Intel® Mobile Application Architecture Guide

Supports the Intel® Mobilized Software Initiative
Notice:

THIS DOCUMENT AND RELATED MATERIALS AND INFORMATION ARE PROVIDED “AS IS” WITH NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION, OR SAMPLE. INTEL ASSUMES NO RESPONSIBILITY FOR ANY ERRORS CONTAINED IN THIS DOCUMENT AND HAS NO LIABILITIES OR OBLIGATIONS FOR ANY DAMAGES ARISING FROM OR IN CONNECTION WITH THE USE OF THIS DOCUMENT.

Intel Corporation may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights that relate to the presented subject matter. The furnishing of documents and other materials and information does not provide any license, express or implied, by estoppel or otherwise, to any such patents, trademarks, copyrights, or other intellectual property rights.

This document contains information, which is the proprietary property of Intel Corporation. This document is received in confidence and its contents cannot be disclosed or copied without the prior written consent of Intel Corporation.

Nothing in this document constitutes a guaranty, warranty, or license, express or implied. Intel disclaims all liability for all such guaranties, warranties, and licenses, including but not limited to: Fitness for a particular purpose; merchantability; not infringement of intellectual property or other rights of any third party or of Intel; indemnity; and all others. The reader is advised that third parties can have intellectual property rights that can be relevant to this document and the technologies discussed herein, and is advised to seek the advice of competent legal counsel, without obligation of Intel.

Intel retains the right to make changes to this document at any time, without notice. Intel makes no warranty for the use of this document and assumes no responsibility for any errors that can appear in the document nor does it make a commitment to update the information contained herein.

Intel® products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.
What is the problem?

Computer users are increasingly embracing notebook PCs, tablets, PDAs and smart phones. These users expect the business applications they access to function as well on the road as they do in the office. However, the reliable, continuous network connections enjoyed by stationary office workers are not always available to mobile users. Companies are recognizing the potential for mobilization to increase employee productivity. Critical to the realization of this potential, however, are applications that are optimized for use on mobile devices and wireless networks.

Why is it important?

As Wi-Fi infrastructures become a standardized commodity and users increasingly turn to wireless platforms, software developers are facing rising user dissatisfaction with applications that were not designed to work in a wireless environment. Intel calls this phenomenon the mobility inflection point – a catalyst that is driving software developers to embrace mobilized software solutions, such as offline data management and synchronization and publish-subscribe services.

What is the solution?

To efficiently support users – both mobile and stationary – applications must be enhanced or even completely re-architected. The resources included in this folder will help you optimize your applications for mobile use, providing a seamless, consistent experience to your users, whether connected or not.
Contents

Introduction 5

Mobile Application Architecture 5
   Requirements for a Mobile Application Architecture 5
   Capabilities Needed to Support Mobility 6
   Mobility Adaptation Models 7
   Adapting an Application to a Mobile Environment 8

Portal Architecture for Mobile Applications 8
   Portal Architecture Overview 9
   Portal Architecture for Mobile Applications 9
      Issues and Challenges 10
      Addressing the Challenges 10
      Implementation Strategy 12
      Modifications to the Portal Server 12
      Modifications to the Client 12
      Summary 12

Document Sharing Architecture for Mobile Applications 13
   Document Sharing Architecture Overview 13
   Document Sharing Architecture for Mobile Applications 14
      Issues and Challenges 14
      Addressing the Challenges 14

Implementation Strategy 15
   Client Components 15
   Server Components 16
   Implementation Issues 17
   Reintegration and Document Structure 17
   Content Adapter: Client vs. Server 17
   Directory Tree Synchronization: Complete vs. Incremental 18
   Server Design: Stateless vs. Stateful 18
   Summary 18

Database Architecture for Mobile Applications 19
   Database Architecture Overview 19
   Database Architecture for Mobile Applications 19
      Issues and Challenges 19
      Addressing the Challenges 19
      Implementation Strategy 20
      Architecture Overview 20
      Client Components 21
      Server Components 22
      Limitations of the Solution 22
   Summary 22

Conclusion 23

References 23
Introduction

Wireless computing devices are proliferating at a rapid pace, requiring that software applications be designed to handle the realities of a mobile computing lifestyle. Users expect the same functionality from applications running on their mobile devices whether they are offline or connected to a network. They expect applications to deal gracefully with intermittent network connections and bandwidth changes and to efficiently accommodate roaming. And they expect power usage to be managed to maximize battery life without degrading performance.

These expectations present a new set of challenges to application developers. This architecture guide is intended to help application developers understand the issues that arise when mobile computers are introduced. It presents implementation strategies to address issues in the areas of portals, document sharing and databases.

Mobile Application Architecture

An architecture designed for mobile applications incorporates the use of portable computing devices and wireless networks into a computing environment in which a user can keep working productively regardless of the status of the network connection.

Portable computing devices range from devices dedicated to specific tasks, such as cell phones and hand-held personal digital assistants, to general-purpose computing devices, such as laptops and tablets. Each of these devices presents a different set of challenges to designing applications for use in a mobile environment. Some of the challenges shared by most portable computing devices include:

- The device physical location and configuration may change unpredictably as the device is connected to or disconnected from the network or moves between network connection points. A mobile application architecture must support consistent operation online and offline and provide seamless connectivity as the device moves between connection points.
- Battery powered devices can, by their very nature, operate only for a limited time without recharging or replacing batteries. A mobile application architecture must be designed to closely manage the limited power available to battery-powered devices, by using strategies that prolong useful battery life by reducing power consumption without sacrificing system performance.

Small dedicated devices may have additional limitations such as a small display or limited storage and processing power. A mobile application architecture must provide support for a wide range of portable devices.

Wireless networks using Wi-Fi technology operate at local area network speeds; they allow users to connect without a physical connection; and they can eliminate public network access charges, overcoming some of the limitations of connecting to the Internet through a modem and phone line. However, wireless networks are susceptible to external interference and attenuation, potentially affecting reliability, decreasing effective bandwidth and presenting challenges to effective security.

