A

Seminar report

On

Web Services On Mobile Platform

Submitted in partial fulfillment of the requirement for the award of degree
of Bachelor of Technology in Computer Science

SUBMITTED TO:

www.studymafia.org

SUBMITTED BY:

www.studymafia.org

Acknowledgement

I would like to thank respected Mr…….. and Mr. ……..for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present a seminar report. It helped me a lot to realize of what we study for.

Secondly, I would like to thank my parents who patiently helped me as i went through my work and helped to modify and eliminate some of the irrelevant or un-necessary stuffs.

Thirdly, I would like to thank my friends who helped me to make my work more organized and well-stacked till the end.

Next, I would thank Microsoft for developing such a wonderful tool like MS Word. It helped my work a lot to remain error-free.

Last but clearly not the least, I would thank The Almighty for giving me strength to complete my report on time.

Preface

I have made this report file on the topic **Web Services On Mobile Platform** ; I have tried my best to elucidate all the relevant detail to the topic to be included in the report. While in the beginning I have tried to give a general view about this topic.

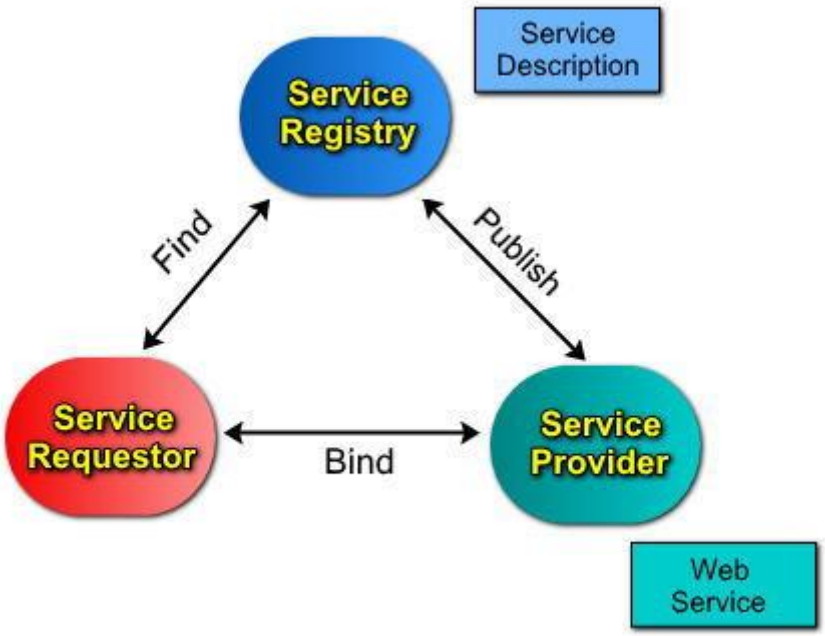
My efforts and wholehearted co-corporation of each and everyone has ended on a successful note. I express my sincere gratitude to ……..who assisting me throughout the preparation of this topic. I thank him for providing me the reinforcement, confidence and most importantly the track for the topic whenever I needed it.

1. Introduction

1.1 What is a Web service?

Web services are distributed application components that are externally available. You can use them to integrate computer applications that are written in different languages and run on different platforms. Web services are language and platform independent because vendors have agreed on common web service standards. A **web service** is typically an application programming interface (API) or **Web API** that is accessed via Hypertext Transfer Protocol (HTTP) and executed on a remote system, hosting the requested service.

The W3C defines a "web service" as "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically Web Services Description Language WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."

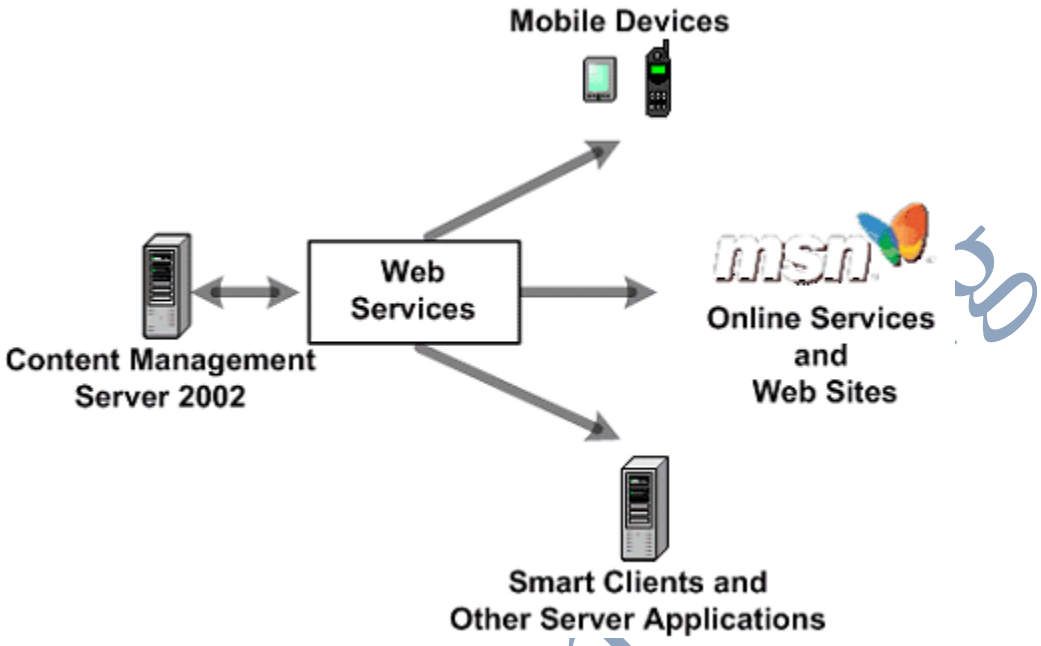


Unlike traditional client/server models, such as a Web server/Web page system, Web services do not provide the user with a GUI. Web services instead share business logic, data and processes through a programmatic interface across a network. The applications interface, not the users. Developers can then add the Web service to a GUI (such as a Web page or an executable program) to offer specific functionality to users.

Web services allow different applications from different sources to communicate with each other without time-consuming custom coding, and because all communication is in XML, Web services are not tied to any one operating system or programming language.

1.2 The Role of Web Services on Mobile Platform

Web services are an emerging technology that provides interoperability between applications running in different platforms. The Web services technology provide the best approach to Service Oriented Architecture envision of component collaboration for better business requirements fulfillment in large enterprise systems. The challenges in implementing Web services consuming clients for low-resources mobile devices connected through unreliable wireless connections are delimited. The paper also presents a communication architecture that moves the heavy load of XML-based messaging system from the mobile clients to an external middleware component. The middleware component will act like a gateway that lightly communicates with the device in a client-server manner over a fast binary protocol and at the same time takes the responsibility of solving the request to the Web service. Keywords: Web Services, SOA, Mobile Computing, Mobile Devices.



1.2.1 Introduction

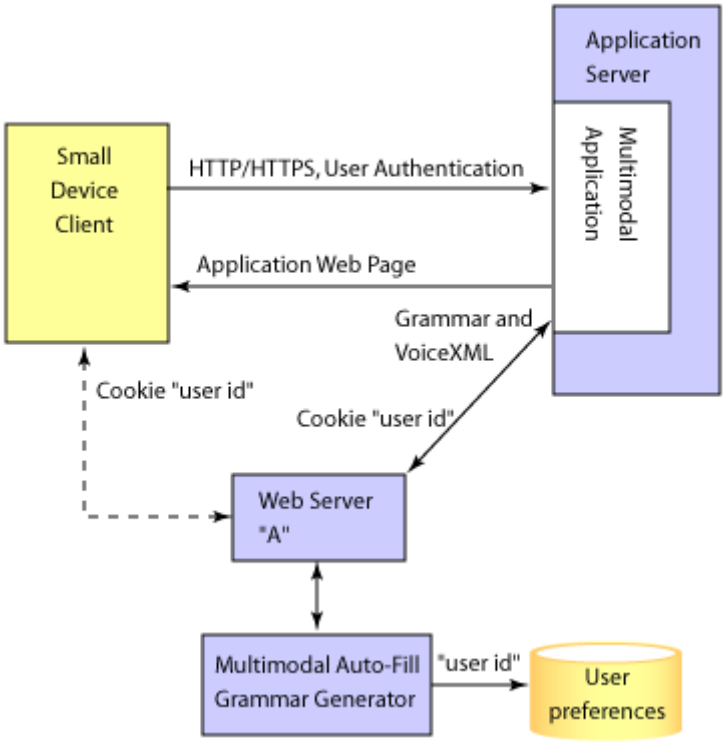
The current trend in the application space is moving towards Service-Oriented Architecture (SOA) paradigm which describes a flexible set of design principles for building interdependent and interoperable application components that act like services by implementing a specific functionality and publishing a communication interface to access it. The paradigm envisions component collaboration for better fulfillment of business requirements in large enterprises systems. Systems built with these principles will be more flexible to changes in business requirements as opposed to traditional monolithic systems that are extremely sensitive to any change in one of its subsystems.

The Web services technology evolved from the most logical approach of applying the SOA vision to the Web space. This technology provides interoperability over the Web between applications running on different platforms, ensuring a standardized protocol for business partners to exchange information. Being built over the eXtensible Mark-up Language (XML) and Hypertext Transfer Protocol (HTTP) standard protocols, Web services can be easily deployed and accessed from anywhere over the Web.

1.2.2 Description of Web services

Web services are software components build upon Web-based technologies including HTTP and XML and allow standard means of interoperability over the Internet or intranets between application running on a large variety of hardware and platforms. Web services act like self-contained components that are published, located and invoked over the Web. The key concept behind Web services is to provide a standard platform and operating system independent mechanism for application communication over the Web.

The Web Service Architecture (WSA) proposed by W3 organization relies on a number of Web standards like XML, Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL) and Universal Description Discovery and Integration (UDDI) that allow services to be described, searched and integrated by any application. Main Web services implementations fall into one of the two categories: SOAP Web services and REpresentational State Transfer (REST) Web services introduced for the first time by. SOAPbased Web services are the preferred way to implement the SOA initiative in today's complex and heterogeneous computing environment. SOAPbased Web services present greater flexibility at lower integration costs over RESTful Web services that instead offer great performance through lighter messaging system.



