

Acceso Dinámico a Servicios de una Infraestructura Web desde Teléfonos Móviles

Elena Sánchez Nielsen
Dpto. de E.I.O. y Computación
Universidad de La Laguna
38271 S/C de Tenerife
enielsen@ull.es

Sandra Martín Ruiz
ETS Ingeniería Informática
Universidad de La Laguna
38271 S/C de Tenerife

Jorge Rodríguez Pedrianes
ETS Ingeniería Informática
Universidad de La Laguna
38271 S/C de Tenerife

Resumen

Las tecnologías móviles están revolucionando la manera en que las personas interactúan en su vida diaria, trabajo y negocio. El acceso dinámico a nuevos servicios ofertados desde una infraestructura Web en tiempo real desde los teléfonos móviles abre nuevas perspectivas de negocio así como nuevos retos en su implementación. Actualmente, el paradigma de la arquitectura orientada a servicios es una de las aproximaciones más prometedoras para llevarlo a cabo. Sin embargo, su implementación mediante la utilización de una plataforma J2ME presenta diferentes limitaciones que pueden ser resueltas actualmente mediante la utilización de un componente proxy. En este artículo, abordamos los problemas a resolver, la arquitectura propuesta, la descripción de este componente, las diferentes formas de implementarlo, el coste efectivo de su desarrollo así como los resultados experimentales obtenidos.

1. Motivación

La evolución de las Tecnologías de la Información y en particular de las redes de comunicaciones y dispositivos móviles ofrece nuevas oportunidades para que la información esté disponible para cualquier persona, en cualquier lugar y en cualquier instante de tiempo. Los dispositivos móviles, tales como pocket pc's, dispositivos wireless y teléfonos móviles se están utilizando cada vez más como herramientas personales y comerciales. Todo ello, se traduce en nuevas perspectivas de negocio, lo cual implica el desarrollo de nuevos servicios así como infraestructuras capaces de proveer dichos servicios.

En adición a las dificultades habituales de diseñar y desarrollar nuevos sistemas con tecnologías para entornos móviles tales como la capacidad de cómputo, memoria, espacio de pantalla, ancho de banda, también existen otras limitaciones fundamentales basadas en la

propuesta de arquitecturas que permitan el acceso y gestión dinámica de servicios.

Actualmente, el paradigma de arquitecturas orientadas a servicios es considerado como una de las aproximaciones más prometedoras para que los dispositivos móviles puedan acceder a nuevos servicios y/o funcionalidades de aplicaciones expuestas como servicios a través de la red (Internet/Intranet). Sin embargo, una aproximación basada en una provisión estática de servicios no constituye actualmente una solución admisible, debido a que no puede gestionar de forma eficiente las necesidades de los usuarios móviles que desean decidir dinámicamente sobre qué servicios les gustaría consumir en función de sus preferencias y contexto. Por ejemplo, un usuario entrando en un aeropuerto podría estar interesado en recibir información acerca de promociones y ofertas de productos del aeropuerto, información sobre su vuelo, etc., sin la necesidad de actualizar su aplicación mediante la descarga de la nueva funcionalidad solicitada. Es decir, a dicho usuario le gustaría poder consultar en tiempo real la oferta de nuevos servicios disponibles en el lugar que se encuentra (en este caso, recinto del aeropuerto) de forma sencilla y fácil, que no implique la modificación de la forma de consulta en que accede a la oferta de servicios disponibles.

El objetivo de este artículo, es abordar los problemas que se presentan en la propuesta de una solución plausible mediante la utilización de la plataforma J2ME que permita proveer una arquitectura dinámica de servicios para facilitar la interacción a servicios diferentes sin la necesidad de conocer a priori la ubicación de dichos servicios y sin la necesidad de actualizar la aplicación del usuario móvil. Fundamentalmente, la utilización de una plataforma J2ME para el desarrollo de aplicaciones móviles plantea el cómo resolver: (1) el descubrimiento e invocación dinámica desde el dispositivo móvil, (2) la utilización del registro UDDI y (3) soporte de parámetros de tipo complejo cuando el

mecanismo de invocación dinámica es utilizada. Actualmente, las aproximaciones basadas en la utilización de un componente proxy permiten ofrecer las mejores soluciones [1], [2]. Tres requisitos fundamentales necesitan ser especificados a la hora de introducir un proxy en una arquitectura orientada a servicios: (1) qué componentes pueden representar un proxy, (2) cómo establecer la conexión con el proxy y (3) como enviar los parámetros de la operación invocada del servicio correspondiente desde el proxy hasta el teléfono móvil.