The use of portable devices and wireless networks allows a user to move between locations and to access a network from a variety of computing devices. The result is that the environment in which the user operates is no longer static, as mobile users elect to work online or offline or roam from one wireless connection point to another.

To effectively support the mobile user, application architectures must be developed that address the issues presented by the use of portable devices and wireless communication in a mobile environment, such as the changing availability of data and resources.

Requirements for a Mobile Application Architecture

To provide the same capabilities as a non-mobile version of the application, an application designed for mobility must support the following:

- **Consistent operation online or offline** – In many application architectures, data is stored on a shared network-accessible system in the form of documents, data records or binary files, with coordinated user access to a single copy of the data. An application designed for mobility should enable users to access data, whether online or offline. When working offline, the user still perceives that the shared data is available for reading and writing. When network connectivity is restored, local data changes are integrated into the network copy of the data and vice versa.
- **Seamless connectivity** – An application designed for mobility should work with an agent or proxy service to transparently handle changes in connectivity.
Connectivity should not be required for functionality, and unexpected, intermittent breaks in connection to a network should be handled gracefully. In addition to online and offline transitions, the agent or proxy service should be able to select on optimal network type from those available, such as an 802.11b wireless local area network or a General Packet Radio Services (GPRS) communication service and handle communication tasks such as security authentication or authorization and logical addressing.

- **Multi-platform support for clients** – An application designed for mobility should, at a minimum, tailor its interaction and behavior to the characteristics of the current device, such as input or output type, resources available and performance level.

- **Optimized power use and performance** – An application designed for mobility should closely manage its use of the limited power of a portable device when battery operated. For example, constant synchronization from a hard disk will quickly drain a battery, while synchronization from memory is much more power efficient. Maintaining an active radio search for the next hot spot can also drain power excessively.

### Capabilities Needed to Support Mobility

A number of capabilities are required to support an environment in which computing devices are mobile. Some of these capabilities are specific to an environment designed for mobility while others may already be available, with or without modification.

Capabilities that may need to be developed to enable an mobile environment include:

- **Resource management** – A resource, such as power or bandwidth or storage space, can be consumed and exists in a finite amount. Resource management should allow monitoring of attributes, such as amount or rate of consumption, and support notifications based on user-defined thresholds.

- **Context management** – The context of an application is the data and state that represent current conditions of the external environment, such as location, proximity and device being used. Context management should allow monitoring of attributes, such as current location or device type, and provide notification of a change in context.

- **Encoding** – Encoding involves the modification of data and protocol streams to meet the requirements of the current context and available resources. Examples of encoding are transcoding, privacy (encryption) and compression.

Encoding is either symmetric, where data is encoded by one component and decoded by another component, or asymmetric, where the data is encoded once. An example of symmetric encoding is compression, where the data is compressed on a system before transmission and then uncompressed on the system to which it is delivered. An example of asymmetrical encoding is transcoding, such as when Hypertext Markup Language (HTML) is converted into Wireless Markup Language (WML) to be received and rendered unmodified by a Wireless Access Protocol (WAP) browser.

An implementation of an encoding capability should allow for the enumeration of encoders and decoders. It should make available information that can be used to indicate when and how an encoder should be used, such as prior to a network transmission to handle a low bandwidth condition. It should also provide a mechanism for interposing encoders and decoders into the protocol pipeline. Lastly, in conjunction with the context management capability, it should provide the ability to negotiate the use of encoding.

- **View consistency** – View consistency provides a consistent, reconcilable view of data and state that is shared among intermittently connected systems using emulation, pre-fetch, and reconciliation. *Emulation* allows the user to work without being concerned about whether the application is working with local or remote data or whether the data is a replicate copy with pending updates or not. *Pre-fetch* manages the caching of data. Reconciliation ensures that offline modifications to shared data are correctly applied to all copies.

Some capabilities are required but are not specific to a mobile environment, so they may already be available without requiring modifications. These capabilities include:

- **Durable storage** – A durable storage capability allows for persistent, writeable, durable storage of structured and unstructured data. Some capabilities use durable storage to maintain configuration and static data, while others, such as reliable messaging and view consistency, use it for data storage.
Reliable messaging – A reliable messaging capability provides the ability to define and control the semantics of message delivery, such as whether the type of deliver is asynchronous, out of order or once and only. An implementation of this capability should provide guaranteed and reliable transmission and receipt of arbitrary messages. It should be independent of message source and target, message format, and underlying transports and networks. It is important that the reliable messaging provider use an interoperable format and protocol for exchanging messages.

Capabilities that are not specific to a mobile environment but may need to be extended to support a mobile environment include:

- **Policy** – Policy provides a common location to collect, correlate and react or adapt to information and events provided by the other capabilities. It receives events from the other capabilities and evaluates them against a set of stored rules. If an event tests true, an action associated with the condition is invoked. These sets of events, conditions and actions are commonly known as event-condition-action (ECA) rules. The policy implementation should provide the ability to define and store ECA rule-based policies, and be able to interact with other capabilities, both when receiving events and invoking methods.

  For example, a condition, such as low bandwidth, could have the action “use compression” associated with it. When an event is received indicating a change in bandwidth, Policy would evaluate the condition, and if true, use the encoding capability to compress data where possible. A common point of policy definition, enforcement and response simplifies the implementation of capabilities and decouples them from one other.

- **Security** – To avoid serious consequences from malicious attackers, poorly designed applications, and inadvertent user errors, the following security measures need to be taken: Systems and users should be authenticated; authenticated systems, users and actions should be authorized; and actions and interactions should be audited.

  Many of these capabilities will require interactions between systems, usually over a network. For example, when determining what encoding to use, both systems involved will need to exchange messages.

## Mobility Adaptation Models

A common task in software architecture and design is functional decomposition and allocation to determine where a given function should be performed and by what software component. For example, in a client/server database application, data validation could be enforced at the client by the application, in the database by Structured Query Language (SQL) code, or in both locations. Similar issues related to decomposition and allocation arise when a mobile application architecture is implemented.