There are two major limitations that SOAP-based Web services have: performance issues due to XML processing overhead and lack of support for transaction in communication. Communication with a Web service may suffer poor performance over busy or unstable networks in comparison with other traditional approaches to distributed computing like CORBA or DCOM, due to the verbose nature of the XML-based messaging system that was simply not intended for efficiency. Moreover, extra overhead when encoding/decoding XML requests/responses definitely count on the overall Web application performance. Lack of transaction support in the communication with the Web service makes this kind of data exchange protocols rather stateless as the Web service provider and Web service consumer don't have knowledge of each other's state.

As wireless network access becomes a standard feature in the nowadays mobile devices, mobile Web services consuming challenges are worth investigating from two points of view: from the point of mobile hardware capabilities (both processing power and network capabilities) and the point of native support on the current mobile platforms. Also,

adapting the existing standards to the limitations of resourceconstrained mobile devices should be considered.

1.2.3 Limitations of mobile devices

About fifteen years ago the mobile phones had huge sizes and carrying them was an issue. They were about phone calls only. Today the mobile revolution brings mobile devices that can easily fit in your back pocket and feature incredible entertainment and multimedia capabilities. The capabilities of mobile devices are constantly increasing to satisfy an even more increasingly need for flexibility in terms of mobility and connectivity. Cellular networks are expected to sustain a 40 time increase of data traffic in the next 5 years. Data traffic will come from Web browsing in general and video watching in particular.

Sites like give an up to date hardware features comparison of existing smart phones - the most appealing mobile devices on the market in terms of mobility. Besides the obvious advances in computational capabilities (e.g. faster processor, greater amount of ROM and RAM), the trend is towards providing wireless network communication capabilities and high connectivity features. The main hardware limitations of mobile devices that need to be addressed in the near future are huge battery consumption - that translates into poor battery life, low wireless network bandwidth and wireless connection instability. Decent levels of mobility and connectivity can be achieved only with a high battery autonomy while being constantly connected to a wireless access point and consuming a service with little or no delays.

1.2.4 Existing support on mobile platforms

This section is intended to compare the development effort required by Web services client implementation on most important mobile platforms and discuss the various levels of support that they provide.

Sun moved recently towards enabling Java technology-capable mobile devices with a standardized model to access the existing Web Services by extending J2ME platform with JSR (Java Specification Request) 172. The request also named J2ME Web Services

Specification (WSS) leverages the J2ME for the standard Java Web services platform thus allowing developers to easily create clients for Java-enabled mobile devices also. The WSS specification is part of the Mobile Service Architecture platform designed to meet the evolution of market towards incorporating new technologies and services in mobile devices by providing a standardized application environment for all Java technology-enabled mobile devices.

Apple has made little effort in supporting Web services consuming on its mobile iPhone platform. The SDK does not present native Objective-C libraries for simple creation of Web services clients and moreover, the existing NSXML library of Cocoa framework is rarely used when implementing the needed XML-based messaging mechanism behind. Libraries like libxml2 or KissXML are definitely preferred when it comes for parsing the XML bloat. Work-arounds on iPhone platform include the usage of existing utility applications like wsdl2objc that generate stubs proxy classes to access the service from the WSDL specification. The generated classes contain all the methods that the Web service exposes. Nevertheless noticeable efforts have been conducted by open source community to create easy to use frameworks for Web services access.

Google has also shown no interest in making SOAP Web services consuming an easy task for the Android platform developer community. The SDK is not even bundled with tools to generate stubs for the Web service's interface. Workarounds are more like do-it-yourself solutions based on kSOAP 2 more flexible and complete XML parsing library as Android is indeed a Java-based platform but not fully a J2ME one. The overall feeling in the development community is that the Android platform best fits as a deliverer of the Google's services in the mobile space.

Consuming Web services is an easy job on Windows Mobile as .NET Compact Framework is well suited for both synchronous and asynchronous access. By simply referencing a Web service, proxy classes that expose the service's interface are generated. This is also the case for the Windows Phone 7 platform.

Nokia provides the Serene framework (from Symbian Series 60 platform) for easy creation of application clients that consume Web services. The framework relies on the J2ME JSR 172 specification to provide full support in creating relevant and rich web applications.

Except for the Windows and Nokia platforms that provide a level of support that allows the developer to concentrate on design and creativity rather than finding dirty workarounds, developing SOAP Web services clients requires a lot of extra effort mostly due to the lack of native support. Moreover, specific workarounds are required for specific platforms making very resource consuming the task of implementing a Web application on all mobile platforms.

2. Types of web services

Types of Web Services:-

On the conceptual level, a service is a software component provided through a network-accessible endpoint. The service consumer and provider use messages to exchange invocation request and response information in the form of self-containing documents that make very few assumptions about the technological capabilities of the receiver.

On a technical level, web services can be implemented in various ways. The two types of web services can be distinguished as: -

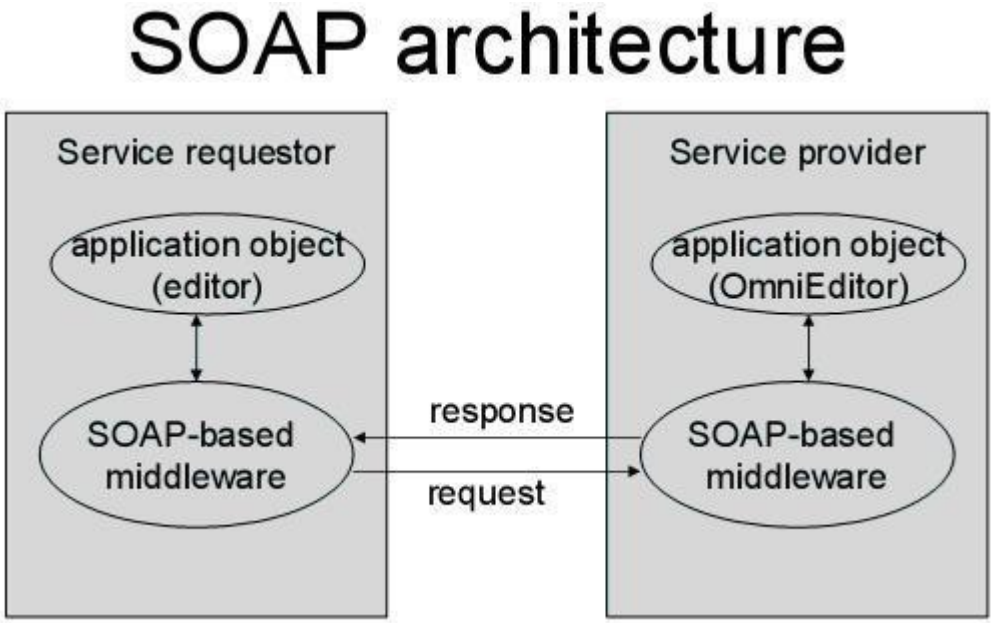
- **"big" web services (SOAP)**
- **"RESTful" web services**

2.1 Big OR SOAP web services:-

SOAP, originally defined as Simple Object Access Protocol, is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks. It relies on Extensible Markup Language (XML) for its message format, and usually relies on other Application Layer protocols, most notably Remote Procedure Call (RPC) and Hypertext Transfer Protocol (HTTP), for message negotiation and transmission. SOAP can form the foundation layer of a web services protocol stack, providing a basic messaging framework upon which web services can be built. This XML based protocol consists of three parts: an envelope, which defines what is in the message and how to process it, a set of encoding rules for expressing instances of application-defined data types, and a convention for representing procedure calls and responses.

SOAP consists of three parts:

- The SOAP envelope construct defines an overall framework for expressing what is in a message; who should deal with it, and whether it is optional or mandatory.
- The SOAP encoding rules defines a serialization mechanism that can be used to exchange instances of application-defined data types.
- The SOAP RPC representation defines a convention that can be used to represent remote procedure calls and responses.



2.1.1 The SOAP Message Exchange Model

SOAP messages are fundamentally one-way transmissions from a sender to a receiver, but as illustrated above, SOAP messages are often combined to implement patterns such as request/response.

SOAP implementations can be optimized to exploit the unique characteristics of particular network systems. For example, the HTTP binding provides for SOAP response messages to be delivered as HTTP responses, using the same connection as the inbound request.

Regardless of the protocol to which SOAP is bound, messages are routed along a so-called "message path", which allows for processing at one or more intermediate nodes in addition to the ultimate destination.

A SOAP application receiving a SOAP message MUST process that message by performing the following actions in the order listed below:

1. Identify all parts of the SOAP message intended for that application
2. Verify that all mandatory parts identified in step 1 are supported by the application for this message and process them accordingly. If this is not the case then discard the message. The processor MAY ignore optional parts identified in step 1 without affecting the outcome of the processing.
3. If the SOAP application is not the ultimate destination of the message then remove all parts identified in step 1 before forwarding the message.

2.1.2 Relation to XML

All SOAP messages are encoded using XML A SOAP application SHOULD include the proper SOAP namespace on all elements and attributes defined by SOAP in messages that it generates. A SOAP application MUST be able to process SOAP namespaces in messages that it receives. It MUST discard messages that have incorrect namespaces and

it MAY process SOAP messages without SOAP namespaces as though they had the correct SOAP namespaces.

SOAP defines two namespaces:

1. The SOAP envelope
2. The SOAP serialization

2.2 RESTful" web services: -

Introduction

REST is a collection of architecture principles and a software architecture style to build network-enabled systems based on mechanisms that define and access resources, such as the World Wide Web. The term REST is often used rather loosely to describe a framework to transmit data over a protocol such as HTTP without adding additional semantic layers or session management.

