mobile applications that are as device and platform-independent as possible. This is not always easy or possible, but good applications tend to be written so that they can run on many devices and platforms.

In practice, however, most applications will probably fall short of these paradigms. In all likelihood, you will need to select a preferred device and platform and write your application accordingly. Thus, you will almost certainly have to choose a mobile device such as the HP iPAQ, which uses Microsoft Windows Mobile 2003, or an HP Tablet PC running Microsoft Windows XP Tablet Edition. Alternatively, you may choose a Palm Pilot running Palm OS. Each device and platform has different characteristics that your application must take into account.

3.7.3 High Performance and Availability

The architecture must typically have excellent performance during normal and spike periods in resource demand. For example, for an e-business brokerage trading site, this might be on a heavy stock trading day. If people can be expected to use the site at any given time, the architecture must also be highly available.

3.7.4 Scalability

The architecture must be scalable to accommodate possibly large increases in the number of users, applications, and functionality. The architecture must typically be designed to easily allow for both horizontal (adding more servers) and vertical (adding faster servers) scaling without adversely affecting any existing applications.

3.7.5 User System Requirements

The architecture should typically handle the widest range and number of users possible. For example, a web application with large graphics to be displayed on a Pocket PC may look beautiful, but if users only possess low-speed modem lines, the performance may not be satisfactory. Thus, the full range of users must be kept in mind, including those with high and low performance systems.

3.8 SUMMARY

Mobile application architectures can be modeled conveniently in terms of client-server architectures. Clients can be always connected, partially connected, or never connected to a server. The code on a client or server can be layered. A client can have zero to three layers while a server has one to three layers.

Mobile client devices typically can contain thin clients or fat clients, or they can be developed so that they can host web pages. The server architecture can have one to three tiers. There are pros and cons associated with developing architectures with different tiers.

If the client is always connected to the server, it is possible to build a thin client and server architecture with no code on the client. If the client is partially connected, a fat client may be needed. Alternatively, a hosted set of web pages may be employed. In any case, good architectural design tenets, such as availability and scalability, should be followed.