An adaptation model describes how an application and the underlying platform allocate responsibilities for dealing with issues related to mobility. Some mobility adaptation models are described below:

- **Laissez-faire application** – With this model, each application is totally responsible for proving support for mobility. It must deal with connectivity, synchronization and platform issues with no or little support from other platform components or the operating system.

- **Application-aware collaboration** – With this model, the platform provides common services that applications can leverage to support mobility. Services range from information and notifications, such as battery lifetime remaining, to functionally transparent subsystems with automatic network state detection and connection management. In more advanced versions, applications are aware of each other and cooperate in their use of mobility-enabling resources and capabilities and policy-based control.

- **Application-transparent system** – With this model, the platform insulates the application from issues related to mobility. The application runs unmodified and is not affected by external changes. While desirable, this model is attainable only for the simplest of applications.
Adapting an Application to a Mobile Environment

Of the mobility adaptation models described in the previous section, the application-aware collaboration model appears to be the most practical. Figure 1 shows the logical architecture of this model. In this model, the mobility adaptation layer is common to all the applications, and the capabilities implemented in this layer are shared by all applications. Each application includes an application-awareness component to mediate its interaction with this layer.

Although a commercially available mobility adaptation layer component does not yet exist, product vendors are developing implementations of capabilities to ensure their software or hardware components support mobility, and platform vendors are building support for these capabilities into the platform. Applications need to be designed to take advantage of these capabilities to ensure efficient, well-functioning operation in a mobile environment. The goal of the developer is to enable users to use an application while not connected or connected under less than optimal network conditions, and to enable the application to operate efficiently in a mobile environment.

Three application areas of particular significance to users in a mobile environment, due to their prevalent use, are portals, document sharing and databases. The following sections describe each of these application areas and their significance in a mobile environment.

Portal Architecture for Mobile Applications

The primary function of a portal is to aggregate and integrate diverse and distributed sources of information and present the result to a user in a simple, consistent and pertinent view through a Web browser interface.

A portal is typically targeted to a specific group or type of user. For example, an internal employee portal may be targeted to all employees of a company, while a finance portal may be accessible only by employees authorized by the finance department. An externally-facing supplier portal might be used by an employee of a supplier to check on an invoice payment.

A portal generated page is assembled under the control of a user-defined template on the server and may contain several types of content:

- Relatively static data, such as banners, graphics, and general structure and content
- Dynamic content, such as stock quotes or current shipment orders on hold that have been retrieved from the internal and external fulfillment systems
- New or transient information, such as notifications, and incremental information

A user may be authorized to update content on the portal. For example, a user of a sales portal may be able to update the status of an account after a visit or add a link to a trip report or meeting minutes.
Portal Architecture Overview

A portal architecture comprises three types of functions:

- **Information sources** – Information sources provide data to the portal, either on request using a “push” mechanism provided by “publish and subscribe” middleware, or through a periodic extract, transform and load (ETL) process. Information sources may include databases, applications, other portals or external systems.

- **Portal functions** – Portal functions provide for aggregation and composition of the information and delivery to the user. The components of a portal are described below.

- **Independent functions** – Independent functions are pre-existing technologies or components, such as a Web browser or protocol proxy. They are generic functions, such as rendering of content in HTML for display on a Web browser or traversing of networks and firewalls using a protocol proxy.

Components of a portal include:

- **Web browser** – Provides a user interface to the portal. If access is over the Internet, a protocol proxy supports communication between the user and the portal using Hypertext Transfer Protocol (HTTP) and HTML, often enhanced through the use of client-side scripting and/or browser-hosted code, such as ActiveX* controls or Java.*

- **Presentation server** – Creates an integrated view of content through interactions with other components.

- **Application server** – Executes any custom code required to dynamically retrieve and reformat information from a non-Web-based system. May also host Aggregation and Integration components (see below).

- **Aggregation and integration components** – Provide dynamic retrieval and reformating of information for standard products such as enterprise application integration (EAI) applications or relational database management systems (RDBMS).

- **Content management, searching and indexing, and collaboration** – Manage the lifecycle of the content including modifications by users, such as creation, versioning, revisions and comments.

- **Personalization service** – Enables each user to customize the layout and content they see when they access the portal.

- **Security** – A requirement for all architectures for mobile applications to ensure the integrity of sensitive information in remote locations.

Today the protocols and interfaces between these components are proprietary to a product, but standards organizations and vendors are actively working to develop a set of XML-based standards. Most notable are the Web Services for Interactive Applications (WSIA) and Web Service for Remote Portals (WSRP) standards under development by the Organization for the Advancement of Structured Information Standards (OASIS).

Although commercially available portal products can be quite capable and complete, some custom implementation by a developer is usually required, such as data mapping, user interface customization or business logic. So, in addition to the portal product software, a portal application consists of external data sources, external data, custom application code, custom information integration/aggregation maps, and global and user rendering templates.

Portal Architecture for Mobile Applications

Due to its server-centric architecture and role in information integration and aggregation, a portal is dependent on network connectivity. A simple solution can sometimes be used to enable offline Web sites, databases and file stores. In this solution, a copy of the portal is hosted on the mobile system. The user interacts with the local portal, code and data. Whenever a network connection becomes available, differences between the local and remote copies are synchronized.

This solution is only feasible, however, for the simplest of portals. If the source data is associated with a large line-of-business system, it may not fit on a mobile system or be able to be run in a distributed manner. Portals themselves are often designed for multi-user, dynamic online interactions and can be large and complex, so it may not be feasible to host a copy on a portable device with limited resources. This solution also presents challenges for operations and licensing, which are beyond the scope of this document.
Issues and Challenges

Without a network connection, the creation of dynamic content from the current state of the portal and back-end systems at runtime is essentially impossible. However, two approaches can be used to provide offline viewing of content:

- Caching of portal-generated content
- Replication on the mobile system of the data and code used to generate the content on the portal and back-end system

The appropriate strategy to use depends on factors such as the amount of data involved, the complexity of user interaction with the data and the frequency of updates. For example, when historical quarterly sales reports are to be viewed, caching would be preferred over replicating the complete sales database from the back-end system. Given a page where the user can look up customers, pending orders, and commitment dates in an ad hoc manner, it would be preferable to replicate the data, code and templates, instead of caching all the possible pages.