REST defines a strict separation of concerns between components that participate in a client-server system that simplifies the implementation of actors involved. REST also strives to simplify communication semantics in a networked system to increase scalability and improve performance. REST relies on autonomous requests, between participants in a message exchange which implies that requests must include all information that a client or server requires understanding the context of the request. In a REST-based system, you use minimal sets of possible requests to exchange standard media types.

The REST principle uses uniform resource identifiers (URIs) to locate and access a given representation of a resource. The resource representation, known as representational state, can be created, retrieved, modified, and deleted. For example, you can apply REST to document publishing to make documents available to readers. At any given time, the publisher can present Web URLs so that readers can access information (representational state) about the publisher's documents. Document readers need only know the URLs to read the document information and, if authorized, modify the information.

One of the defining principles of REST is that it can exploit existing technologies, standards, and protocols pertaining to the Web, such as HTTP. This reliance on existing technologies and protocols makes REST easier to learn and simpler to use than most other Web-based messaging standards, because little additional overhead is required to enable effective information exchange.

2.2.1 Fundamental entities of REST: -

REST defines the following fundamental entities:

- **Data elements:** Data, identifiers (URIs and URLs), and representations of data such as HTML documents, XML documents, and images
- **Components:** Origin servers such as Apache httpd and Microsoft® Internet Information Services (IIS), gateways such as Squid and CGI, proxies such as Gauntlet and Netscape proxy, and user agents such as Web browsers or mobile devices
- **Connectors:** Client connectors such as libwww, server connectors such as NSAPI, caches such as a browser cache, and others

Some of the fundamental REST entities and how they can interact within a typical networked system. Notice that requests are transmitted and received by connectors that are tied to a given entity, such as a server or database. Connectors handle transmission duties for a given protocol, thereby allowing an entity to transmit or receive requests in the same manner over any protocol, as long as a connector is present for the desired protocol.

2.2.2 REST and HTTP

According to Fielding's dissertation, in an HTTP-based REST system, you should use standard HTTP methods—namely, GET,PUT, POST, and DELETE —to access the representational state for a resource:

- **GET:** Use this method to transfer the ***current*** representational state of a resource from publisher to consumer.
- **PUT:** Use this method to transfer the ***modified*** representational state of a resource from consumer to publisher.
- **POST:** Use this method to transfer the ***new*** representational state of a resource from consumer to publisher.
- **DELETE:** Use this method to transfer information needed to change the representational state of a resource to ***delete.***

Java server technologies, such as servlets and JavaServer Pages (JSP™), support the standard HTTP methods. The following sections discuss how to extend these technologies to support the APP from a REST-based system using Java technology.

2.2.3 The Atom Publishing Protocol

The APP is an HTTP-based protocol for publishing, deleting, and updating resources. APP stresses the concept of using the fundamental operations that the HTTP protocol provides (such as GET, PUT, POST, and DELETE) to transfer instances of introspection documents and collection documents that contain the kinds of resources and feeds

available for publish, updates, and so on. After the resources and feeds are discovered, their associated URIs are used for publication and modification. Member feeds contain things like blog entries, podcasts, wiki documents, and calendar events. The APP is also being investigated and used as a front-end API for a wide array of data-storage services, such as database servers, document and content management services, and software repositories.

The APP builds upon the popular Atom Syndication Format to formalize many of the mechanisms used in the exchange of semantically rich content using REST concepts and technologies. The protocol operates on containers of Web resources known as collections. Collections are presented for discovery in introspection documents. All collections contain member resources, which are the ultimate targets for creation, access, modification, and deletion. Interactions with a collection and its member resources are based on the common HTTP verbs:

- **GET:** Used to retrieve a representation of a collection or member resource
- **POST:** Used to create a new member resource
- **PUT:** Used to update a member resource
- **DELETE:** Used to remove a member resource

2.2.4 Fundamental entities of APP

The APP defines the following fundamental entities:

- **Introspection document:** The introspection document has a media-type *application/atomserv+xml* and describes*workspaces*, which are arbitrary server-defined groupings of collections. A collection can appear in more than one workspace. Listing 1 is an example of a typical introspection document.

Listing 1. A typical introspection document

```
<?xml version="1.0" encoding="utf-8" ?>
<service xmlns="http://www.w3.org/2007/app" xmlns:atom="http://www.w3.org/2005/Atom">
- <workspace>
   <atom:title>Main Site</atom:title>
   - <collection href="http://localhost:8080/atompub/services/collections/main">
       <atom:title>My Main Page</atom:title>
       <accept>application/atom+xml;type=entry</accept>
       <categories fixed="yes" />
     </collection>
     <collection href="http://localhost:8080/atompub/services/collections/pics">
       <atom:title>My Pictures</atom:title>
       <accept>*/*</accept>
       <categories fixed="yes" />
     </collection>
  </workspace>
- <workspace>
   <atom:title>Documents</atom:title>
```

```
  - <collection href="http://localhost:8080/atompub/services/collections/docs">
      <atom:title>My Documents</atom:title>
      <accept>application/atom+xml;type=entry</accept>
      <categories fixed="yes" />
    </collection>
  </workspace>
</service>
```

The introspection document illustrated above defines a service with two workspaces. The first workspace, called "Main Site", defines two collections called "My Main Page" and "My Pictures". The "My Main Page" collection is found at http://localhost:8080/atompub/services/collections/main and the "My Pictures" collection is found at http://localhost:8080/atompub/services/collections/pics. The second workspace, "Documents", contains one collection named "My Documents" and is found at http://localhost:8080/atompub/services/collections/docs.

- **URI:** A Uniform Resource Identifier.
- **IRI:** An Internationalized Resource Identifier.
- **Member resource:** A data object or service known as an *entry resource* or a *media resource* that an IRI or URI locates. Entry resources are represented as Atom entry documents. Media resources can have representations in any media type. A media resource is described within a collection using a media link entry.
- **Representation:** The state of a given resource transmitted by a request or a response.
- **Collection:** A container of member resources identified by a unique URI. Collections are represented as Atom feeds. To create new entries in a collection, clients send HTTP POST requests to the collection's URI. New entries are assigned a unique URI to be used as location references. To modify entries in a collection, a client retrieves the resource using its URI, makes modifications, then uses PUT to move the modified representation of the resource to the server. To remove the member resource from a collection, issue an HTTP DELETE request to the URI of the member resource. All members of a collection must have an updated property by which a collection document is ordered.
- **Workspace:** A named group of collections.
- **Service document/element:** The top-level element of an introspection document that describes the location and capabilities of one or more collections, grouped into workspaces.
- **Category document:** A document that describes the categories allowed in a collection.

2.2.5 Wrap up

REST is a software architecture style and collection of principles for building distributed systems using existing protocols and standards to act on resources representing data or services. REST has evolved (or devolved) to describe any framework that transmits data over a protocol such as HTTP without relying on additional semantic layers or session management.

The APP is an HTTP-based protocol that accesses, publishes, and modifies resources representing data or services. The APP is designed around the idea of using the standard operations that the HTTP protocol provides (GET, PUT, POST, and DELETE) to introspect collections of member resources and to manipulate member resources representing things like blog entries, images, documents, and calendar entries.

In this tutorial, you learned the basics of REST and the APP. You also learned how to apply REST within an enterprise system to send and receive HTTP requests and responses through a Java servlet-based system using the APP to facilitate accessing and manipulating the representational state of member resources and containers of member resources.

3. Introduction of a middleware component

3.1 What is middleware?

Middleware is the software that connects software components or enterprise applications. Middleware is the software layer that lies between the operating system and the applications on each side of a distributed computer network. Typically, it supports complex, distributed business software applications.

Middleware is the infrastructure which facilitates creation of business applications, and provides core services like concurrency, transactions, threading and messaging for service-oriented architecture (SOA) applications. It also provides security and enables high availability functionality to your enterprise.

Middleware includes Web servers, application servers, content management systems, and similar tools that support application development and delivery. It is especially integral to information technology based on Extensible Markup Language (XML), Simple Object Access Protocol (SOAP), Web services, SOA, Web 2.0 infrastructure, and Lightweight Directory Access Protocol (LDAP)m etc.

3.2 Functions of Middleware

Applications use intermediate software that resides on top of the operating systems and communication protocols to perform the following functions:

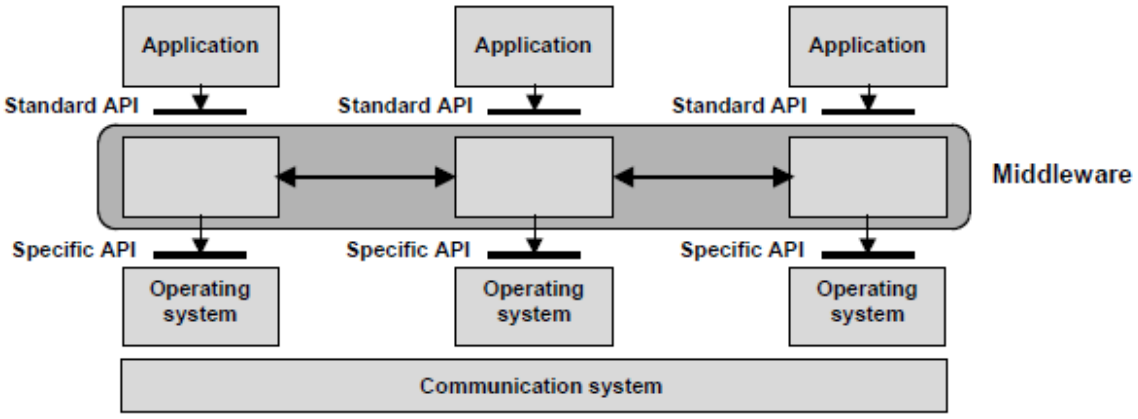
- Hide the distributed nature of the application. An application represents a collection of interconnected parts that are operational and running in distributed locations, out of view.
- Hide the heterogeneity of the enterprise. This includes the hardware components used, computer operating systems, and communication protocols.
- Provide uniform, standard, high-level interfaces to the application developers and integrators, so that applications can be easily composed, reused, ported, and made to interoperate.

