


Programación de Dispositivos Móviles




Sesión 20:
Aplicaciones corporativas

Java y Dispositivos Móviles

© 2007-2009 Depto. Ciencia Computación e IA

Aplicaciones corporativas-1

Índice




- Front-end de aplicaciones corporativas
- Integración con aplicaciones corporativas
- Arquitectura MVC
- Modo sin conexión

Java y Dispositivos Móviles

© 2007-2009 Depto. Ciencia Computación e IA

Aplicaciones corporativas-2

Aplicaciones corporativas



- Front-end de aplicaciones corporativas
- Integración con aplicaciones corporativas
- Arquitectura MVC
- Modo sin conexión

Java y Dispositivos Móviles

© 2007-2009 Depto. Ciencia Computación e IA

Aplicaciones corporativas-3

Front-ends



- En un PC normalmente accedemos a las aplicaciones web mediante un navegador, a través de una interfaz HTML
- En dispositivos móviles podemos utilizar un paradigma similar, con lenguajes como WML o cHTML
- Sin embargo, la utilización de aplicaciones J2ME aporta las siguientes ventajas:
 - Interfaz de usuario flexible
 - Permiten trabajar sin conexión
 - Se conectan mediante protocolo HTTP estándar, no necesitaremos conocer el tipo de red subyacente

Optimizaciones



- Reducir el tráfico en la red
 - Validar datos en el cliente
 - Mantener copias de los datos en local (RMS)
- Operaciones de larga duración
 - Accesos a RMS, conexiones de red
 - Realizar siempre desde un hilo
 - Proporcionar información al usuario sobre el progreso
 - Permitir interrumpir si es posible
- Personalización
 - Guardar las preferencias del usuario en el móvil
 - Recordar login y password para futuras sesiones

Aplicaciones corporativas

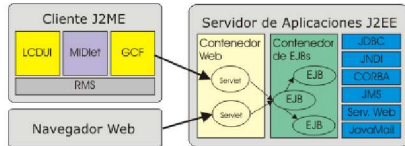


- Front-end de aplicaciones corporativas
- Integración con aplicaciones corporativas
- Arquitectura MVC
- Modo sin conexión

Comunicación con el servidor



- El MIDlet cliente utilizará:
 - GCF para comunicarse con el servidor web
 - LCDUI para la interfaz con el usuario
 - RMS para almacenar datos de forma local en el móvil
- En la aplicación web J2EE utilizaremos:
 - Un servlet que se comuniquen con el cliente J2ME
 - Podemos definir otro servlet para acceder mediante una interfaz web
 - Podemos reutilizar desde ambos servlets la misma lógica de negocio implementada mediante EJBs



Codificación de los datos



- En la comunicación con el servidor (Servlet) se debe acordar una codificación de los mensajes que ambos entiendan.
- **Binario**
 - Mensajes compactos y fáciles de analizar.
 - Alto acoplamiento.
 - Podemos utilizar la serialización de objetos definida en MIDP
 - Asegurarse de que el objeto es compatible con J2ME y J2EE
 - Tanto en el cliente como en el servidor se deberán utilizar los mismos métodos de serialización
- **XML**
 - Mensajes extensos y complejos de analizar por un móvil.
 - Bajo acoplamiento.

Mantenimiento de sesiones



- Las sesiones normalmente se mantienen con elementos que gestionan los navegadores web como las cookies
- Para poder utilizar sesiones deberemos implementar en nuestro cliente alguno de los métodos existentes
 - Cookies
 - Reescritura de URLs
- Las cookies en algunos casos son filtradas por gateways
 - Será más conveniente utilizar reescritura de URLs

Reescritura de URLs



- En el lado del servidor debemos obtener la URL reescrita

```
String url_con_ID = response.encodeURL(url);
```

- Se adjunta un identificador a dicha URL que identifica la sesión en la que nos encontramos
- Devolvermos la URL al cliente
 - Por ejemplo, mediante una cabecera HTTP

```
response.setHeader("URL-Reescrita", url_con_ID);
```

- La próxima vez que nos conectemos al servidor deberemos utilizar la URL reescrita
 - De esta forma el servidor sabrá que la petición la realiza el mismo cliente y podrá mantener la sesión

Aplicaciones corporativas



- Front-end de aplicaciones corporativas
- Integración con aplicaciones corporativas
- Arquitectura MVC
- Modo sin conexión

MVC para aplicaciones J2ME



- Podemos aplicar el patrón de diseño Modelo-Vista-Controlador a las aplicaciones J2ME
- En esta arquitectura distinguimos:
 - Modelo
 - Datos de la aplicación
 - Vista
 - Presentación de la aplicación
 - Pantallas de nuestro MIDlet
 - Controlador
 - Controla el flujo de la aplicación
 - Decide qué pantalla mostrar y qué operaciones realizar en cada momento

Modelo



- **Tenemos aislados los datos del resto de la aplicación**
 - Nos facilitará implementar la posibilidad de trabajar en modo *online* y modo *offline*
- **Podemos dividir el modelo en dos subsistemas:**
 - **Modelo local**
 - Accede a los datos almacenados localmente para trabajar *offline*
 - Puede utilizar un adaptador RMS para acceder a estos datos
 - **Modelo remoto**
 - Podemos definir un proxy para acceder al servidor
 - El proxy encapsula la conexión con el servidor para acceder a sus funcionalidades, proporcionándonos una interfaz local
- **Podemos utilizar el patrón de diseño fachada para integrar estos dos subsistemas**
 - Proporcionamos una interfaz única que nos dé acceso a ellos
 - Reduce la complejidad subyacente, aísla al resto de la aplicación

Aplicaciones corporativas



- Front-end de aplicaciones corporativas
- Integración con aplicaciones corporativas
- Arquitectura MVC
- Modo sin conexión

Tipos de aplicaciones



- **Según la forma de conectarse, podemos distinguir varios tipos de aplicaciones:**
 - *Thin*
 - Todo el procesamiento se realiza en el servidor
 - Este tipo de aplicaciones son por ejemplo a las que accedemos mediante un navegador
 - Siempre necesitamos conexión para acceder a ellas
 - *Thick*
 - Aplicaciones dedicadas
 - Se instalan en el cliente para realizar una tarea concreta
 - Necesitan trabajar de forma coordinada con el servidor
 - *Standalone*
 - Todo el procesamiento se realiza en el cliente
 - Por ejemplo calculadora, bloc de notas, juegos, etc
 - Pueden conectarse eventualmente para actualizar datos, normalmente a petición del usuario

Replica de datos



- Vamos a centrarnos en las aplicaciones *thick*
- Para permitir que estas aplicaciones trabajen sin conexión deberemos replicar los datos del servidor
 - Mantendremos una copia local de los datos
- El modelo de réplica se caracteriza por
 - ¿Se replican todos los datos o sólo una parte de ellos?
 - ¿Las estructuras de datos se replican fielmente o no?
 - ¿Los datos son de lectura/escritura o de sólo lectura?
 - ¿Los mismos datos pueden ser compartidos y replicados por muchos usuarios?
 - ¿Los datos tienen fecha de caducidad?

Sincronización de datos



- Los datos en el cliente y en el servidor deberán ser consistentes
 - Deberemos sincronizar los datos para que los cambios hechos en cliente o servidor se actualicen en el otro lado
- Podemos distinguir tres formas de envío de datos:
 - El cliente descarga datos del servidor
 - Mantenemos una caché de datos
 - El cliente envía datos no compartidos al servidor
 - El cliente envía datos compartidos con otros usuarios al servidor
 - Este es el caso más problemático
 - Varios clientes pueden modificar sus copias locales concurrentemente y causar conflictos en la actualización de datos

Caché de datos



- Debemos decidir cuando actualizar la caché
 - Si conocemos la fecha de caducidad podemos utilizar esta información
 - Si no la conocemos podemos conectar periódicamente al servidor o a petición del usuario
- Podemos utilizar *timestamps* para conocer qué datos no se han descargado todavía
 - A cada dato que se añada en el servidor se le asignará un *timestamp* superior al del anterior dato
 - El cliente conocerá el *timestamp* del último dato descargado
 - Cuando solicite datos al servidor, enviará este *timestamp* para que el servidor nos devuelva todos los datos posteriores
 - Recibiremos el servidor el *timestamp* correspondiente al último dato devuelto actualmente

Enviar datos al servidor



- Si modificamos o creamos datos en el cliente deberemos actualizar los cambios en el servidor
- Podemos añadir a los datos almacenados localmente un *flag* que indique si el dato está pendiente de ser actualizado en el servidor
- Será conveniente que la granularidad de los datos sea lo más fina posible
 - Almacenar menor cantidad de datos juntos en un mismo registro
 - De esta forma actualizaremos sólo la porción modificada, y no toda la estructura