Addressing the Challenges

Multi-platform support for clients – Portals usually support access from different platforms and handle device characterization and any required transcoding of the content. As content is often dynamic and the client device type unpredictable, these activities must occur at runtime as the request is being processed.

A client application that supports mobility does not need to support dynamic transcoding because the client device type is static. The application also does not need to handle dynamically changing personalization as the offline device can be expected to be used by a single person.

Ability to work offline – The mechanisms described below enable working offline in a mobile environment:

- Content caching involves offline caching of the content provided by the portal in response to a request for a Uniform Resource Identifier (URI), such as a Web page address. The code that generated the content is not cached. For example, a hyperlink may refer to a Java Server Page (JSP) or Active Server Page (ASP). When the link is clicked, the application server runs the script that generates the content of the page, which streams HTML back to the client. It is the HTML that is cached, not the JSP or ASP.

Navigating a portal, following every hyperlink and storing the output to the local system for offline viewing is resource intensive and inefficient. Not all the pages may be required and they may contain transient information or provide ad hoc interfaces to back-end systems. Therefore, content caching should be under the control of local configuration files that specify the pages of interest to cache or provide page selection criteria.

Content caching is an implementation of the document sharing architecture described in the section Document Sharing Architecture for Mobile Applications on page 13. In the case of a portal, some of the content may be generated dynamically. Both static and dynamically-generated data can be revision controlled and bidirectional updates applied.

- Code replication allows portal content to be more dynamic. The portal can execute code, such as Java servlets, JSPs, ASPs, and server controls, in the process of serving content, interacting with the user, and collecting and manipulating data and data stores. Code is replicated from the portal to clients only. Though some replicated code involves user interface components, most is involved in the collection, manipulation and storage of data.

- Data replication on the client enables more transaction-like interactions and ad hoc queries. Data may be replicated from the portal to client, client to portal, or in both directions. Any new and updated data is exchanged and applied to the copies. If the data is writeable only at the client or only at the server, implementation can be simple; however, the implementation of schemes where multiple copies of the data can be updated independently can be very difficult. See the section Database Architecture for Mobile Applications on page 19 for more details.

- Emulation – When a URI is selected in a browser, either by entering a Uniform Resource Locator (URL) in the address box or by clicking on a hyperlink, the browser attempts to contact the server specified in the URI, and request the resource, using HTTP over Transmission Control Protocol/Internet Protocol (TCP/IP). If the browser is unable to contact the server, the browser will return an error. The view consistency capability, described in the section Capabilities Needed to Support Mobility on page 6, uses emulation to mask the non-availability of a resource, so that data and services continue to be available locally to the user.
Normally, the local Web browser attempts to contact the presentation server over the network. When the system is offline, the Web browser should be redirected to the local presentation server (residing on the client) that is serving the cached content. This can be easily implemented using the autoprox mechanism supported by most browsers. A JavaScript function can be created for the autoprox mechanism that dynamically rewrites requested URLs before a request is submitted over the network. This function could, based on network state, modify the URL to redirect the browser to a presentation server running locally. This function could also perform other actions, such as changing a file extension to correspond to cached content (mypage.html) rather than a script (mypage.jsp).

**Optimized power use and performance** – Ideally, an implementation of the capabilities required for mobility will handle all aspects of managing power and performance-related resources, but application components should be aware of and closely manage power consumption rates and levels.

**Seamless connectivity** – Two areas of concern for seamless connectivity are the management of network connectivity and the security-related user experience. For example, the user should not have to physically re-authenticate each time the system reconnects.

Seamless connectivity is supported by emulation, which provides the appearance of network resource availability. For example, when a shareable file is designated “Make Available Offline,” Microsoft Windows XP copies the file to the local system and provides a virtual share through the same name. A user can access the file offline using the same drive and path, even though the network share is not available. The emulation capability “emulates” the network device and redirects requests for the file to the local file store transparently.

Emulation does not, however, handle issues related to the availability of a network or selection of the best network to use based on context. Network connections can be considered resources, and therefore could be managed by the Mobility Adaptation layer if it were available (see *Adapting an Application to a Mobile Environment* on page 8).

---

**Figure 2.** Portal application architecture for mobile applications showing in color the components added to a generic portal architecture.
Implementation Strategy

It is important to distinguish between the functionality provided by a commercial portal product and an implementation of a portal. Although a portal product may provide an extensive range of capabilities, some amount of custom data mapping, user interface, and business logic will need to be coded by the developer. So in addition to the portal product software, a portal application consists of external data sources, external data, custom application code, custom information integration and aggregation maps, and global and user rendering templates.

The portal architecture for mobile applications shown in Figure 2 reflects several types of modifications: the addition of completely new components and technologies, enhancements to existing components, and the partitioning and distribution of certain component responsibilities. These additions are shown in color in Figure 2. Other aspects of an existing portal application not shown in the figure, such as its structure or code, will almost certainly also need to be modified.

Modifications to the Portal Server

The portal server requires the addition of two technologies: synchronization and reliable messaging.

To support synchronization, the content management, presentation, and application server portal components may need to be enhanced to perform such functions as maintaining a log of modifications and using time stamps to help with serialization. Some components, such as the content management component, already maintain this information. It may also be possible to use synchronization data from the back-end databases directly, using specialize synchronization components for data, code and content to handle the task of keeping clients and the server synchronized.

Again, a reliable messaging facility is required for the exchange of information between the server and clients.

Modifications to the Client

As with the server, reliable messaging and synchronization components are required. For offline operation, three additional components are needed: data stores for data, content and code; an application server; and a presentation server.