El resto del artículo se estructura de la siguiente manera: la sección 2 describe los conceptos necesarios para la comprensión del artículo. La sección 3 aborda la arquitectura propuesta y las dos aproximaciones de implementación del proxy. La sección 4 muestra los resultados experimentales obtenidos mediante las diferentes formas de llevar a cabo la implementación del proxy. La discusión sobre las ventajas, desventajas y rendimiento obtenido con la plataforma J2ME se detalla en la sección 5.

2. Background

Esta sección describe una visión general sobre la arquitectura orientada a servicios y las limitaciones actuales en el diseño de aplicaciones clientes, cuando los teléfonos móviles actúan como demandantes de servicios Web y una plataforma J2ME es utilizada.

2.1. Arquitectura orientada a Servicios

Una arquitectura orientada a servicios (SOA) es una arquitectura contractual que ofrece y consume software como servicios. Tres entidades básicas conforman SOA: (1) proveedores de servicios, (2) demandantes de servicios (conocidos también como consumidores de servicios) y (3) registro de servicios. Los proveedores de servicios son los que ofrecen los servicios, definiendo las descripciones de sus servicios y publicándolas en un registro de servicio. Los demandantes de servicios localizan los servicios de interés en dicho registro, con el fin de invocar el servicio correspondiente. Actualmente, la tecnología de los servicios Web implementa SOA mediante la utilización de estándares XML. Básicamente, tres iniciativas basadas en XML son utilizadas para soportar el modelo de interacción entre los diferentes elementos: (1) SOAP (una forma para comunicarse), (2) WSDL (una forma para describir los servicios) y (3) UDDI (una forma para registrar los servicios).

Mediante la utilización de SOAP, los servicios intercambian mensajes, transformando la

invocación a un servicio en un mensaje XML, intercambiando el mensaje y transformando de nuevo en el destino el mensaje XML en la invocación del servicio. Mediante la utilización de WSDL, el desarrollador especifica la interfaz de programación de un servicio Web. Esta interfaz especifica los métodos soportados por el servicio Web, donde cada método puede recibir un mensaje de entrada y retornar otro mensaje como salida. Cuatro tipos de mensajes SOAP son posibles: RPC/encoded, RPC/literal, document/literal y document/encoded.

Utilizando el paradigma de servicios Web, una aplicación cliente (consumidor de servicios) realiza una llamada a un procedimiento de un servicio Web (proveedor de servicio) en la misma manera en que invoca una llamada local. Hay tres tipos de comunicación entre una aplicación cliente y un servicio Web:

- **Stub estático:** en este caso, una llamada a un procedimiento de una aplicación cliente se traduce a una invocación a un procedimiento localizado en el stub ubicado con la aplicación cliente en tiempo de compilación. De esta forma, la aplicación cliente invoca los métodos de un servicio Web directamente a través del stub. La principal ventaja de este modelo radica en la sencillez y facilidad de codificación. Sin embargo, el cambio más insignificante de la definición del servicio Web, ocasiona que el stub no pueda volver a ser reutilizable y la generación de un nuevo stub. Por tanto, la utilización de una aproximación basada en stub estáticos es sólo adecuada en contextos estáticos, donde los proveedores de servicios no desean ofertar nuevos servicios y/o actualizar los servicios que se prestan.
- **Proxy dinámico:** en este caso, la aplicación cliente invoca a un procedimiento remoto a través de un proxy dinámico que es creado en tiempo de ejecución. Este proxy necesita ser re-instanciado siempre que se modifique el endpoint de la interfaz del servicio.
- **Interfaz de invocación dinámica (DII):** esta aproximación permite realizar invocación dinámica de servicios Web, sin tener conocimientos previos de los detalles de implementación de la interfaz en tiempo de compilación. De esta manera, una aplicación cliente podría solicitar en tiempo real un servicio sobre el que nunca ha oído. Es decir, una aplicación podría invocar un servicio sobre el que no tiene conocimiento en tiempo de ejecución, descargando dinámicamente el fichero WSDL apropiado, realizando el análisis correspondiente y construyendo los elementos requeridos para la utilización del

servicio. Sin embargo, la utilización de esta aproximación requiere una mayor complejidad desde el punto de vista del programador.

2.2. Infraestructura J2ME

La plataforma Java, Micro Edition (Java ME) proporciona un entorno para el desarrollo de aplicaciones que se ejecutan sobre dispositivos móviles, tales como los teléfonos móviles. Esta plataforma se configura en perfiles, configuraciones y paquetes opcionales. El entorno requerido para desarrollar aplicaciones móviles clientes se compone de: KVM como máquina virtual, MIDP como perfil y CLDC como configuración. Asimismo, la especificación JSR-172 define una serie de APIs que permiten a los dispositivos móviles mediante aplicaciones J2ME acceder a servicios Web a través de protocolos SOAP y XML.