- Supply a set of common services to perform various general purpose functions to avoid duplicating efforts, and to facilitate collaboration between applications.

3.3 Middleware Architecture Design

The function of middleware is to mediate interaction between the parts of an application, or between applications. Therefore, considerations for architectural structure play a central role in middleware design. The architectural design encompasses the organization, overall structure, and communication patterns, both for applications and for the middleware itself.

Besides architectural aspects, the main problems of middleware design pertain to various aspects of distributed systems. Any middleware system relies on a communication layer that allows its different pieces to interoperate. In addition, communication is a function provided by middleware itself to applications, in which the communicating entities may take on different roles such as client server or peer-to- peer. Middleware allows different interaction modes (synchronous invocations, asynchronous message passing, coordination through shared objects) embodied in different patterns.



Therefore, middleware system design faces several challenges:

Middleware systems rely on interception and indirection mechanisms, which induce performance penalties. Adaptable middleware introduces additional indirections, which make the situation even worse.

As applications become more and more interconnected and interdependent, the number of objects, users, and devices tends to increase. This poses the problem of the scalability of the communication and object management algorithms, and increases the complexity of administration. The availability, reliability, concurrency, security, and performance of applications may also be an issue.

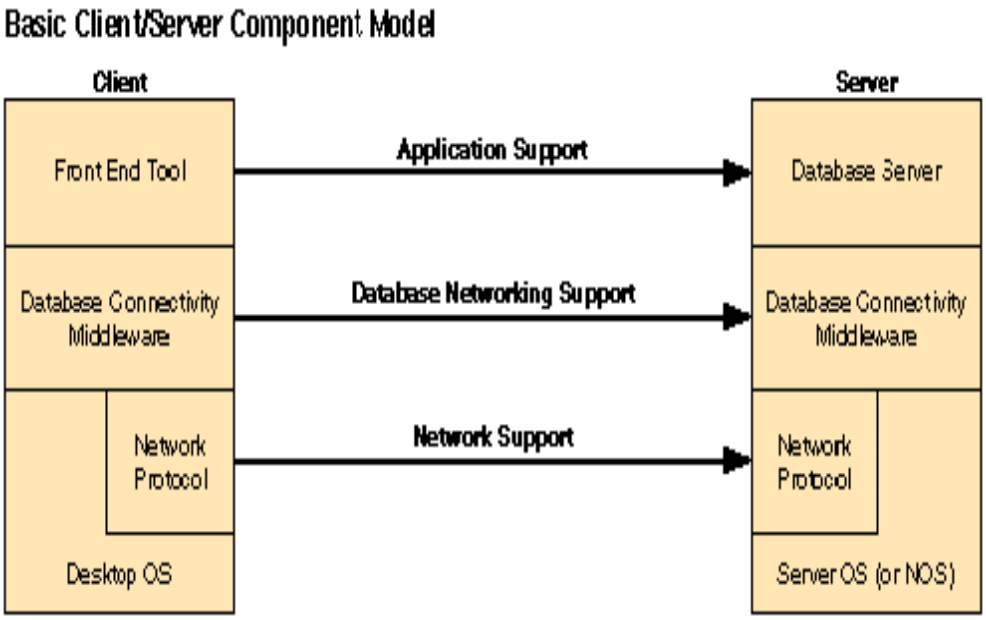
Widespread computing is a vision of the near future, in which an increasing number of devices embedded in various physical objects participate in a global information network. Mobility and dynamic reconfiguration will be dominant features, requiring permanent adaptation of the applications.

Managing large applications that are heterogeneous, widely distributed, and in permanent evolution raises many questions, such as consistent observation, security, trade-offs between autonomy and interdependence for the different subsystems, and definition and implementation of resource management policies.

Maybe we can be a bit more specific about what middleware is by describing what it does.

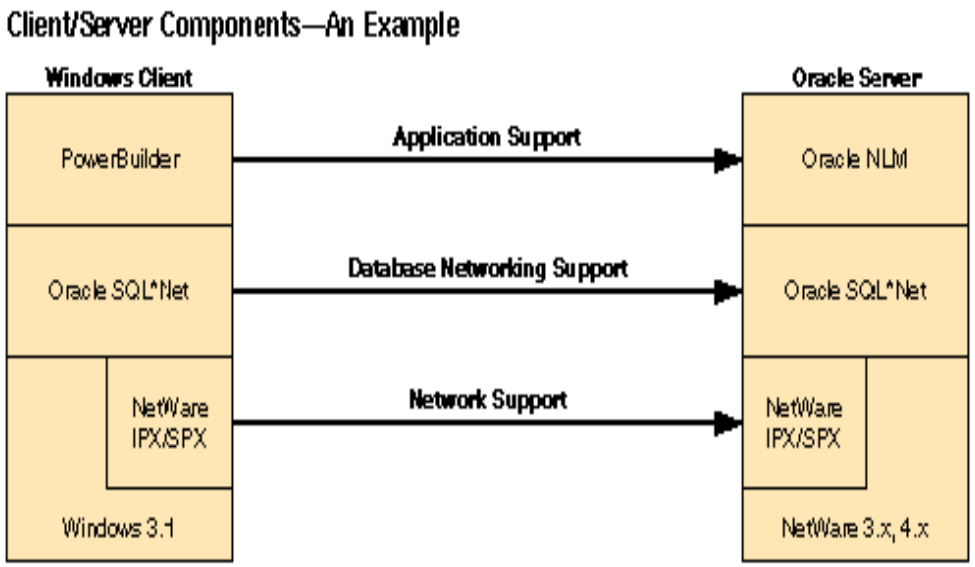
3.3.1 Basic Client/Server Component Model

We've created a simple network application model to help explain the concept of middleware, and we'll use it throughout this guide. Instead of the OSI network model, we have basic chunks on servers and clients, including operating system, application program, and the connectivity pieces like network protocols and middleware software. Here's our model which is defined in the context of database networking middleware: [Diagram: Basic Client/Server Component Model]



The middleware is right there in the middle (called database connectivity middleware in our diagram). Notice that middleware components are instantiated on both client and server platforms. Since some definitions of middleware include code that isolates operating systems from hardware platform differences (like the hardware abstraction layer or HAL in Windows NT), let's keep our discussion focused on middleware for distributed systems. In some cases, there will be more middleware services provided by an intermediate device like a database gateway. The next diagram provides a more specific example using Oracle and PowerBuilder on common platforms:

[Diagram: Client/Server Components-An Example]



What does the middleware do? Simply put, it provides a set of services to applications like a client/server database as depicted in our diagram.

3.4 Basic Middleware Services

The basic set of middleware services are offered by most products today. We'll refer to Microsoft's SQL Server RDBMS and its DB-Library data access middleware as an example. Services offered include:

- **Client/Server Connectivity** -Middleware provides the mechanism by which network applications communicate across the network. This includes in the case of database networking for example the service of putting packages of query results data into network transport packets.
- **Platform Transparency** -Client and server don't have to have intimate knowledge of each other in order for work to get done. Differences between platform specific encodings like big-endian and little-endian or EBCDIC and ASCII are typically hidden by middleware. Middleware often runs on a variety of platforms, letting the organization utilize all its existing desktop and server hardware as applications require.

- **Network Transparency and Isolation** -Middleware often makes networking choices transparent to application programmers. Ac tually, though, every middleware product we've ever heard of runs on TCP/IP, with all the other protocols coming in a distant second.
- **Language Support** -Middleware often provides transparency across different SQL database dialects. Even when coding in embedded SQL in a 3GL, the middleware might allow the use of a single SQL dialect across a variety of RDBMS back ends. Outside of the database specific middleware products, generic middleware products often allow different programming languages to be used to create the distinct pieces of an application.
- **RDBMS Support** -When we focus on database networking middleware (also called data access middleware), middleware may also provide a level of transparency across different data storage formats. It will make different RDBMSs look like the same RDBMS.

3.5 Advanced Middleware Services

These advanced services are not yet common in middleware products, database-specific or more generic. Still, the functionality will be increasingly required by new enterprise applications.

- **Enhanced Security** -Some middleware vendors have security options much better than just usernames and passwords. Look for support of card key solutions like SecureID's products.
- **Location Transparency** -Many middleware solutions don't offer a simple name service for their server or service names. If the user wants to connect to a server and can't remember the name of it, they'll have to call tech support. Advanced middleware solutions offer centralized naming services with some level of distribution. The issues are the same as those associated with DNS on the Internet or NDS on NetWare.

- **Application Oriented Services** *(*Transaction Monitoring, Queuing) -For more generic middleware environments, different application services may be required, including transaction monitoring and message queuing.
- **Interaction With Other Network Services** -While some middleware does provide its own set of advanced services, a better approach may be to offer support for external enterprise services that might already be in use in the organization.

4. Web Services on J2ME

4.1 Introduction

WSA is designed to work with J2ME profiles based on either the Connected Device Configuration (CDC) or the Connected Limited Device Configuration (CLDC 1.o or CLDC 1.1). The remote invocation API is based on a strict subset of J2SE's Java API for XML-Based RPC (JAX-RPC 1.1), with some Remote Method Invocation (RMI) classes included to satisfy JAX-RPC dependencies. The XML-parsing API is based on a strict subset of the Simple API for XML, version 2 (SAX2).

The goal of WSA is to integrate fundamental support for web services invocation and XML parsing into the device's runtime environment, so developers don't have to embed such functionality in each application -- an especially expensive proposition in resource-constrained devices like mobile phones and personal digital assistants.

4.2 Core Specifications

The core specifications and application-level protocols that define web services are promoted by the Web Services Interoperability Organization (WS-I), and governed by the World Wide Web Consortium (W3C) and the Organization for the Advancement of Structured Information Standards (OASIS). Four key standards specify how to create, deploy, find, and use web services:

Web Services Standard	Description
Simple Object Access Protocol (SOAP) 1.1	Defines transport and data encoding
Web Services Definition Language (WSDL) 1.1	Defines how remote services are described
Universal Description, Discovery, & Integration (UDDI) 2.0	Defines how remote services are discovered
Extensible Markup Language (XML) 1.0, and XML Schema	Defines the Extensible Markup Language (XML) and XML Schema

These primary specifications tend to be very broad, and developers of web services have found it difficult to achieve full interoperability. To resolve differences in interpretations of the standards, WS-I has defined a set of conformance rules called the WS-I Basic Profile, version 1.0. JSR 172 conforms to the Basic Profile.