The use of an application server and presentation server on the client reduces the functionality that will need to be different for a mobile version and non-mobile version of the application. Two different versions will be needed only for the presentation layer. The presentation server, or Web server, ensures that the portal application Web browser can run unmodified. The application server executes code replicated from the server.

Summary

A portal integrates distributed sources of information and presents the results in a simple, consistent view through a Web browser interface. In an always-connected environment, the external information sources used by the portal to generate displayed data are continuously available. In a mobile environment several methods are used to provide access to information even when the user's system is offline. These methods include caching of portal-generated data and replication on the client of code and data used to generate portal content.
Document Sharing Architecture for Mobile Applications

One of the primary uses of a personal computer is as an electronic file cabinet that stores data in a way that allows it to be retrieved when it is needed. Normally, data is stored in a file. For a file to be useful, it must be associated with an application that comprehends both the structure and the data of the file. A database is a file whose structure and data are managed by a database management application. Other files, called documents, may contain text, spreadsheets, slides, or audio or video clips and are managed by applications such as a text editor or word processor, a spreadsheet editor, or video or audio recorder or player.

When a copy of a document has been modified and the modifications need to be reintegrated into the original document, the most common solution is to simply overwrite the entire original document with the modified version. Because this file level reintegration requires no understanding of the structure of the data in the file, the application that manages the document's contents is generally not involved. In this section, a document is distinguished from a database by the way in which data in two slightly different copies of the same file is reintegrated or synchronized. A document is defined as a file that is reintegrated in its entirety, without the intervention of the application that understands its structure.

While a document is reintegrated at the file level, a database, with the assistance of its database management application, can be reintegrated at the sub-file level. When two different versions of a database need to be reintegrated, only the section of the file containing the modified data is updated. This sub-file level reintegration is possible because the database application, which understands the structure of the database, manages the update. Since databases can be quite large, this approach to reintegration is more efficient than overwriting the entire file.

This distinction between a file and a database is somewhat arbitrary. In some situations, a document management application's understanding of document structure is required for basic mobile functionality. For example, adapting document content to fit a particular type of device cannot be accomplished without this knowledge. Also, an implementation that does not allow user-driven sub-document reintegration will not be very useful. The more documents are treated like databases, however, the more complex the solution becomes. For a more complete discussion of this topic, see the section *Reintegration and Document Structure* on page 17.

This section describes a document sharing architecture for mobile applications, while the section *Database Architecture for Mobile Applications* on page 19 discusses a database architecture for mobile applications.

Document Sharing Architecture Overview

When the contents of a document are to be made available to others, the document containing the information must be shared. Sharing of documents can be accomplished in either of the following ways:

- **Single copy document sharing** – With single copy document sharing, users share access to the original document. Multiple versions of the document are not created. Any changes to the document, including its destruction, are absolute, and the information in the document is guaranteed to be current.

  A disadvantage of this sharing method is that because only one copy of the document exists, only one person can access the document at any one time. Also, one copy of the document presents a single point of failure in this document architecture. If a user neglects to make a document available again after accessing it, other users will be unable to retrieve it. Because the risk of unavailability, and the resulting reduction in productivity, is so high, single copy document sharing is generally not acceptable.

- **Multiple copy document sharing** – With multiple copy document sharing, each user has a copy of the document.

  The disadvantage of this sharing method is the difficulty of keeping the information in the document current. If two users are both using a document and one changes the information in their copy of the document, the information in the other user's copy becomes inaccurate. If both copies of the document are changed at the same time, both copies are inaccurate and must be synchronized or re-integrated.
A document sharing architecture should accomplish the following:

- Allow information in documents to be shared
- Allow information from documents to be available on demand
- Maintain the concurrency of the information in documents

Both document sharing methods described above provide challenges to meeting these requirements. Single copy document sharing prevents access by more than one user at a time, while multiple copy document sharing poses risks that modifications to one copy of the document may not be reflected in other copies of the document. The limited access to a single copy of a document is likely to have a serious impact on productivity. However, the disadvantages of multiple copy document sharing are more easily resolved, because a document sharing architecture based on multiple copy document sharing can provide a means to synchronize modifications between multiple copies of a document.

To provide basic functionality, the document sharing architecture must contain original documents, copied documents, shared document stores, shadow document stores, and a management system that tracks document location, and propagates modifications through every copy in the file system, resolving version conflicts automatically if possible, and with user intervention if necessary.

To provide document availability while maintaining coherency between copies of the document, the document sharing architecture must have constant access to all copies of all documents under its control. In other words, document copies, document stores and the management system must be constantly connected to each other. If a connection can be guaranteed, the system can allow documents to be shared, provide continuous document availability, and maintain concurrency.

---

Document Sharing Architecture for Mobile Applications

Issues and Challenges

In a mobile environment, a constant connection between documents and the document management system can no longer be guaranteed. The document sharing architecture must be modified to allow information in documents to be shared while maintaining the concurrency of the information in these documents. It must accomplish this while taking into account the limited hardware resources, such as storage space or display area, of a typical mobile device.

Addressing the Challenges

**Ability to work offline** – To enable users to work offline, the documents to be accessed must be pre-fetched and stored in a file cache. If a mobile system’s storage resources were sufficient, all documents to be shared could be copied from the server to the mobile system. However, many mobile platforms have limited storage resources, which means that the architecture must provide the ability to pre-fetch selectively.

In addition to making shared documents available, the document sharing architecture must also be able to re-integrate modifications to the local copy into the copy on the server and all other copies on the system.

**Seamless connectivity** – The document sharing architecture should be able to free the user from concern about connectivity. It should take advantage of a connection as soon as it is available and be able to gracefully continue without error or state loss when the connection disappears. Also, functionality should not be disrupted when a power state change occurs.

For this reason, the architecture must provide reliable messaging between all components in the system. When multiple paths to the server are possible, the architecture should also be able to choose the best connection based on reliability, speed and quality of service. And finally, as the connection between client and server fluctuates, the architecture should be able to handle session-based security issues.
Multi-platform support for clients – The document sharing architecture should provide support for multiple platforms. It should be able to identify a mobile device and the resources available to the mobile device, such as network bandwidth, system architecture, transient and persistent storage, display type, and available power schemes. It should also be able to modify the content of documents in response to device characteristics.