Actualmente, el diseño de aplicaciones clientes utilizando la especificación JSR-172 presenta las siguientes limitaciones:

- No hay soporte para proxies dinámicos ni para la interfaz de invocación dinámica. Es decir, el subconjunto Java ME únicamente soporta el modelo de stubs estáticos. Como consecuencia de ello, se requiere generar y añadir a la aplicación cliente tantos stubs como servicios diferentes se deseen proporcionar a un usuario móvil.
- Únicamente hay soporte para el tipo de mensaje document/literal.
- No hay soporte para registro de servicios y descubrimiento ni se soporta la especificación UDDI 2.0.
- JAX-RPC para Java ME no soporta todos los tipos básicos del JAX-RPC 1.1. Por ejemplo, actualmente, sólo existe soporte parcial para el manejo de parámetros de tipo complejo.

3. Arquitectura dinámica de servicios para teléfonos móviles

Las arquitecturas orientadas a servicios representan actualmente uno de los paradigmas fundamentales en el desarrollo e implementación de un sistema o entidad federada que desee ofertar servicios a usuarios móviles. Actualmente, las arquitecturas orientadas a servicios tradicionales se basan en un modelo de stub estático como mecanismo de comunicación entre la aplicación cliente y los proveedores de servicios. Sin embargo, este modelo no es adecuado para ofertar servicios dinámicos a usuarios móviles, debido a que: (1) cada servicio necesita ser codificado en la aplicación cliente asumiendo un conocimiento

detallado de cada servicio que será invocado en tiempo de ejecución y (2) se requiere un stub para cada servicio, el cuál necesita ser añadido a la aplicación cliente, lo cuál implica un mayor consumo del recurso de memoria. Esta situación implica que todos los servicios necesitan ser descritos (operaciones, parámetros...) en tiempo de diseño y que no pueden ser añadidos nuevos servicios en tiempo de ejecución, sin descargar una nueva versión para la aplicación cliente. Todo ello conlleva a que los proveedores de servicios no puedan crear, actualizar y cambiar servicios en cualquier instante de tiempo y que los usuarios finales sólo pueden acceder a servicios predefinidos en sus dispositivos. Teniendo en cuenta este background, nuestro objetivo se basa en proponer una arquitectura dinámica que utilizando la tecnología actual y J2ME permita proporcionar: (1) acceso automatizado desde teléfonos móviles a servicios de la Web sin conocimiento a priori de dichos servicios, (2) no actualización de las aplicaciones de los usuarios, cuando se añadan nuevos servicios, (3) respuestas de tiempo apropiadas y (4) permitir que los servicios puedan ser publicados en UDDI y accedidos de forma dinámica desde cualquier dispositivo móvil en tiempo real.

Con el fin de poder proporcionar una solución plausible a los objetivos planteados, proponemos la introducción de un nuevo componente intermediario entre los proveedores de servicios y clientes. Esta entidad intermedia está constituida por un proxy que opera como cliente DDI (descubrimiento e invocación dinámica) de los servicios ofertados por los diferentes proveedores de servicios sobre la red y como servidor de servicios a los dispositivos móviles. A este proxy, le denominamos *manejador de servicios*. Con esta aproximación, delegamos la lógica de negocio a los manejadores de servicios, proporcionando una solución admisible a las limitaciones que se presentan cuando se ha de acceder desde un dispositivo móvil a servicios Web (ver 2.2). La arquitectura propuesta se muestra en la figura 1, la cuál está compuesta de: proveedores de servicios, demandantes de servicios (usuarios móviles), registro UDDI, y manejadores de servicios. Uno o varios manejadores de servicios pueden ser provistos en función si se desea una aproximación centralizada o distribuida. Las aplicaciones clientes que residen en los dispositivos móviles únicamente interactúan con el manejador de servicio, lo cuál hace necesario la generación de un único stub que corresponde al manejador de servicio y no múltiples stubs correspondiendo a los diferentes servicios disponibles. Asimismo se reduce el número de interacciones entre el dispositivo móvil y la red. Las características fundamentales de la arquitectura propuesta son:

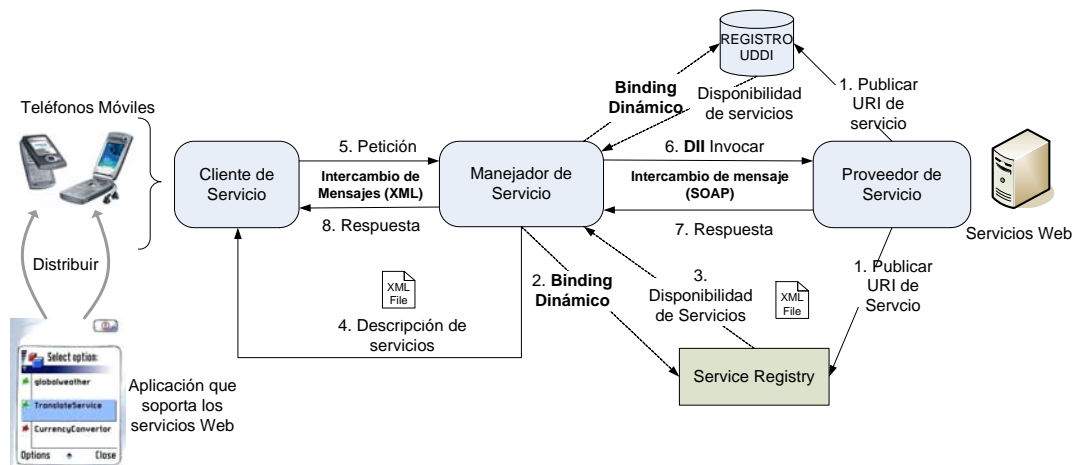


Figura 1. Arquitectura dinámica de servicios

- La utilización de un registro de servicios. Ese registro permite a los proveedores de servicios almacenar su propia lista de direcciones URL de servicios disponibles en cualquier instante de tiempo. Este servicio se utiliza fundamentalmente, porque actualmente la utilización del registro UDDI proporciona tiempos elevados de respuesta así como gran parte de los servicios no están correctamente publicados. No obstante, también es posible la consulta a UDDI sobre servicios de interés. En esta situación, el manejador de servicios es el responsable de la invocación dinámica así como de la verificación del correcto funcionamiento de los servicios publicados. Sin embargo, estas comprobaciones aumentan el tiempo de respuesta a los usuarios móviles.
- Una infraestructura basada en XML con dos propósitos: (i) definir una estructura de servicios que permita a los proveedores de servicios crear, actualizar y cambiar servicios en cualquier instante de tiempo y (ii) establecer la comunicación entre el manejador de servicio y los dispositivos móviles. Esta infraestructura ha sido descrita en [1].

A continuación se describe el componente fundamental de la arquitectura propuesta y el flujo de interacciones entre dispositivos móviles y manejadores de servicios.

3.1. Interacciones

Las interacciones entre proveedores de servicios y dispositivos móviles utilizando un manejador de servicio constan de los siguientes procesos:

- **Inicialización:** el manejador de servicio procesa un registro de servicios, cuando es inicializado. Este registro es una estructura que permite a los proveedores de servicios almacenar la lista de direcciones URL (URI) de servicios a ofrecer. URIs nuevas pueden ser añadidas en cualquier momento. El manejador de servicio mantiene una estructura dinámica basada en XML como registro. El manejador de servicio utiliza la interfaz de invocación dinámica con la finalidad de obtener descripciones de los servicios en tiempo real a partir de los documentos WSDL de cada servicio disponible.
- **Entrega de descripciones de servicios:** la descripción del conjunto de servicios Web disponibles (operaciones, parámetros...) se envía desde el manejador de servicios al usuario móvil según el formato XML descrito en [1].
- **Invocación del servicio:** una vez que los usuarios móviles han recibido la descripción de los servicios disponibles, envían las peticiones de los servicios de interés al manejador de servicio. Con el fin de invocar las funcionalidades de los servicios de interés, se utiliza el mecanismo de invocación dinámica por parte del manejador de servicio.
- **Envío de resultados:** una vez que el manejador de servicio ha recibido la respuesta del proveedor de servicio correspondiente, envía dicha información en un mensaje XML al dispositivo móvil. Esta información es mostrada dinámicamente sobre la pantalla del dispositivo móvil.
- **UDDI:** los clientes móviles también pueden demandar servicios soportados por el registro UDDI. En esta situación, el cliente hace la petición al manejador de servicio utilizando

palabras claves. A continuación, el manejador de servicio utiliza el mecanismo de descubrimiento dinámico para descubrir aquellos servicios en UDDI que cazan con el criterio de búsqueda del usuario. La descripción de estos servicios se envían desde el manejador de servicio hacia la aplicación del usuario móvil. El usuario selecciona el/los servicios de interés. Finalmente, el manejador de servicios maneja esta petición de la misma forma que procesa la petición de servicios previamente descrita.

3.2. Manejadores de Servicios

Los manejadores de servicios actúan como una capa intermedia entre los proveedores de servicios y clientes móviles. Son responsables del flujo de información entre ambos componentes. Dicho manejador de servicio puede ser implementado mediante dos aproximaciones diferentes:

- Un servicio Web SOAP: en este caso, el manejador de servicios se implementa a su vez como un servicio Web que se comunica con los proveedores de servicios mediante SOAP.
- Un Java Servlet: utilizando principios de diseño REST [3], el manejador de servicios puede identificarse como un recurso accesible por su URI y con una interfaz genérica.

Dado que la mayor complejidad en cuanto a la implementación del manejador de servicio reside en la aproximación basada en la utilización del estilo de diseño REST, a continuación describimos los aspectos fundamentales de su desarrollo e implementación.

Desarrollo de un Proxy basado en URI

REST [3] representa un estilo de arquitectura software para sistemas distribuidos como la Web, que proporciona pautas de diseño en el desarrollo y uso de URIs y protocolo HTTP con el fin de proporcionar una mayor flexibilidad y escalabilidad en sistemas distribuidos hipermedia.

El primer paso a realizar en el desarrollo de la creación de un manejador de servicio usando REST consiste en la identificación del proxy que va a ser expuesto en la Web. Con este motivo, debe crearse un URL para este nuevo recurso. La descripción de este recurso debe permitir que los clientes (teléfonos móviles) puedan recibir una representación del recurso que corresponda a: (1) el conjunto de servicios Web disponibles hasta el momento y (2) los resultados de la invocación de las operaciones de un servicio Web. Ambos

resultados se transmiten al cliente según un formato XML. Con este objetivo, el proxy basado en URI es implementado como un Java Servlet y diferentes clases de soporte que permiten recibir las peticiones de clientes e invocación dinámica de servicios Web en tiempo real. La implementación del manejador de servicio como un Java Servlet permite que dicho manejador de servicio reciba las peticiones de clientes a través de un método GET en el servlet implementado. A continuación, a partir de la URL se determina la operación y parámetros del servicio invocado. Posteriormente, se invoca el servicio y los resultados de dicha invocación se transfieren al cliente en un formato XML.

El código principal referente a la invocación de la operación de un servicio Web es el siguiente:

```
public class DynamicInvokerServlet extends
    HttpServlet {
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
    // Obtener a partir de la URL el servicio
    // seleccionado por el cliente
        String webService = getWebService(request.
            getRequestedURI().toString());
    // Obtener a partir de la URL la operación
    // seleccionada por el cliente
        String operation = getOperation(request.
            getRequestedURI().toString());
    // Utilización de un parser XML para la gestión de
    // los eventos SAX
        DynamicInvokerOperationParserXML handler =
            new DynamicInvokerOperationParserXML();
        SAXParserFactory factory =
            SAXParserFactory.newInstance();
        SAXParser saxParser;
        try {
            saxParser = factory.newSAXParser();
            saxParser.parse(new
                ByteArrayInputStream(
                    parameters.getBytes()), handler);
        }
    // Invocación dinámica del servicio Web utilizando
    // la clase de soporte StandardServices
        String responsetomobile =
            StandardServices.getInstance().
                invokeStandardServices(webService,
                    operation, handler.getParameters());
    // Respuesta enviada al teléfono móvil en formato
    // XML
        response.getWriter().
            write(responsetomobile);
    }
}
```

Desarrollo de un cliente para un proxy basado en REST

El desarrollo de un cliente para un proxy basado en REST implica proporcionar soluciones a los siguientes requisitos: (1) cómo establecer la conexión con el proxy, el cuál es accesible como un recurso, (2) cómo realizar la petición de un servicio, (3) cómo invocar una operación de un servicio a partir de un conjunto de servicios Web disponible y (4) cómo enviar los parámetros, cuando dichos parámetros no son en si mismo recursos. Actualmente, cada uno de estos

problemas deben ser resueltos por el desarrollador debido a la falta de herramientas que proporcionen dichas funcionalidades.

Para desarrollar dicho cliente, en un primer lugar, hay que realizar la petición al proxy, seleccionando el servicio Web y operación deseada a partir del recurso “webservices”. Este recurso representa el conjunto de servicios disponibles hasta el momento actual. A continuación, se transforman los parámetros de la operación a un formato XML. Finalmente, la información correspondiente a la petición se asocia como parte de una URL de la siguiente manera:

<http://xxx.xxx.xxx.xxx/webservices/webserviceselect/operation name?parameters=xml parameters>

De esta forma, el resultado del mensaje corresponde a la longitud de la URL, es decir, se envía toda la información a través de la URL y el cuerpo del mensaje se envía vacío.

La figura 2 muestra las diferencias cuando el manejador de servicios es implementado desde una perspectiva REST y SOAP. Desde la aproximación REST, el proxy corresponde a una solución basada en servlet mientras que desde la aproximación SOAP, el proxy corresponde a un servicio Web.

La principal ventaja de utilizar una aproximación SOAP reside en que todo el proceso de establecer la conexión y creación del mensaje es totalmente transparente al desarrollador de software. Sin embargo, la utilización de una aproximación REST, requiere la implementación de dicho proceso, tal como se muestra en el siguiente código:

```
public class InvocationREST {
// Identifying the Service manager as a resource
(REST proxy)
    public static final String URI_SERVER_REST =
        "http://190.165.1.40:8080/Server/dynamicInvoke
r/";
// How to invoke an operation of a Web service
registered at proxy using REST
    public static String invokeREST(WebService
wsSelected, Operation operation,
String[] valueParameters) throws Exception {
String resultInvocation = null;
// 1. Establecer conexión
String request = prepareRequest(
wsSelected, operation, valueParameters);
HttpConnection c=
getHttpConnection(request);
// 2. Obtener Writing stream
OutputStream os = c.openOutputStream();
// Se ha establecido la conexión
// Se envían los datos
os.flush();
// Fase de lectura
// 1. Se obtiene un input Stream. Se abre una
//conexión para una petición GETF
// 2. Se lee el estado de petición
if (c.getResponseCode() ==
HttpConnection.HTTP_OK) {
// 3. Se leen las cabeceras
...
return resultInvocation;
// Es un documento XML
}
}
// Como establecer la conexión
```

```
private static HttpConnection
getHttpConnection(String request) throws
Exception {
HttpConnection c=(HttpConnection)
Connector.open(URI_SERVER_REST+request);
String size = URI_SERVER_REST + request;
// Se establece el tipo de petición
c.setRequestMethod(HttpConnection.GET);
// Se establecen las cabeceras
c.setRequestProperty("User-Agent",
"Profile/MIDP-1.0 Configuration/CLDC-
1.0");
c.setRequestProperty("Content-Language",
"es-ES");
c.setRequestProperty("Content-Type",
"application/x-www-formurlencoded");
// Cerrar la conexión después de la petición
c.setRequestProperty("Connection",
"close");
return c;
}
// Cómo hacer la petición
private static String
prepareRequest(WebService wsSelected,
Operation operation,String[] valueParameters){
// Datos del servicio Web a ser invocado desde el
// conjunto de servicios Web disponibles en el
// Manejador de servicios
StringBuffer request = new StringBuffer();
// A) Buscando en el recurso webservices
request.append("webservices/");
// B) Servicio Web seleccionado
peticion.append(wsSelected.getName()+"/");
// C) Operación seleccionada del servicio Web
// seleccionado
peticion.append(operation.getName());
// D) Los parámetros son transformados a un
//formato XML
request.append(
"?parameters=<parameters>");
for (int i = 0; i<valueParameters.length;
i++) {
request.append("<parameter>" +
valueParameters[i] + "</parameter>");
}
request.append("</parameters>");
return request.toString();
}
}
```

4. Implementación y Resultados

La arquitectura descrita en la sección 3 permite que los usuarios móviles puedan acceder a servicios Web publicados en la Web mediante la petición a un proxy, que hemos denominado manejador de servicios. No se requiera ninguna actualización ni descarga de software cuando se proporcionan nuevos servicios en tiempo de ejecución. Para su implementación se ha utilizado el siguiente software: Apache Tomcat 5.0.28 como servidor de aplicaciones, J2ME Wireless Toolkit (WTK) para el desarrollo de aplicaciones wireless que se ejecutan sobre teléfonos móviles, y Eclipse 3.1 como plataforma de desarrollo con WTP (Web Tools Platform) plug-in para el desarrollo de aplicaciones Web. Axis y UDDI4J librerías Java se han utilizado como motor SOAP e implementación del protocolo UDDI. La aplicación cliente ha sido implementada como un MIDlet usando el Toolkit J2ME Wireless. Es decir, una aplicación Java desarrollada con un perfil MIDP y una configuración CLDC.

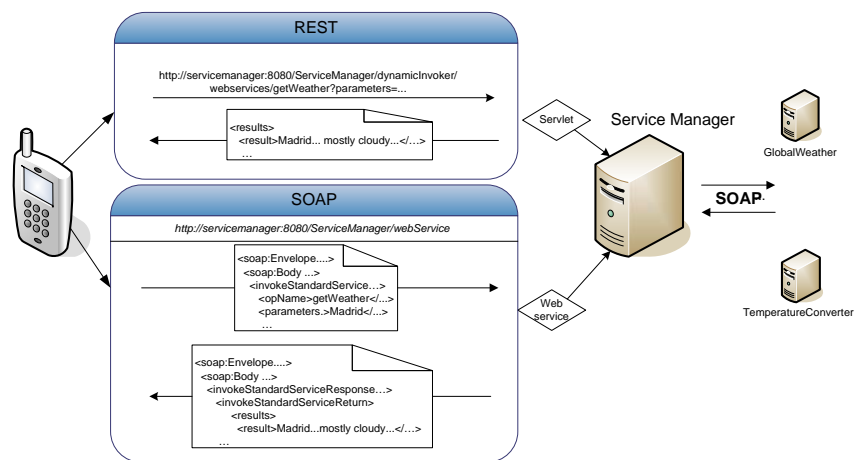


Figura 2. Manejador de servicio utilizando una aproximación REST y SOAP. La utilización de REST conlleva a que toda la información se envía a través de la URL (todo el proceso necesita ser implementado por el desarrollador de software) y el manejador de servicio corresponde con una solución basada en servlet. Utilizando una aproximación SOAP, la información es enviada a través del mensaje (todo el proceso es totalmente transparente al desarrollador de software) y el manejador de servicio corresponde con un servicio Web.

4.1. Resultados Experimentales

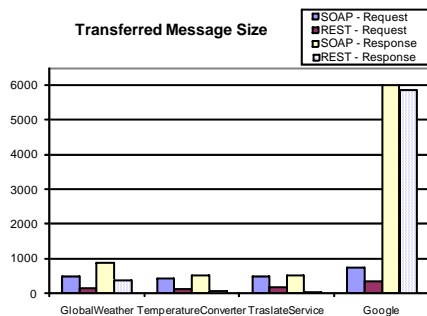
El componente principal de nuestra arquitectura es el manejador de servicio (proxy). Una característica fundamental de este componente reside en su habilidad para procesar peticiones procedentes desde los teléfonos móviles. Por ello, medimos el rendimiento de nuestro manejador de servicios para las peticiones realizadas desde los teléfonos móviles utilizando las dos aproximaciones descritas en este artículo: (i) SOAP y (ii) REST. La primera aproximación corresponde con el caso en que el manejador de servicios es un servicio Web y el protocolo de comunicación utilizado con los teléfonos móviles es SOAP. En este caso, las invocaciones a los diferentes servicios se realizan mediante un único stub (el cuál corresponde al manejador de servicios) y ha sido añadido al cliente en tiempo de compilación, tal cómo se describe en el artículo [1]. La segunda aproximación corresponde con el caso en que el manejador de servicios es implementado como un servlet y el cliente REST utiliza el protocolo HTTP GET para enviar las peticiones de solicitud desde su dispositivo móvil hacia el manejador de servicios. Ambas aproximaciones utilizan el mecanismo de invocación dinámica con el fin de realizar en tiempo de ejecución las invocaciones correspondientes por parte del manejador de servicios. Los siguientes servicios se han utilizado como escenarios para llevar a cabo las medidas de evaluación y rendimiento: (1) servicio de

meteorología, (2) convertor de temperaturas, (3) servicio de traducción de un lenguaje a otro y (4) búsqueda con Google.

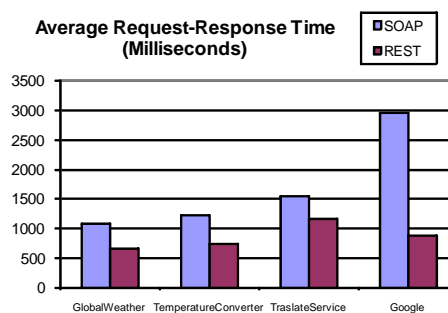
Se ha calculado los resultados para las siguientes medidas de evaluación: (i) tamaño del mensaje enviado y (ii) tiempo de respuesta por parte del manejador de servicios considerando la aproximación SOAP y REST. Para cada una de las medidas (i) y (ii) se muestra el resultado medio tras la realización de diferentes invocaciones a cada uno de los servicios. La gráfica de la figura 3a muestra el tamaño del mensaje enviado para cada servicio propuesto y aproximación evaluada. El eje x representa el tipo de servicio y el eje y corresponde al tamaño de mensaje enviado en bytes. Los resultados de esta figura muestran que: (i) el volumen medio de datos transmitido por una petición REST es 3 veces inferior a una petición SOAP y (ii) el volumen medio de datos transmitido por una respuesta REST es 1.5 veces inferior en relación a una respuesta SOAP. La figura 3b muestra el tiempo medio de petición-respuesta para cada servicio y aproximación a ser evaluada. El eje y representa el tiempo medio medido en milisegundos. Los resultados de esta figura muestran que el tiempo medio de una petición-respuesta utilizando una aproximación REST es justamente la mitad que si se utiliza una aproximación SOAP.

4.2. Coste del desarrollo

Uno de los principales requerimientos desde el punto de vista del desarrollador del software se



(a)



(b)

Figura 3. SOAP vs. REST: (a) Longitud de mensajes (en bytes) and (b) Tiempo medio petición-respuesta (en milisegundos)

centra en la reducción del ciclo de vida del desarrollo. En la actualidad, consideramos que desde el punto de vista de la implementación tanto en el lado del servidor como en el lado de la aplicación cliente, es más fácil y sencillo utilizar una aproximación SOAP. La razón principal estriba en que la utilización de la aproximación SOAP dispone de un conjunto de herramientas apropiadas de desarrollo así como los estándares disponibles. Sin embargo, si se adopta una aproximación REST, se necesitan abordar diferentes problemas tales como: (i) cómo llevar a cabo la implementación de la conexión, (ii) cómo realizar el análisis de la información enviada desde el teléfono móvil hasta el manejador de servicios y (iii) cómo enviar los parámetros de una operación invocada desde el teléfono móvil hasta el manejador de servicio, debido a que los parámetros no son en sí mismos recursos. Asimismo, hemos encontrado que la información disponible para el desarrollo e implementación de servicios para teléfonos móviles utilizando una aproximación REST no es suficiente.

5. Conclusiones

En este artículo se describe una arquitectura dinámica orientada a servicios que permita a usuarios desde su dispositivo móvil descubrir y acceder a servicios de su interés en tiempo real. Se propone la utilización de un proxy como elemento intermediario entre los proveedores de servicios y el dispositivo móvil con el fin de proporcionar las funcionalidades específicas que requiere el descubrimiento y acceso dinámico a servicios ubicados en una infraestructura Web. Dicho proxy puede implementarse desde una aproximación REST y/o SOAP. Desde una aproximación REST, el proxy es expuesto a la Web como un recurso accesible a través de su URI. La principal ventaja reside en la reducción del tamaño medio de mensajes tanto en la petición de servicios desde el

dispositivo móvil así como en la respuesta hacia dichos dispositivos. Sin embargo, las principales limitaciones de adoptar una aproximación REST son: (1) cómo enviar parámetros de una operación invocada desde un teléfono móvil hacia el proxy, debido a que los parámetros no son en sí mismos recursos y (2) cómo enviar grandes volúmenes de información debido a que no existen estándares y especificaciones sobre cómo estructurar la información a enviar. Todo ello, se traduce en una mayor complejidad en el desarrollo e implementación por parte del desarrollador de software. Por ello, la utilización de una aproximación basada en REST debería ser una solución admisible en aquellas situaciones donde se debe limitar las cabeceras y capas adicionales introducidas por SOAP y/o se desea diseñar una interfaz sencilla para el componente proxy. El rendimiento de la arquitectura propuesta ha sido evaluado en una plataforma J2ME, obteniendo que el volumen medio de datos transferidos y el tiempo medio de petición-respuesta es justamente la mitad cuando se adopta una aproximación REST frente a una aproximación SOAP.

Referencias

1. Elena Sánchez-Nielsen, Sandra Martín-Ruiz, Jorge Rodríguez-Pedrianes. "An open and dynamical service oriented architecture for supporting mobile services". ACM ICWE 2006, pp. 121-128, Palo Alto, California.
2. T. Pilioura, S. Hadjiefthymiades, A. Tsalgatidou, M. Spanoudakis. "Using Web services for supporting the users of wireless devices". Journal of Decision Support Systems 43 (2007) 77-94.
3. R.T. Fielding. Architectural Styles and the Design of Network-Based Software Architectures. Doctoral Dissertation, Dept. of Computer Science, Univ. of California, Irvine, 2000.