4.3 Web Services on the J2ME Platform

JSR 172 specifies standardized client-side technology to enable J2ME applications to consume remote services on typical web services architectures, as Figure 1 illustrates:

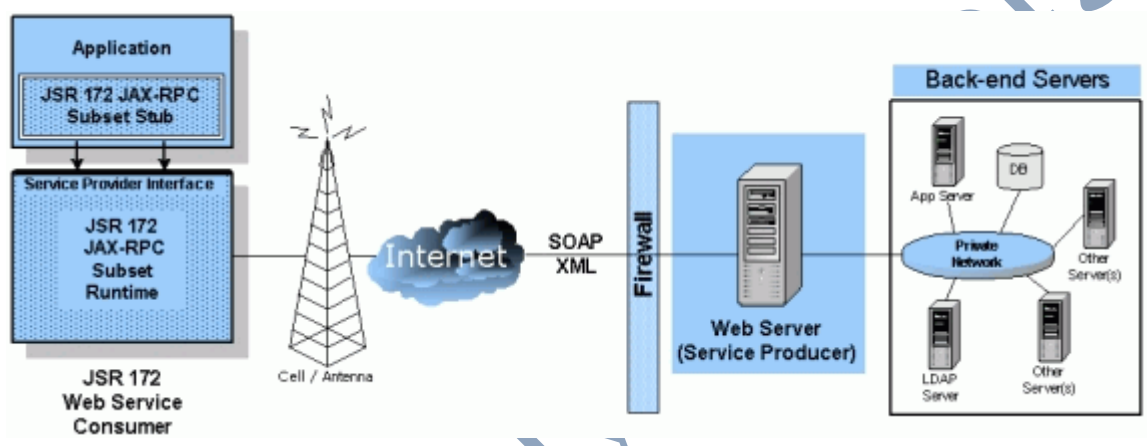


Figure 1. J2ME Web Services in a Typical Web Service Architecture

At a high level, this web service architecture has three elements:

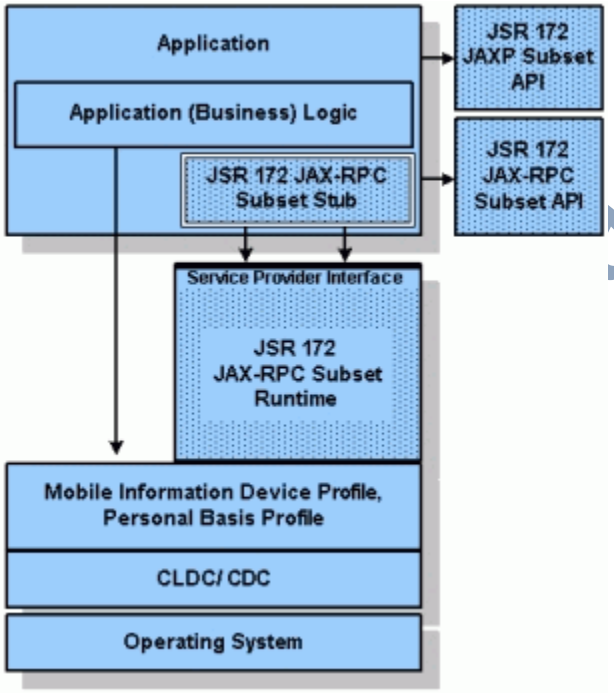
- A network-aware application residing on a WSA-enabled wireless device. The application includes a JSR 172 stub that uses the JSR 172 runtime to communicate with the network. I'll describe soon the roles that the stub and the runtime play.
- The wireless network and the Internet, and the corresponding communication and data-encoding protocols, including binary protocols, HTTP, and SOAP/XML.
- A web server, acting as the service producer, typically behind one or more firewalls and a proxy gateway. The web server often provides access to back-end applications and servers on a private network.

The first version of WSA addresses only the consumption of web services. It does not support creation and deployment of service endpoints; a J2ME device can be a service

consumer, but not a service producer. JSR 172 also does not specify an API for service discovery using UDDI.

4.4 Understanding JSR 172

To begin to understand how J2ME Web Services works, let's first look at how a typical JSR 172-based application is organized:



The application itself is a smart client based on the Mobile Information Device Profile (MIDP) or the Personal Basis Profile (PBP), with business-specific logic, user interface, persistence logic, and life-cycle and application-state management. To handle XML documents, the application can employ the JAXP subset API. To consume web services, it can use the JAX-RPC subset API, employing JSR 172 stubs and the runtime.

In devices such as cell phones, typically the application and the JSR 172 stub reside in the device's memory, while all the JSR 172 elements, along with the underlying profile and configuration, are embedded in the device itself.

The JSR 172 Runtime and Service Provider Interface:

At the center of JSR 172 operations is the runtime, with its service provider interface, which enables the stubs to perform all the tasks associated with invoking an RPC service endpoint:

- Set properties specific to an RPC invocation
- Describe the RPC invocation input and return values
- Encode input values
- Invoke the RPC service endpoint
- Decode and return to the application any values that the service endpoint returns

Figure 3 depicts the relationships between the client application, the stub, and the runtime:

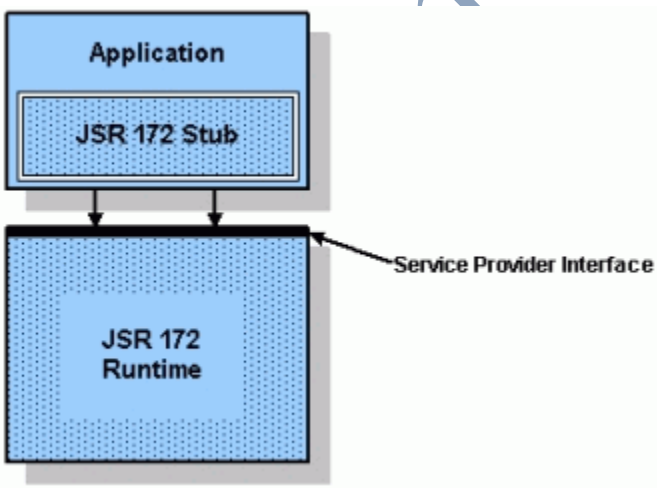


Figure 3. The JSR 172 Runtime and SPI

While the JSR 172 runtime hides complexities such as connection management and data encoding, the SPI decouples stubs from details of the runtime implementation, allowing portability of stubs between vendor's implementations.

The client application doesn't interact with the runtime and SPI directly; it uses stubs instead. The runtime and SPI are mainly concerns of third-party vendors who intend to

develop JSR 172 runtimes and automation tools, such as the WSDL-to-Java mapping tool that developers use to generate stubs.

Because application developers do not use the SPI directly, this article does not cover the runtime and SPI in more detail. You can consult the JSR 172 specification if you want more information on them.

Web Services on Android

Introduction

The Android platform is the product of the Open Handset Alliance, a group of organizations collaborating to build a better mobile phone. The group, led by Google, includes mobile operators, device handset manufacturers, component manufacturers, software solution and platform providers, and marketing companies. From a software development standpoint, Android sits smack in the middle of the open source world.

The Android platform

With Android's breadth of capabilities, it would be easy to confuse it with a desktop operating system. Android is a layered environment built upon a foundation of the Linux kernel, and it includes rich functions. The UI subsystem includes:

- Windows
- Views
- Widgets for displaying common elements such as edit boxes, lists, and drop-down lists

Android includes an embeddable browser built upon WebKit, the same open source browser engine powering the iPhone's Mobile Safari browser.
Android boasts a healthy array of connectivity options, including WiFi, Bluetooth, and wireless data over a cellular connection (for example, GPRS, EDGE, and 3G). A popular technique in Android applications is to link to Google Maps to display an address directly within an application. Support for location-based services (such as GPS) and accelerometers is also available in the Android software stack, though not all Android devices are equipped with the required hardware. There is also camera support.
Historically, two areas where mobile applications have struggled to keep pace with their desktop counterparts are graphics/media, and data storage methods. Android addresses the graphics challenge with built-in support for 2-D and 3-D graphics, including the OpenGL library. The data-storage burden is eased because the Android platform includes the popular open source SQLite database. Figure 1 shows a simplified view of the Android software layers.

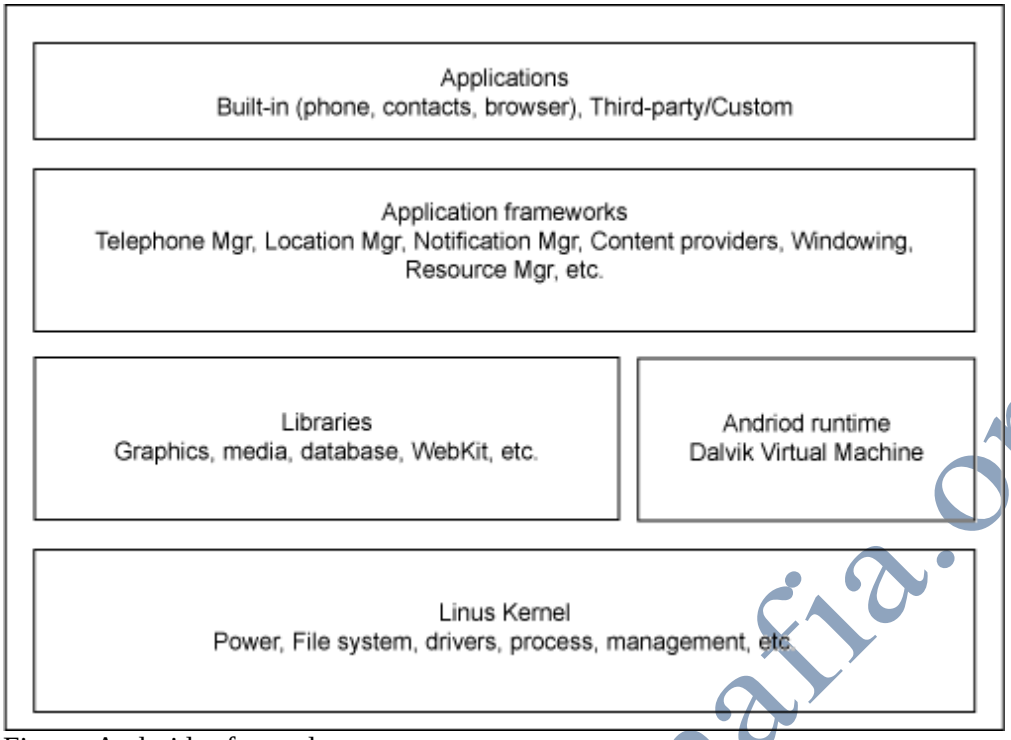


Figure: Android software layers

Application architecture

As mentioned, Android runs atop a Linux kernel. Android applications are written in the Java programming language, and they run within a virtual machine (VM). It's important to note that the VM is not a JVM as you might expect, but is the Dalvik Virtual Machine, an open source technology. Each Android application runs within an instance of the Dalvik VM, which in turn resides within a Linux-kernel managed process.

An Android application consists of one or more of the following classifications:

Activities

An application that has a visible UI is implemented with an activity. When a user selects an application from the home screen or application launcher, an activity is started.

Services

A service should be used for any application that needs to persist for a long time, such as a network monitor or update-checking application.

Content providers

You can think of content providers as a database server. A content provider's job is to manage access to persisted data, such as a SQLite database. If your application is very simple, you might not necessarily create a content provider. If you're building a larger application, or one that makes data available to multiple activities or applications, a content provider is the means of accessing your data.

Broadcast receivers

An Android application may be launched to process a element of data or respond to an event, such as the receipt of a text message.

An Android application, along with a file called AndroidManifest.xml, is deployed to a device. AndroidManifest.xml contains the necessary configuration information to properly install it to the device. It includes the required class names and types of events the application is able to process, and the required permissions the application needs to run. For example, if an application requires access to the network — to download a file, for example — this permission must be explicitly stated in the manifest file. Many applications may have this specific permission enabled. Such declarative security helps reduce the likelihood that a rogue application can cause damage on your device.
The next section discusses the development environment required to build an Android application.

Android HttpMethods

These are simple request/response through HttpRequests (f.e. for xml file). The major advantage is that it makes the service to be lightweight .

The Default Http Connection object seems to spool up and run, at least with the current version of the emulator, hideously slowly. You can probably get better performance by tinkering with the various subclasses of the HttpClient class. Your mileage may vary, but if you need a fast and easy proof-of-concept demonstration, the default one may be the way to go.

Ansynchronisation is also possible by running the Request in another thread, using an android intern message handler & the object is received from HttpResponse

Android applications often must access data that resides on the Internet, and Internet data can be structured in several different formats. In this article, you'll see how to work with two data formats within Android applications:

- JSON
- XML

First you'll develop a Web service that converts CSV data into XML & JSON. Then you'll build a sample Android application that can pull the data from the Web service in any of these formats and parse it for display to the user.

To perform this, you need the latest Android SDK and the Android 2.2 platform. The SDK requires that you also have a Java™ Development Kit (JDK) installed; one can use JDK 1.6.0_17. You don't need a physical Android device; all of the code will run on the SDK's Android emulator. You also need a Java web application server to run the Web service that the Android application uses. Alternatively, you can deploy the server-side code to the Google App Engine.

Comparative Analysis

Smart phones are cell phones with more advanced capabilities than ordinary cell phones. They have usually more processing power and larger screens than normal phones and they order PC-like functionality. Last year their prices dropped significantly which gave a boost to their market share.

A Web service is a way to call a procedure on a remote computer system with web-related standards like SOAP and HTTP. They are becoming a de facto standard for information exchange between different computer systems.

When one combined smart phones and Web services i.e. to run a Web service on a mobile device one can greatly increase the functionality of mobile devices to interact with its environment. This makes a wide range of new functionality possible, especially in the field of ubiquitous computing.

Running Web services on mobile devices opens a wide range of new possibilities and solves heterogeneity and interoperability issues. However the performance characteristics are important to ensure the mobile device is able to handle multiple requests while the device is able to function normally. We have studied the performance characteristics of Web services running on the various mobile phone platforms.

Concept

Scalability is a capability of a Web service that describes how well a Web service performs under different load conditions. Scalability is closely tight to the performance of a Web service. A Web service that serves thousands of requests is not of much use if half of them are error responses.

In order to measure the performance and scalability of a Web service we have identified some metrics that characterize Web service performance:

- Total number of Responses
- Number of successful responses and successful responses per second
- Percentage of error responses
- Average execution time: The time from the last byte sent to the first byte received in milliseconds

Implementation

To test the performance of a mobile Web service a Web service needs to be constructed. The Web service needs to do some processing in order to try to increase the test result differences. An echo service simply replies the input without any processing and therefore the CPU is busy with making and breaking up of connections. To let the CPU do some actual work the test Web service determines if an IP address is inside the Netherlands. To speed up that search process the Web service makes use of the binary search algorithm.

All the three Web services function in the same way. First the Web service listens for incoming connections. The HTTP frameworks incorporated into the different APIs support outgoing connections only. To make an incoming connection possible you have to create a TCP socket yourself and start listening on it. After an incoming connection is made another socket is created to handle this connection. First the request is parsed and adequate action is taken. Five different responses are possible:

- Get-request for geoip.wsdl. Returns the WSDL-file. That is an XML description of the Web service.
- Get-request for any other file. Returns a HTTP 404 file not found error response.
- Valid post-request to invoke the Web service. Returns a SOAP message with either true or false.
- Invalid post-request. Returns a invalid SOAP request error response.
- Invalid HTTP request. Returns a HTTP 400 bad request error response.

In the case of a valid post request the IP address is extracted from the request. The IP address is in the dot-decimal notation *(xxx.xxx.xxx.xxx)* and needs to be converted to a 32 bit unsigned integer.

Experimental Setup

To test the performance of the Web services, Para soft SOA test 6:1 and Load test 6:1 will be used. In Load test 6:1 it is possible to define an amount of thread clients that concurrently sends requests to the Web service for a certain amount of time. When the tests are done the metrics mentioned in the concept section will be displayed.

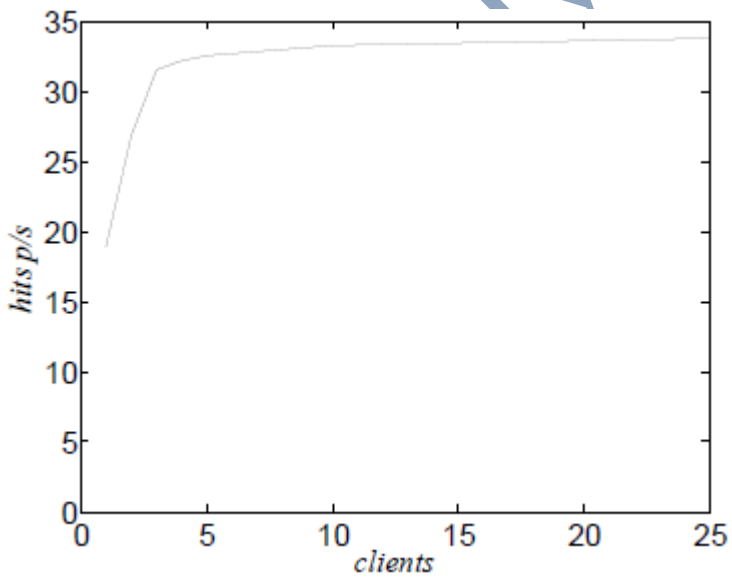
The test software and emulators will run on a modern PC running Windows XP (Core 2 Duo E8400, Intel P43 Chipset, 4GB main memory) to ensure that the PC is not a bottleneck. While the performance tests are running the CPU usage will be monitored to confirm that the host PC has enough computational power.

Experiments

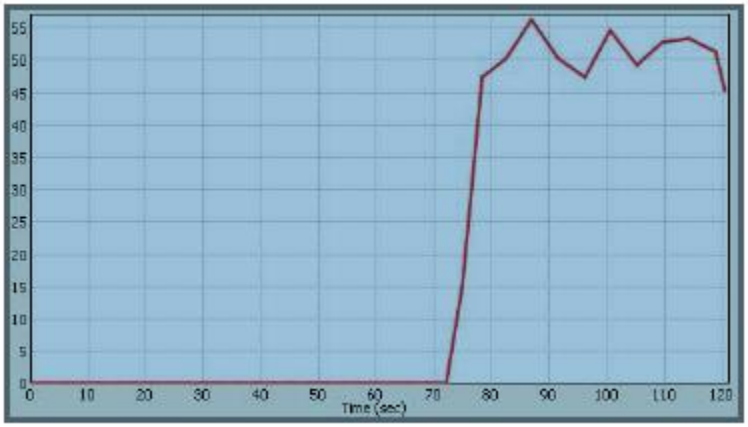
J2ME Scalability

No of Clients	Total No of Response	No of Successful Response	% of Error in Response	Avg. exec Time (ms)	Avg. Server Time (ms)
1	2277	2277	0.0%	2.0	2.0
2	3234	3234	0.0%	3.0	3.0
3	3820	3797	0.6%	4.0	4.0
4	5084	3868	23.9%	7.0	6.0
5	7804	3914	49.8%	10.0	8.0
25	12483	4057	67.5%	54.7	69.2

J2ME Hit response ratio



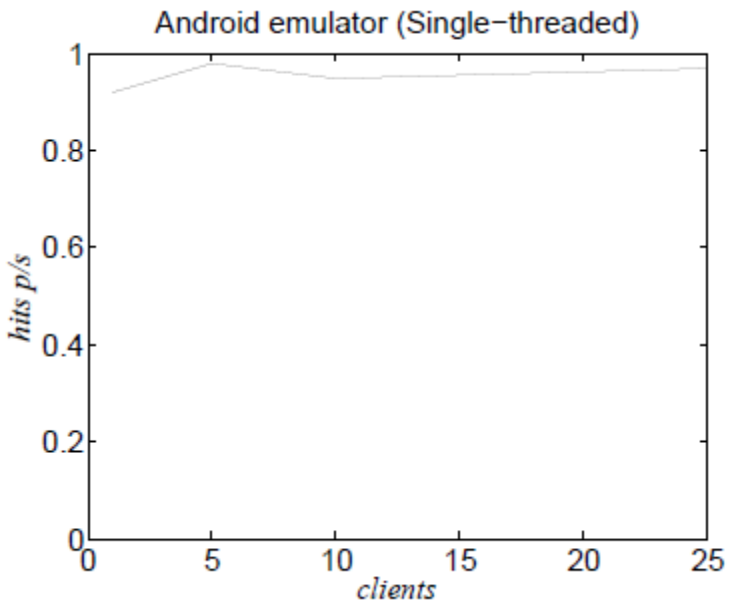
Error Rate in J2ME



Android Scalability

Android Hit response ratio

No of Clients	Total No of Response	No of Successful Response	% of Error in Response	Avg exec Time (ms)	Avg. Server Time (ms)
1	111	111	0.0%	28	27
5	122	122	0.0%	24	24
10	124	124	0.0%	30	30
25	141	141	0.0%	66	66



Over all comparision

Platform	Percentage of Error Responses (25 Clients)	Avg. Exec. Time (25 Clients)	Avg. Server Time (25 Clients)
J2ME	67.5%	54.7	69.2
Android	0%	66	66

Results

The most obvious difference is the performance of the emulators: The J2ME emulator is by far the fastest & about thirty times faster than the Android emulator. This difference in performance could be explained by the efficient low-level nature of J2ME.

A similarity between the two emulators is that they all use very little processing power of the host PC. So processing power is not the bottleneck but opening/closing connections are probably the bottleneck.

The scalability differs enormously. The Android version has a zero error rate at a load of 25 clients probably caused by the low performance. J2ME OS emulator has a very bad scalability (67.5% clients).

It is certainly possible to run a Web service on a mobile device although they are not designed for running services. The performance of the emulators is much lower than expected and the differences in performance between the 2 platforms are huge. The emulators use just a fraction of the processing power of the host pc, so making/destroying an emulated connection must take a lot of time. The Web service running on a real device performs quite well and far better than the emulators.

Security

Security in J2ME

Securing content, not connections
Although HTTPS/SSL protocols are popular and powerful, they are designed for the old wired-Internet topography. A number of serious problems can crop up when we begin to apply SSL to the new generation of dynamic wireless applications, as you can see:

- Peer groups and subscription-based multicast applications will be a major model for the smart wireless applications of the future. Being a one-to-one protocol, SSL does not support multicast applications very well.
- SSL is a point-to-point protocol that secures direct connections between hosts. The emerging Internet topography, however, is based on Web services. As such, it will require multiple intermediaries to help process and deliver XML-based service requests. The need for an end-to-end security solution is on the horizon.
- SSL indiscriminately encrypts all data with the same key strength, which can be unnecessary or even undesirable for some applications. The computational overhead for setting up and running SSL connections is simply too high for some wireless applications.

The currently emerging problems with SSL will only become more pressing as the mobile commerce network expands. To resolve these issues, we need an end-to-end security model with flexible encryption schemes to meet a range of different requirements. We need to focus on securing content rather than connections. We'll close this discussion with a look at several up-and-coming content formats, security protocols, and tools for implementing end-to-end wireless security.

Secure content through secure XML
XML is our format of choice for data communications between J2ME wireless applications and back-end services. To provide end-to-end security, we need to secure XML documents. For this, we need special XML standards to associate security meta information with individual documents.
Several XML security protocols have been proposed to support communication data security in XML applications. Among them are the following:

- **Security Assertion Markup Language (SAML)** is a protocol to transport authentication and authorization information in an XML message. It could be used to provide single sign-on Web services.
- **XML digital signatures** define how to digitally sign part or all of an XML document to guarantee data integrity. The public key distributed with XML digital signatures can be wrapped in XML Key Management Specification (XKMS) formats.
- **XML encryption** allows applications to encrypt part or all of an XML document using references to pre-agreed symmetric keys.
- The **Web Services secure XML protocol family (WS-Security)**, endorsed by IBM and Microsoft, is a complete solution to provide security to Web services. It is based on XML digital signatures, XML encryption, and an authentication and authorization scheme similiar to SAML.

All of the above security protocols can bind to Web services messaging protocols. For example, we can embed a SAML segment in a SOAP message header to authenticate and authorize the access to the requested services. We can also embed an XML Digital Signature segment in a SOAP header to authenticate a credit card number in that message.

Security on Android

The most proposed solution for security is the usage of HTTP secure protocols (e.g. HTTPS;authorization and authentication mechanisms) that offer IP-to-IP secure solutions and not application-to-application ones. However, the fact that ROA and REST in particular are relying on HTTP is not preventing the usage of other protocols at application instead of at the communication infrastructure level. Authorization in ROA is typically done on the HTTP protocol level. By using the common HTTP interface and it's GET, POST, PUT, DELETE operations, it is fairly easy to restrict e.g. DELETE-operations in general. But especially ROA implementations that are not strictly following the ROA principles are a potential security leak. E.g. if the GET operation with a query-string like "method=delete" is used instead of HTTP/DELETE.

Besides the canonical security issues, the peculiarity of ROA where new resources are dynamically created raises an additional problem that is defining who is responsible for assigning security policies to the created resources (the one willing to create the resource, the resource used to create the new one) that become themselves directly accessible over the network. This issue could be however coped by applying some secure policies based on addressing schema but probably this would not allow, for instance, determining the exact access rights of created resources using creator identity.

Wrap up

Although the terrain is complex, we conclude that J2ME does indeed offer many security advantages, over Android alternatives. At the high end, the CDC's Personal Profile brings the full power of J2SE to wireless applications, including, in most cases, the ability to implement sophisticated applications with strict security requirements.
The current MIDP 1.0 specification does not provide enough in the way of security APIs. But with the help of third-party libraries and vendor support, we have found that we can use MIDP 1.0 to implement secure applications. Furthermore, and as we have shown, the upcoming MIDP 2.0 spec promises to provide a set of core security APIs that should allow us to create mobile commerce applications with more advanced and flexible security features.

Maps & Location Based Services

J2ME

Microsoft's MapPoint Web Service is a managed solution accessible via a set of platform-independent SOAP Web Services APIs. MapPoint also allows users to import their own geographical data and points-of-interest listings for customized services.

MapPoint Web Services

MapPoint Web Services expose a very rich set of SOAP APIs. Important remote methods in MapPoint v3.0 are divided into four categories.

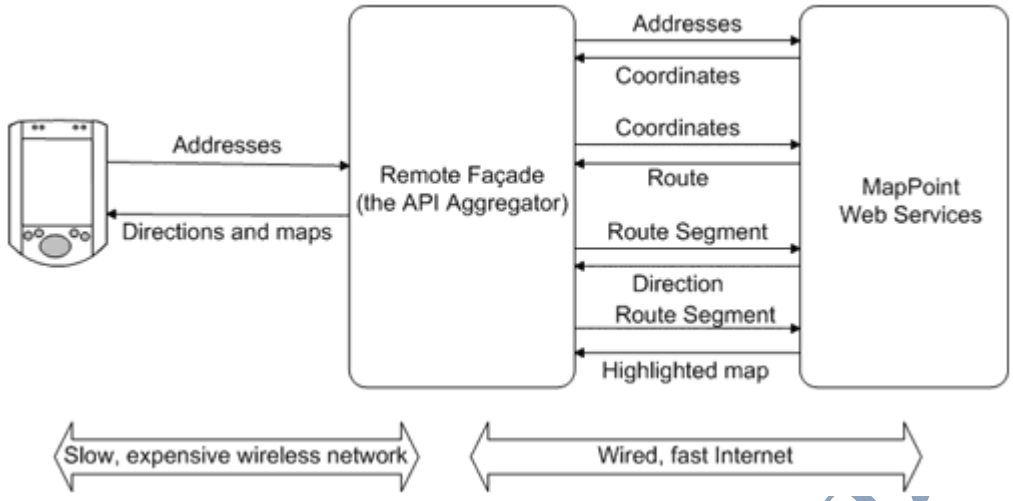
- Common Service contains utility functions that are common to other services.
- Find Service allows the user to do generic geocoding tasks, including finding addresses from latitudes and longitudes, and vice versa. It also allows searching for nearby points of interests.
- Route Service calculates routes and directions based on locations and waypoints. It also generates map view representations of the calculated routes. The map view can be used to highlight routes on a map.
- Render Service renders the map for the locations or routes. It highlights routes, places pushpins and icons, and supports zoom.

The Aggregated API

Through the SOAP API, MapPoint Web Service offers fine-grained access to its GIS back end, allowing us to design flexible applications without arbitrary limitations imposed by preset scripts. However, the trade-off is that we have to make separate method calls for each small step to complete a task. For example, in order to retrieve driving directions between two addresses, we have to go through the following steps.

- Convert the addresses to latitude and longitude coordinates.
- Calculate the route according to options.
- Render the overview map with the route and start and end locations highlighted.
- Retrieve turn-by-turn instructions for each route segment.
- Render highlighted turn-by-turn maps for each route segment.

The last two steps need to be performed repeatedly for a long route with many segments. Since all those method calls are remote SOAP calls, those excessive round trips cause long delays over slow wireless networks. To minimize the data transfer over wireless networks, we can place a remote facade between the mobile client and MapPoint. The facade takes in two human-readable addresses in a single SOAP call, queries MapPoint via many API calls over the wired Internet, constructs the resultant driving directions, and returns the results. It provides a coarse-grained, simple interface that aggregates functions of the fine-grained MapPoint API methods for specific applications. Figure 18.1 illustrates the facade architecture.



Activities

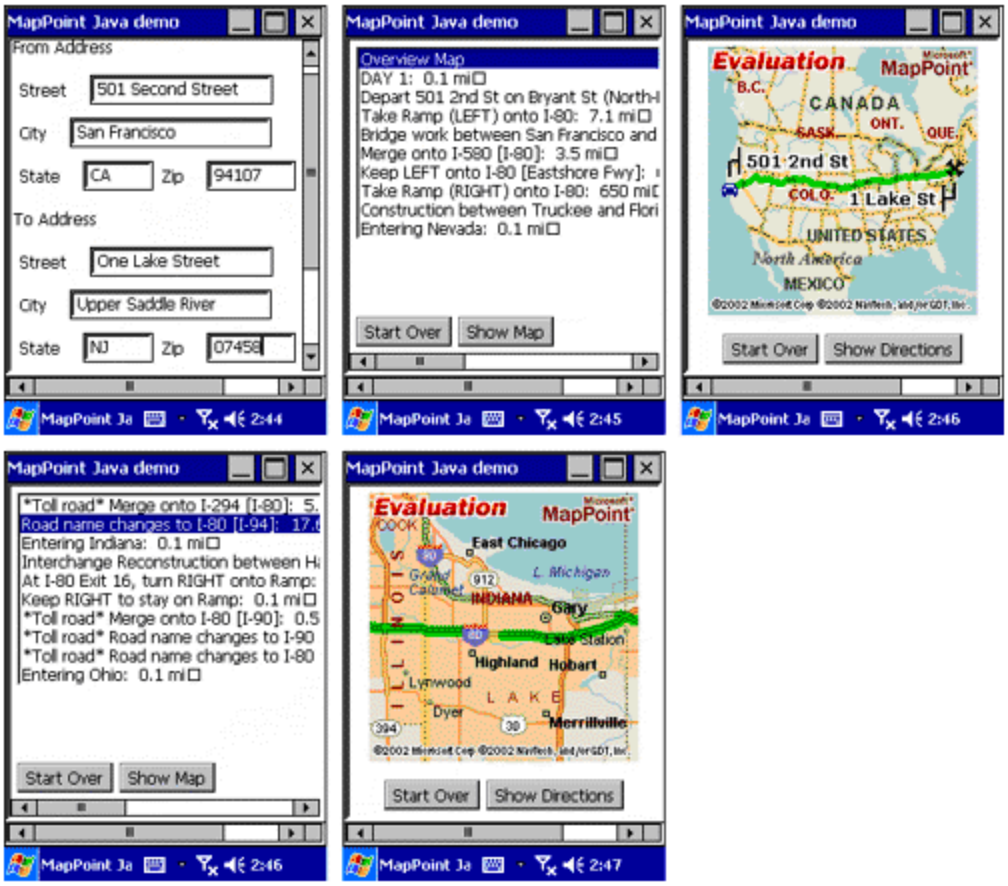
An application that has a visible UI is implemented with an activity. When a user selects an application from the home screen or application launcher, an activity is started.

Services

A service should be used for any application that needs to persist for a long time, such as a network monitor or update-checking application.

Content providers

You can think of content providers as a database server. A content provider's job is to manage access to persisted data, such as a SQLite database. If your application is very simple, you might not necessarily create a content provider



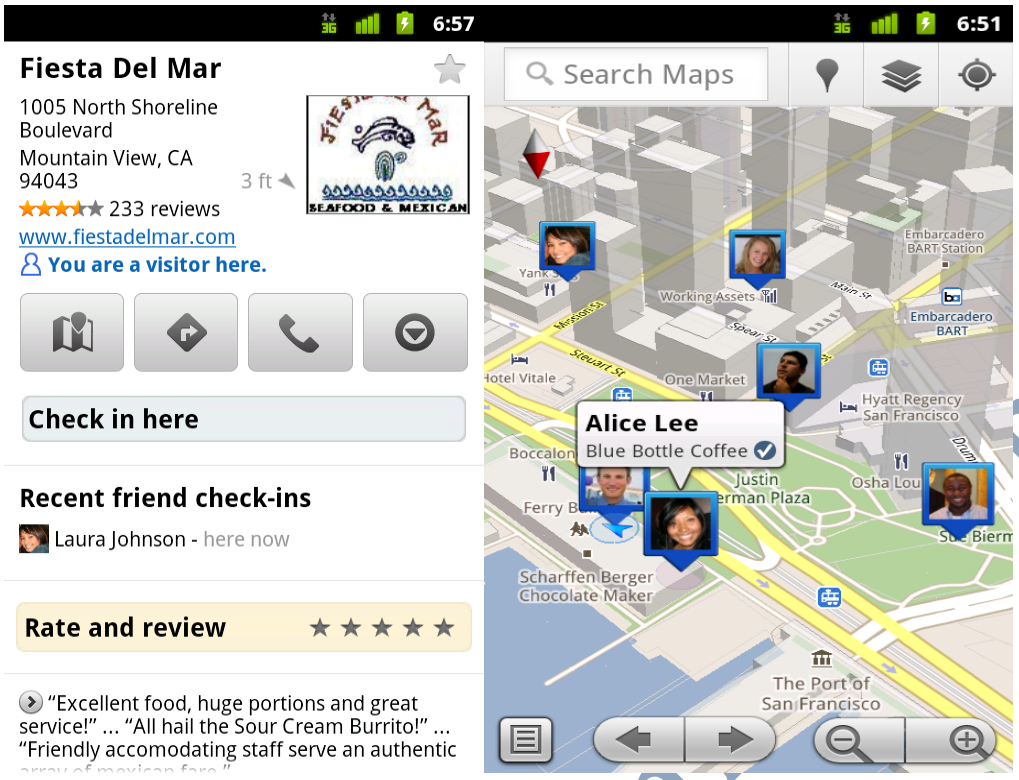
J2ME Client

Android

To make it easier for you to add powerful mapping capabilities to your application, Google provides a Maps external library that includes the com.google.android.maps package. The classes of the com.google.android.maps package offer built-in downloading, rendering, and caching of Maps tiles, as well as a variety of display options and controls.

The key class in the Maps package is com.google.android.maps.MapView. A MapView displays a map with data obtained from the Google Maps service. When the MapView has focus, it will capture keypresses and touch gestures to pan and zoom the map automatically, including handling network requests for additional maps tiles. It also provides all of the UI elements necessary for users to control the map. Your application can also use MapView class methods to control the MapView programmatically and draw a number of Overlay types on top of the map.

In general, the MapView class provides a wrapper around the Google Maps API that lets your application manipulate Google Maps data through class methods, and it lets you work with Maps data as you would other types of Views.



The Google Maps API provides these web services as an interface for requesting Maps API data from external services and using them within your Maps applications. These services are designed to be used in conjunction with a map, as per the Maps API Terms of Service License Restrictions.
These web services use HTTP requests to specific URLs, passing URL parameters as arguments to the services. Generally, these services return data in the HTTP request as either JSON or XML for parsing and/or processing by your application.
A typical Web Service request is generally of the following form:

http://maps.googleapis.com/maps/api/service/output?parameters

where service indicates the particular service requested and output indicates the response format (usually json or xml.
Full documentation of each service is contained within the particular developer guides for those services. However, this guide serves to hold some common practices useful for setting up your web service requests and processing your web service responses.

Discussing mapping without mentioning GPS is nearly impossible now because GPS has become one of the indispensable features most people would want from their mobile devices. The packageandroid.location is included for GPS support. LocationManager is the most important API; it provides access to the system location services. The mapping and GPS APIs are the essential elements for building location-based services (LBS).

The Google Web Services provide responses which are easy to understand, but not exactly user friendly. When performing a query, rather than display a set of data, you probably want to extract a few specific values. Generally, you will want to parse responses from the web service and extract only those values which interest you.
The parsing scheme you use depends on whether you are returning output in XML or JSON. JSON responses, being already in the form of Javascript objects, may be processed within Javascript itself on the client; XML responses should be processed using an XML processor and an XML query language to address elements within the XML format. We can use XPath, as it is commonly supported in XML processing libraries.

The Google Elevation API

The Google Elevation API provides you a simple interface to query locations on the earth for elevation data. Additionally, you may request sampled elevation data along paths, allowing you to calculate elevation changes along routes.

Use of the Google Elevation API requires that you indicate whether your application is using a "sensor" (such as a GPS locator) to determine the user's location. This is especially important for mobile devices.

Applications that determine the user's location via a sensor must pass sensor=true within your Elevation API request URL. If your application does not use a sensor, pass sensor=false.

Wrap Up

Though the J2ME uses Microsoft Midpoint Service which is powerful for using the a cross platform application based on maps & user locations still it lacks new & rich features like zooming of the map, the street view & also results into high dependence of J2ME on a third party service.

Most of the services used in J2ME are designed for normal location information. New applications coming up on Android do a lot with these location based web services like allowing you to keep a track of your friends group, suggesting a near by restaurant depending on your preferences, allowing you to know your elevation from the ground, providing a clear street view for any of your pre-decided path, providing you path for a certain location & a lot more.

Looking at all these rich features for locations & map based services & high integration with Google maps, puts Android ahead of J2ME when it comes to such services.

Conclusion

- **The J2ME web services have to depend on third party apis for integration with maps, mail, videos & etc..**
- **Though these services have better performance, still we need to understand that these services were designed while keeping feature phones in mind.**
- **The J2ME service market is quite large, but Android service market is rising fast (for smart phones) a segment which has become more and more popular in the recent years.**
- **It depends on the requirements that which kind of technology we select for creating & hosting our services, who are the end consumers & what level of performance is required from these services.**
- **Until wireless networks are as reliable, cost-efficient and fast as LAN networks, mobile applications still need to be a blend of online and offline models.**

Reference

www.google.com
www.wikipedia.org
www.studymafia.org
www.projectsreports.org