Optimized power use and performance – The document sharing architecture must optimize the capability of devices to function on battery power for extended periods of time. If performance comes at the price of power, the architecture should allow the user to make tradeoffs between performance and power usage.

Implementation strategy

A document sharing architecture that supports mobility makes shared documents available to mobile clients. The client and server components required for such an architecture are described below, along with a discussion of several implementation issues.

Client Components

The client components of a document sharing architecture for mobile applications are shown in Figure 3 and described below:

- **Document management application** – A document management application is any application that uses documents, such as a word-processing application, MP3 player or file system browser.

- **File system agent** – The file system agent is the main component of the document sharing architecture. It handles all document requests from the document management application and determines whether documents are local or remote. It interacts with system capabilities that support mobility (described below) to communicate with and handle messages from the document sharing application on the server (see the section **Server Components** on page 16). The file system agent manages the contents of the remote file cache according to user-defined settings.

The **content adapter** is a part of the file system agent that is responsible for adapting the content of shared documents to accommodate the mobile device storage and display characteristics. How content is adapted is determined by user settings, device characteristics, and the document structure. Depending on the document structure, the content adapter may require the services of the document management application.

The **reintegration manager** is the part of the file system agent responsible for resolving document version conflicts. As far as the document structure will allow, it attempts to reintegrate versions automatically. When conflicts cannot be resolved automatically, it utilizes the document management application – and ultimately the user – to accomplish the reintegration.

- **Local file system** – Contains all local documents. It generally has a management application of its own that is usually part of the operating system.

---

Figure 3. Client components of a document sharing architecture for a mobile application.
- **Remote document cache** – Contains copies of all documents that originated on remote file systems. The remote document cache can be considered a database whose database management application is the file system agent. It may be stored on the local file system.

- **System capabilities supporting mobility** – Comprise all network and power connection features. These capabilities include connection awareness, message queuing and security.

  Connection awareness allows the document sharing architecture to be aware of network and power connection states, thus freeing users from such concerns.

  Message queuing is an essential element of hiding network connection states and ensuring reliability.

  Security is a requirement for all distributed architectures. In a document sharing architecture for mobile applications, the security architecture must deal with mobile nodes, a variety of wireless protocols, and unreliable connections and sessions. Because of the complexity of the issue and the variety of ways to implement a security infrastructure in a mobile environment, a full discussion is beyond the scope of this architecture guide.

**Server Components**

A server makes files in its local file system available to clients. In a document sharing architecture, the server may also be a client and a client may also be a server. The server side of a document sharing architecture for mobile applications is shown in Figure 4 and described below:

- **Document sharing application** – Documents that are shared must be managed by a document sharing application. This application is responsible for communicating with and handling messages from the file system agent on each client. It keeps a log of all document changes and contains functionality that provides document version awareness. It may also include a content adapter.

  The document sharing application tracks all changes to files. This log of document changes is used primarily when a client requests an incremental update of the directory tree of the shared document store.

  To facilitate the reintegration functionality of the document sharing architecture, the document sharing application provides version awareness.

  A content adapter in the document sharing application on the server provides the same functionality as a content adapter in the file system agent on the client. The content adapter modifies document content according to client device capabilities, network bandwidth, document structure and user settings. Depending on resource requirements, the content adapter may reside on the client or on the server.

![Figure 4. Server components of a document sharing architecture for a mobile application.](image-url)
**Shared Files** – The files that are made available to remote users.

**System capabilities supporting mobility** – Network and power connection features including connection awareness, message queuing and security. See the section [Client Components](#) on page 15 for more details about system capabilities that support mobility.

### Implementation Issues

#### Reintegration and Document Structure

If the reintegration manager were to utilize information provided by the document management application about a document's structure, it could provide sub-file level reintegration similar to that provided in a database. Only segments of a document would then need to be reintegrated. This would allow more efficient use of some system resources, such as network or storage resources. However, more robust processing resources would be required, especially if the reintegration involved non-deterministic human language.

The distinction between a document and a database is further eroded if the file system itself is viewed as a database and the files in the file system as if they are objects in the database. In addition, a file itself can be considered a complex collection of objects in a database, such as, for example, an HTML document. If the document management application becomes more tightly integrated with the database management application that manages the file system, the distinction between document and database all but disappears.

The complexity of the document structure must also be considered. Automatic reintegration is possible for a document that has a rigid structure, such as an Adobe* Acrobat* form. If two users make simultaneous edits to two different copies of such a document, and one modifies the contents of a cell that contains a street address, for example, and the other modifies the contents of cell that contains marital status information, the reintegration manager can easily combine the edits into a common version of the document. On the other hand, a document such as an MP3 audio file has very little structure, making sub-file integration impractical.

### Content Adapter: Client vs. Server

The content adapter may reside on the client system or the server system. In either location, it requires access to the following:

**System resources** – From a network resource utilization point of view, the best location for the content adapter is on the server. Documents on the server typically contain the richest content, so when the content adapter is on the server, it will typically transfer smaller documents to resource-constrained clients. The server may also have access to more processing resources.

**Knowledge of document format** – Unless the document format is based on a standard, such as HTML, the best resource for knowledge of the document format is the document management application. The server system may or may not have a document management application for all of the documents in its shared file store. However, the client system must have access to the document management application for any document it is requesting if the data in the document is to be useful. Therefore, a client adapter located on the client always has access to the document format information available from the document management application.

**Knowledge of target system resources** – For the server to obtain knowledge about system resources for all its clients, it would need some kind of database in which to store the client resource information, or client resource information would need to be given to the server by the client when the client requests a document. However, when a content adapter is located on the client, it has inherent access to client resource information.

The best solution may be to allow the document sharing architecture the flexibility to make the decision to determine whether to use a content adapter located on the server or on the client at the time files are shared.
Directory Tree Synchronization: Complete vs. Incremental

The document sharing architecture must ensure concurrency between the version of a document on the server and the version in a client's remote file cache. To ensure concurrency, the client must regularly request a snapshot of the server directory tree that includes version information for each file. It can request a complete snapshot or an incremental snapshot that contains only changes since the last request.

A complete snapshot guarantees no disparity between client and server data. However, if a large number of documents are being shared, performance may be unacceptable and much of the information exchanged unnecessary.

An incremental snapshot contains only the changes to the directory tree since the client last submitted a request, as determined by the time stamp of the last snapshot obtained. This method, while more efficient, introduces the possible loss of concurrency.

The optimum solution will likely combine the use of complete and incremental snapshots.

Server Design: Stateless vs. Stateful

One concern is whether or not the document sharing application should maintain state information about clients using the documents it shares. For example, maintaining a database of subscribers and shares would allow the document sharing application to immediately push document changes out to users without waiting for them to request updates. In an always-connected environment, this is advantageous because changes are sent over the network immediately. In a mobile environment, however, updates can only be made when the client is connected to the network. Not only is the immediacy lost, but if the queue of updates is large, substantial system resources may be required to empty it.

In a document sharing architecture for mobile applications, the process of polling by the client to detect changes is efficiently enough to outweigh the complexity of a stateful server. Client authentication and authorization information will need to be maintained for security purposes, but the document sharing architecture does not need to maintain state information.

Summary

A document sharing architecture allows electronic documents to be shared while maintaining concurrency between copies of the documents. In an always-connected environment, concurrency can be accomplished through various means such as restricting modification permissions. In an environment where users are not always connected, the architecture must provide a means to resynchronize modified copies of a document by integrating changes into a common version of the document.
Database Architecture for Mobile Applications

A database application provides an organized collection of data records or files, such as sales transactions, product catalogs and inventories, and customer profiles, allowing users to read or modify data, perform analyses and generate reports.

A database differs from a document in that when modifications are made to a copy of the data, only the portion of the file containing the modified data needs to be reintegrated into the original version. When a copy of a document has been modified and needs to be reintegrated, the entire original must be overwritten with the modified version.

Database Architecture Overview

Users of traditional databases access data residing on a database server from a client machine physically connected to a corporate network. The data that appears on the client machine is simply a view of the data that resides on the server. This particular architecture is secure, but at the same time limited in that the user can't view or work with data without a network connection. All processing takes place on application servers built specifically for that purpose.

In the simplest form, a database is a single file containing many records of data. In the traditional client-server environment, more than one user can use the same database simultaneously. RDBMS make this possible through the use of internal locking mechanisms that prevent more than one user from modifying a record at the same time.

Database Architecture for Mobile Applications

A database architecture that accommodates mobility allows users to increase their productivity by enabling them to work remotely and access information at anytime from anywhere. It allows users to take advantage of local processing power and accommodates features of portable devices, such as the small form factor of a handheld device or the large display of a laptop.

Issues and Challenges

In a mobile environment, copies of data may exist on several client systems. Since client systems are not continuously connected to the central database, the central database RDBMS is not able to prevent simultaneous changes to the data by more than one user. Instead, local data on each client system must periodically be synchronized with data in the master database—a multifaceted problem with many complexities.

Some of the challenges to designing a database architecture that supports mobile applications include:

- Data on the client system may become outdated during periods when the client is not connected. Messages about pending updates will not be available while the system is disconnected, introducing uncertainty into the validity of the data.
- Conflict resolution becomes more challenging as changes are no longer under the control of the RDBMS.
- Local processing power on the client may be limited compared to the processing power available on the server.
- Proprietary data must be kept secure in remote locations.

Addressing the Challenges

Availability of Data

A mobile user should be able to select the data to be replicated on the client system for use when the system is disconnected from the corporate network. A replication of the entire database should not be required, nor should the user be limited to an arbitrary set of data. The amount of data to be replicated should be limited only by the storage space available on the user's system.

Client/server disconnections should be transparent and seamless to the user. The client application should continue to be well-behaved and data continue to be available to the user.

A user needs to know if data being used in an offline environment is old, irrelevant or transient. The user must be able to base decisions on this data, but the data should be flagged so that the decision results can be updated when live data is again available.

Connectivity and Data Transfer

The database architecture for mobile applications should ensure that transactions are reliably transmitted. During a normal transaction, a network connection is established between the client and the server and the data transfer initiated. When the data transfer is completed, a notification that the transfer was either successful or not is sent to the sender. The failure or success of the transaction shouldn't limit the work the user can do.
For example, if a user is connected to the network and updates a data field, the transaction should be transmitted to the server immediately. If the user’s connection is lost during transmission, the transaction should be queued and transferred whenever connection is re-established. Meanwhile, the user should be able to reference the updated data even though the transmission hasn’t been completed.

The transmission of queued data should not require user intervention. However, the user should be able to define policies that determine which available network to use based on cost, speed and resources such as battery life or power usage. The connection manager should be able to delay transmission based on user-defined policies, even when a viable network connection is available.

**Implementation Strategy**

**Architecture Overview**

The core component of an intelligent offline database architecture is a database proxy that acts as an interceptor between the application logic and the local client database. All database activity is managed by the database proxy.

The database proxy is a part of the mobility proxy, which knows how to communicate to the database proxy and the transport proxy. The transport proxy, system-wide policy manager and context manager are active even when the application is not executing.

The transport proxy is responsible for reliable messaging to and from the master database. It is aware of the client system’s resources, such as network connectivity, cost of connectivity, battery life and other system resources based on information supplied by the context manager.

The context manager detects changes in the system state and notifies the transport proxy according to how the user has defined actions to be taken.

Policies are stored in a system-wide user policy manager that is used by every application on the client capable of functioning in a mobile environment.

These components are shown in Figure 5 and described in more detail below.

![Figure 5. Client-side database architecture for mobile applications.](image-url)
Client Components

The following components are needed to achieve a reliable, offline database system for a mobile application:

- **Transport proxy** – Consists of a reliable messaging system and a repository for inbound messages from external sources. The transport proxy is available to all applications and acts as a system service that is always ready to send and receive messages based on the network state. Each application can register with the transport proxy through the database proxy (described below). Registration allows applications to use the transport proxy as a common service while maintaining message ownership.

  Inbound messages are stored in the IN queue for all applications running on the system. When a connection is made and the client system is determined to be online by the policy manager, a message is sent to the external source of each message letting it know how to contact the client. This is needed because the Internet Protocol (IP) address of the client system may change as the client roams or moves from network to network.

  When the client system is online, data is transmitted from the OUT queue to the external source. The method of transport is determined by the policy manager. When the application becomes available, data is processed from the IN queue by the database proxy through the DB Proxy interface. Each packet in the IN or OUT queue is associated with a particular application through a unique application identifier embedded within the package. The proxy needs to know how to contact the server associated with each message in the queue. This should be done in a secure manner, preferably by minimizing user interruptions through the use of a single sign-on service.

- **Policy manager** – Uses the context manager to determine the network connectivity state. It receives notification messages about context, such as which network adapter is being used, the state and quality of the connection, or available battery power. The policy manager uses this information to make decisions according to a policy set by a user based on external factors, such as cost and location.

  For example, a particular user could have a policy that states that when an ultra-high priority message is ready to be sent or received, any available network should be used no matter the cost of connection or the amount of battery power remaining. If the system supports the ability to continue partial downloads, a break in the network connection won’t require a full retransmission of data. Another policy might state that when the battery is below 10%, the display brightness is to be reduced and network devices turned off to save power.

- **Context manager** – Continuously monitors the state of managed resources within the system, such as network connectivity and power. For network connectivity, the context manager tracks the quality of service, network connectivity and bandwidth of all network devices to generate trends, raw data and notification messages to be used by the policy manager. Other resources are managed in the power library component of the context manager, including battery life, processor frequency and display brightness, which can affect the quality of experience and the amount of time the system is available.

- **Database proxy** – An intermediate layer between the application and the database and between the database and the reliable message queues. When an application requests data from the database, the call goes through the database proxy, which has adapters for Open Database Connectivity (ODBC) and Java Database Connectivity (JDBC) interfaces. The information is then retrieved from the local database.

  When an application writes to the database, the database proxy first writes the data to the local database and then sends the information to the transport proxy for synchronization with the remote database. The information sent to the remote database may instead be a stored procedure, such as an SQL command, rather than the entire set of data records modified by executing the procedure.

  The database proxy is in constant communication with the application interface, which tells the database proxy what information is currently being used. When the system is online, the database proxy is in charge of registering methods of communication with the transport proxy, such as addresses to functions for updating the data, which the transport proxy uses to put the data into the local storage. This allows the database proxy
to know what data is new at the time the data is entered into the local database. The database proxy may not know, however, when the data is stored in the transport proxy’s queue, because the application may not be available at that time.

Since the database proxy knows what data is currently being used and what data has been changed, it can then send messages, based on user policy, back to the application through the application interface. These messages allow the application to take actions such as refreshing the data currently being shown or asking the user to refresh the data.

- **Client mobility proxy** – The centralized communicator between the application and the other proxies. It launches with the application and notifies the database proxy so that it can process inbound messages. The client mobility proxy contains the application interface through which the application is notified when events occur, such as database updates. The user has the option to force synchronization by downloading the entire subscribed data or portions of it.

  The application interface can retrieve the state of the system from the policy manager to perform such actions as, for example, shutting down components of the application in response to a low power condition.

### Server Components

The server side of the database architecture for mobile applications is similar to the client side of the architecture with a few notable exceptions described below:

- **Transport proxy** – On the server side, the transport proxy tracks the destination address of each client. It communicates with any proprietary single sign-on service to verify the identity of the client so that it can read client messages to determine the location of the client. The transport proxy may rely on other system capabilities, such as security functions or a single sign-on service, to maintain the mapping between a client and the client IP address. When a client IP address changes, all packets in the OUT queue that are waiting for delivery are rerouted to the correct address.

- **Database proxy** – On the server side, the data proxy uses an ODBC, JDBC or other database adapter to manage updates from the IN queue to the back-end database. Items in the queue can have varying priority based on circumstances defined by business logic and/or user policies. For example, a user can decide that all data transmitted as a result of a forced synchronization will receive the highest priority, or when crucial data needs to be pushed from the server, the server can force a higher priority on the data packets to be sent.

  The typical procedure for sending data is as follows: When data that a client has subscribed to changes, a message is sent to the client providing an alert of the change. User-defined policies that take into account availability of resources determine whether the resulting action is to stop or allow push transactions from the server.

### Limitations of the Solution

The most significant limitation of this solution is that it does not include a formal definition of conflict resolution and security capabilities. Although these capabilities can be discussed in generalities, specific implementations are largely dependent upon case-by-case business logic and are left up to the developer to implement. Another limitation of this solution is that details are not provided about how to handle some of the more complex issues that arise related to database synchronization, such as operations dependent on data that is no longer valid or that is in conflict.

### Summary

A database provides a collection of data that is organized so that its contents can be easily accessed, managed and updated. In an always-connected environment, data is maintained in a single central database controlled by an RDBMS. In a mobile environment, data is replicated on the client system to make it available to an offline user. The integrity of data is maintained through automatic updates whenever the client system is connected to the network.
Conclusion

An architecture for mobile applications allows a mobile user to keep working productively whether offline or online. It enables consistent operation regardless of the status of the network connection, allows seamless connectivity independent of network type, provides multi-platform support for a variety of client devices, and optimizes power and performance.

Three types of applications widely used in mobile environments include portal, document sharing and database applications, which have been described in some detail. This *Mobile Application Architecture Guide* illustrates the issues that arise in each of these application areas in an environment in which mobile computing devices are used and presents implementation strategies to help developers address these issues.

References

The following references contain information that may be of interest to readers of this guide:

