

RESUMEN DE LA MEMORIA
PARA OPTAR AL TITULO DE
INGENIERO CIVIL EN COMPUTACION
POR: VICTOR CARRASCO H.
FECHA: 15/01/2007
PROF. GUIA: Sr. SERGIO OCHOA

“SERVICIOS WEB EN DISPOSITIVOS MÓVILES PARA EL SOPORTE DE APLICACIONES COLABORATIVAS”

Este trabajo de título tiene como objetivo el investigar y aplicar dos estándares de servicios Web llamados WS-Attachment y WS-Security. Estos estándares proveen una mejor comunicación y coordinación entre aplicaciones colaborativas que necesitan intercambiar información y autenticarse. Además, ellos facilitan la integración de servicios expuestos o consumidos en dispositivos computacionales móviles, lo que permiten un trabajo colaborativo en escenarios ad-hoc.

Para llevar a cabo este trabajo se tomó como base un servidor Web para dispositivos móviles, el cual fue diseñado para ocupar el mínimo de recursos de hardware. Este servidor solamente aceptaba clientes que solicitaban algún servicio Web, ejecutaba el servicio requerido y enviaba al cliente la respuesta del servicio. Por lo tanto, luego de conocer en detalle las especificaciones estándares para adjuntos (WS-Attachments) y seguridad (WS-Security), se estudiaron los procesos generados por el servidor Web, para así poder agregarle de la forma más modular posible, las nuevas implementaciones correspondientes a dichos estándares.

De esta forma, se obtuvo un MicroServidor Web ideal para aplicaciones colaborativas móviles que necesiten intercambio de archivos y autenticación. Además, este producto utiliza pocos recursos de hardware y permite obtener buenos tiempos de respuesta aún en dispositivos pequeños como PDAs. El MicroServidor fue sometido a pruebas de desempeño y de concurrencia, alcanzando resultados muy alentadores. Sin duda, las mejoras desarrolladas en el MicroServidor, sobre adjuntos y seguridad, le permiten a los servicios Web abarcar muchos más escenarios de integración de sistemas.

La memoria también presenta un par de aplicaciones colaborativas, cuyos desarrollos fueron inicialmente independientes del MicroServidor. Sin embargo, luego de re-implementarlas sobre el MicroServidor, se pueden ver las mejoras en los servicios introducidos por el uso de este componente base. A pesar de lo útil que es ahora el MicroServidor, se debe continuar su desarrollo en base a otros estándares, para un mayor beneficio a usuarios que, con el uso de dispositivos móviles, buscan mejorar sus capacidades de trabajo.

Índice

Capítulo 1. Introducción	5
1.1. Motivación.....	6
1.2. Objetivo General	6
1.3. Objetivos Específicos	6
1.4. Metodología.....	7
Capítulo 2. Marco Teórico	8
2.1. Servicios Web.....	8
2.1.1. SOAP (Simple Object Access Protocol)	9
2.1.2. WSDL (Web Services Definition Language).....	10
2.2. Estándares de Servicios Web.....	11
2.2.1. WS-Attachments.....	12
2.2.1.1. DIME (Direct Internet Message Encapsulation)	12
2.2.2. WS-Security	13
2.3. Trabajo Relacionado.....	14
2.4. Resumen	16
Capítulo 3. WS-Attachments	17
3.1. Adjuntos en el MicroServidor Web.....	17
3.1.1. Interfaz de Adjuntos	17
3.2. Módulo DIME	19
3.2.1. Tarjeta DIME.....	19
3.2.2. Generador DIME	21
3.2.3. Intérprete DIME	22
3.2.4. Atributo DIME	23
3.3. Modificaciones en el MicroServidor Web.....	24
3.3.1. Recepción de Datos de un Cliente.....	24
3.3.2. Módulo Attachments	25
3.3.3. Módulo SOAP	26
3.4. Usando WS-Attachments	27
3.4.1. Crear un adjunto	27
3.4.2. Adjuntos en un servicio Web	28
3.4.3. Adjuntos en un cliente	28
3.5. Resumen	29
Capítulo 4. WS-Security	31
4.1. Datos de Autenticación.....	31
4.2. Módulo Seguridad	32
4.3. Seguridad en el Microservidor Web.....	34
4.3.1. Herramienta de Seguridad	34
4.3.2. Archivo de Configuración de un Servicio Web	35
4.3.3. Modificaciones en el Módulo SOAP.....	35
4.4. Seguridad en un Cliente.....	37
4.5. Resumen	39
Capítulo 5. Resultados, Conclusiones y Trabajo a Futuro	40
5.1. Pruebas de Desempeño	40
5.2. Pruebas Concurrentes	41
5.3. Adjuntos v/s Parámetros.....	42
5.4. Mapa Móvil	44

5.5. Mobile File Sharing.....	44
5.6. Conclusiones.....	45
5.7. Trabajo a Futuro	45
Bibliografía y Referencias.....	47
Apéndice A: Código Fuente.....	49
A.1. Código Fuente WS-Attachment.....	49
Interfaz IAttachment.....	49
DimeAttribute.....	50
A.2. Código Fuente WS-Security	53
Interfaz ISecurityToken.....	53
UsernameToken.....	53
A.3. Mejoras Realizadas en el MicroServidor.....	55

Capítulo 1. Introducción

Los servicios Web crecen con éxito como alternativa de integración de servicios propietarios entre organizaciones independientes. Gracias a los estándares creados para facilitar aquella integración, todo se ha hecho más fácil. Tal como su nombre lo indica, los servicios Web son aquellos servicios que expone una organización, empresa, o cualquier tipo de proveedor, para que sean utilizados por usuarios a través de la Web, aprovechando todos los recursos que esta plataforma de intercomunicación entrega.

Dentro de los beneficios que aportan los servicios Web, aparte de la mencionada integración entre organizaciones independientes, está el hecho de que son simples y fáciles de utilizar, tanto desde el punto de vista del proveedor de servicios como del consumidor. Ahora bien, esta ventaja está sujeta a los recursos que un servidor o computador tiene, pues la capacidad de procesamiento, disponibilidad de memoria y la velocidad de intercambio de información a través de la Web, afectan la eficiencia de estos servicios. Por el momento, esto restringe el uso de servicios Web en dispositivos móviles pequeños (como teléfonos celulares o PDAs), pues los recursos de hardware de estos dispositivos son limitados con respecto a los de un computador de escritorio o un notebook.

La idea de exponer o consumir servicios Web en dispositivos móviles pequeños nace del importante uso que éstos están teniendo en el último tiempo, y al incremento permanente de las prestaciones de dichos dispositivos. Por dar un ejemplo, las PDAs (Personal Digital Assistants) toman especial consideración, pues están siendo muy exitosas entre hombres de negocios y administradores, al ofrecer herramientas tan útiles como las de un computador personal o notebooks [2], a un costo muy inferior y con mayor facilidad para ser transportadas. De esta forma aparecen los denominados “usuarios móviles” o “trabajadores móviles”, quienes tienen la necesidad de obtener servicios en cualquier momento y en cualquier lugar.

Tal como se indicó en un inicio, se han creado estándares para facilitar la integración de servicios. Desafortunadamente, las implementaciones de estos estándares existentes hasta el momento están principalmente enfocadas a dispositivos con un poder computacional superior a una PDA. Por esa razón, este trabajo de título consiste precisamente en investigar y aplicar ciertos estándares de Servicios Web, a un servidor ya existente para dispositivos móviles pequeños, considerando, al mismo tiempo, que deben ser aplicados de acuerdo a los recursos de hardware (limitados) que éstos poseen.

La existencia de un servidor para dispositivos móviles hace más fácil el trabajo colaborativo, ya que permite una constante comunicación y coordinación entre un grupo de personas. Ahora bien, si el servidor tuviera la capacidad de recibir o enviar diferentes tipos de archivos (documentos, programas, etc.) o si tuviera la capacidad de identificar a las personas que lo invocan, daría mayor valor a las aplicaciones colaborativas que comparten información entre personas que realizan una tarea en común. Entre los estándares de servicios Web existen definiciones de archivos adjuntos (WS-Attachments) y de seguridad (WS-Security) los cuales corresponden a los estándares que se investigarán y aplicarán en este trabajo de título.

A continuación se presenta la motivación, los objetivos y el plan de trabajo definidos para esta memoria. Luego, en el capítulo 2 se incluye un marco teórico para acercar al lector al conocimiento de un servicio Web y sus estándares asociados. En los capítulos 3 y 4 se presenta el

trabajo realizado sobre WS-Attachment y WS-Security respectivamente, y las soluciones que se implementaron al respecto. Finalmente, el capítulo 5 presenta las conclusiones y el trabajo a futuro.

1.1. Motivación

Este trabajo de título se hace parte del gran desafío de disminuir la distancia que separa lo que es un dispositivo móvil pequeño, de un computador personal. Éste es un desafío complicado si se considera que los dispositivos móviles pequeños están continuamente progresando tecnológicamente, lo que significa que cualquier desarrollo realizado en este ámbito debe ser lo suficientemente flexible para adaptarse a los inminentes cambios tecnológicos. En este sentido el trabajo de título realizado se convertirá en un aporte a los usuarios móviles, desde dos puntos de vista:

1. Tener un dispositivo que exponga y/o consuma servicios Web, adhiriendo a los estándares, independientemente de si es un teléfono celular, una PDA o un notebook. De esa manera se facilita su integración con cualquier otro servicio Web.
2. Tener una herramienta que les permita exponer y/o consumir servicios Web, manteniendo los recursos disponibles para utilizar otras herramientas que los dispositivos móviles ofrecen.

Sin duda, éste es un trabajo que incentiva a los usuarios móviles a ampliar sus opciones de movilidad y a mantener su actividad de interacción con otros usuarios (personas u organizaciones), en cualquier momento y en cualquier lugar.

1.2. Objetivo General

El objetivo general al que apunta este trabajo de título es investigar y aplicar soluciones estándares de servicios Web, a una plataforma de groupware móvil, para así facilitar la integración de servicios expuestos o consumidos en dispositivos computacionales móviles que van desde un notebook hasta un teléfono celular.

1.3. Objetivos Específicos

Los objetivos específicos de este trabajo de título son los siguientes:

1. Determinar hasta qué grado el desarrollo existente de la plataforma de groupware, cumplía con las expectativas de ocupar el mínimo de recursos de hardware de un dispositivo móvil, para así hacer los ajustes necesarios para tener una base con la cual partir el desarrollo de aplicaciones móviles basadas en servicios Web.

2. Investigar, diseñar e implementar cómo un dispositivo móvil se comunica con el proveedor de un servicio o con un cliente que consume servicios Web del dispositivo, siguiendo los estándares WS-Security y WS-Attachments.¹
3. Implementar una aplicación que exponga y consuma servicios Web utilizando los estándares antes mencionados, para así poder determinar los alcances y/o capacidades de las implementaciones hechas.

1.4. Metodología

El plan de trabajo que se diseñó para lograr los objetivos definidos, fue el siguiente:

1. Terminar el estudio, desde el punto de vista del desarrollo, de la documentación relacionada con estándares de implementación, en particular de seguridad y mensajería (WS-Security y WS-Attachments).
2. Diseño e implementación de una solución de software que permita a un servicio Web estar activo como servidor en un dispositivo móvil, aplicando el estándar WS-Attachments.
3. Diseño e implementación de una solución de software que permita a un servicio Web estar activo como servidor en un dispositivo móvil, aplicando el estándar WS-Security.
4. Diseño e implementación de servicios Web complejos, que utilizan WS-Security y WS-Attachments.
5. Prueba del comportamiento de los servicios Web complejos cuando son expuestos/consumidos por dispositivos PDAs que se desplazan.
6. Análisis de resultados y ajuste de la solución propuesta.

Un punto a mencionar dentro del plan de trabajo, es que las implementaciones y sus correspondientes pruebas se realizaron inicialmente utilizando PDAs y notebooks. Luego, varias de éstas se aplicaron a teléfonos celulares, con el objeto de evaluar las prestaciones de la solución, sobre dispositivos con hardware muy limitado.

¹ Estos estándares se detallan en el Capítulo 2 de este documento, bajo el subtítulo “Estándares de servicios Web”.

Capítulo 2. Marco Teórico

Este capítulo introduce los conceptos necesarios para comprender el resto del trabajo de título. En la próxima sección se explica con más detalle lo que es un servicio Web, dando especial consideración a la estructura de mensajes existentes para la comunicación entre componentes independientes. Además, se verán los estándares que serán aplicados a los servicios Web en dispositivos móviles. Se incluyen también las referencias de un desarrollo existente de servicios Web para dispositivos móviles.

2.1. Servicios Web

Los servicios Web podrían verse como la transformación de las funciones de software tradicionales a funciones estandarizadas, las cuales son visibles y accesibles por cualquier aplicación de software, utilizando a Internet como medio de comunicación. Estos servicios se enfocan en tareas específicas y abarcan desde simples pedidos hasta complejos procedimientos técnicos, que pueden ser invocados por otros servicios Web o aplicaciones [14].

Un servicio Web es comparable con un módulo o componente de software. Éste evita el conocido problema de hacer cambios o modificaciones a una aplicación, en donde generalmente un cambio significa cambios en todo el sistema, debido a la alta dependencia entre ellos (alta mantenibilidad y flexibilidad). Por otra parte, los servicios Web son utilizados por aplicaciones u otros servicios Web, sin que los desarrolladores originales de las aplicaciones tengan conocimiento de su implementación. Estos servicios pueden ser vistos como un plug-in en los desarrollos de sistemas de software.

Los servicios Web ofrecen la flexibilidad a los desarrolladores de trabajar en cualquier lenguaje de programación, permitiéndoles además utilizar estos servicios sin tener que adaptarse a nuevos ambientes de desarrollo. Por otra parte, este tipo de soluciones promueve la interoperabilidad entre sistemas, ayudando a la integración de organizaciones independientes.

A grandes rasgos, se pueden distinguir tres aspectos importantes, que juntos hacen posible el desarrollo de los servicios Web: (1) un protocolo estándar de comunicación e interoperabilidad entre servicios, como SOAP (Simple Object Access Protocol); (2) una forma de proveer, tanto a los desarrolladores como a aplicaciones, información detallada de los servicios ofrecidos y cómo utilizarlos, como los WSDL (Web Services Definition Language); y (3) un lugar donde los servicios Web puedan ser publicados y así ser buscados por los clientes, como lo son los directorios UDDI (Universal Description, Discovery Interface) [8].

Es de vital importancia en este trabajo de título conocer más a fondo los aspectos descritos anteriormente, pues éstos son la base del desarrollo y existencia de un servicio Web. A continuación se explica con más detalle SOAP y WSDL, pues son dos aspectos muy relacionados entre sí, cuyo entendimiento es vital en este trabajo de título.

2.1.1. SOAP (Simple Object Access Protocol)

SOAP es un protocolo estándar de comunicación e interoperabilidad entre servicios Web y aplicaciones. SOAP define mensajes de formato XML que son los que finalmente se intercambian entre servicios y aplicaciones. En general, un archivo XML que contiene elementos definidos por SOAP (elementos SOAP) es un mensaje SOAP.

La importancia de los mensajes SOAP reside en que en ellos se describen las funciones, con sus respectivos parámetros, que pueden ser ejecutadas en un servicio Web. Cuando un emisor envía un mensaje SOAP, luego éste recibe de vuelta otro mensaje SOAP, pero en este caso con los resultados de la función requerida. La Figura 2.1 muestra que la comunicación se realiza solamente utilizando los mensajes SOAP.

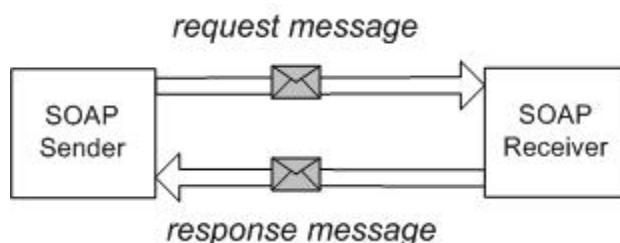


Figura 2.1. Comunicación usando SOAP

La estructura de un mensaje SOAP contiene los siguientes elementos:

1. El elemento raíz distintivo de este tipo de mensaje, llamado “Sobre” (“Envelope”), con el cual se puede determinar la versión del mensaje que se está utilizando¹. Este elemento contiene, a la vez, los elementos “Encabezado” (“Header”) y “Cuerpo” (“Body”).
2. El elemento “Header”, el cual es optativo, pero importante al momento de querer negociar el comportamiento de las peticiones requeridas o respuestas esperadas. Éste es el elemento que hace de SOAP un protocolo extensible. De hecho, es muy importante a la hora de aplicar los estándares de servicios Web, el cual es uno de los objetivos de este trabajo de título.
3. El elemento “Body”, que contiene la información que el servidor espera, para poder ejecutar los procesos o funciones especificadas; o bien, es la información que un cliente espera como respuesta de una petición. Dentro de este elemento se agrega uno llamado “Fault”, que es útil para indicar con detalle peticiones que han fallado.

¹ Actualmente existen 2 versiones de SOAP: 1.1 y 1.2, siendo SOAP 1.1 la versión por defecto. Sin embargo, hay una alta probabilidad de que pronto SOAP 1.2 sea la nueva versión por defecto.

La Figura 2.2 ilustra la estructura de un mensaje SOAP.

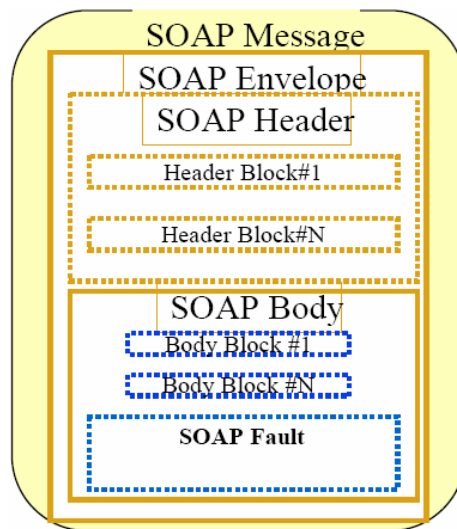


Figura 2.2. Estructura mensaje SOAP

Luego de tener una noción global de SOAP, se describe en esta sección la importancia que tienen estos mensajes para el desarrollo de servicios Web. Incluso se especifica qué es el mensaje a manipular, si se quieren aplicar nuevos conceptos o estándares a un servicio Web, por lo que es necesario familiarizarse con ellos para entender este trabajo de título.

2.1.2. WSDL (Web Services Definition Language)

WSDL es un documento XML que describe un conjunto de mensajes SOAP y cómo los mensajes son enviados y recibidos. Es un documento que especifica, sin ambigüedad, lo que debe contener un mensaje de petición y lo que debe contener un mensaje de respuesta.

Parte de las especificaciones en un WSDL consiste en determinar dónde está disponible un servicio y qué protocolo de comunicación utiliza. Todo lo anterior significa que el archivo WSDL define todo lo necesario para escribir un programa que trabaja con un servicio Web. La estructura de un documento WSDL puede describirse como sigue:

1. *Types*: Se definen los tipos que serán utilizados por los mensajes que se especifican a continuación en el documento. Cuando se habla de “definición de tipos” se hace referencia a la definición que permite realizar, por ejemplo, XSD (XML Schema Definition), el cual permite definir estructuras de documentos.
2. *Message*: Ésta es una definición abstracta de los datos que son transferidos entre un cliente y un servidor. Cada mensaje tiene asociado un tipo, ya definido anteriormente en el documento.
3. *PortType*: Éste representa el conjunto de operaciones que un servicio Web expone, en donde cada operación está compuesta por un mensaje de entrada y salida, y mensajes ya definidos anteriormente en el documento. Opcionalmente, y dependiendo de la implementación de cada servicio Web, puede incluirse un mensaje de error.

4. *Binding*: El binding especifica el conjunto de operaciones a usar dependiendo del protocolo de comunicación utilizado. Pueden hacer referencia a los mismos portTypes ya definidos.
5. *Port*: Éste especifica la ubicación concreta de dónde se encuentra el servicio Web dependiendo del protocolo de comunicación utilizado. A la vez, tiene asociado el binding a usar.
6. *Service*: Es el elemento que contiene el conjunto de ports.

La Figura 2.3 ilustra la estructura de un documento WSDL, como el antes descrito.

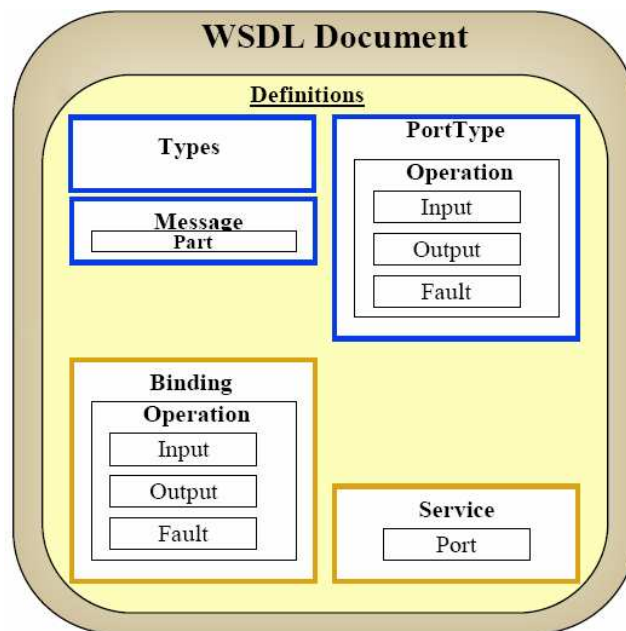


Figura 2.3. Estructura Documento WSDL

Existe un calce directo entre la definición de un documento WSDL y la estructura de un mensaje SOAP. Por lo tanto, el documento WSDL es importante para invocar un servicio Web. Sin el entendimiento del WSDL, simplemente no se tiene conocimiento de cómo interactuar de forma exacta con cierto servicio Web.

2.2. Estándares de Servicios Web

Debido a la gran cantidad de servicios Web desarrollados, y gracias a las ventajas que éstos dan, se han creado estándares para proveer a los usuarios de servicios Web de capacidades adicionales, que ayudan a aumentar la disponibilidad, confiabilidad y seguridad de estos servicios. Así nace la Global XML Web Services Architecture (GXA), la cual ofrece un conjunto de estándares para aplicar a los servicios Web. GXA continúa con la misma línea existente, es decir, el uso de SOAP y WSDL, aprovechando el hecho de que SOAP es un protocolo de comunicación fácil de extender. Dentro de los muchos estándares que GXA ha propuesto [10], a continuación sólo se detallan los involucrados en este trabajo de título.

2.2.1. WS-Attachments

WS-Attachments es un estándar propuesto para servicios Web, que aprovecha las ventajas del protocolo DIME (Direct Internet Message Encapsulation) de encapsulación de mensajes. WS-Attachments fue diseñado para facilitar el envío de archivos adjuntos con los mensajes SOAP. El objetivo de DIME es transportar de manera más eficiente archivos adjuntos a mensajes SOAP, lo que es especialmente útil en el caso de servicios Web que necesitan incluir archivos binarios de gran tamaño; por ejemplo archivos multimedia o archivos de datos binarios. Debido a que el protocolo principal a entender en WS-Attachments es DIME, es necesario conocer más a fondo en qué consiste éste.

2.2.1.1. DIME (Direct Internet Message Encapsulation)

DIME nace con la idea de transmitir datos en “paquetes” aprovechando algunos beneficios, tales como no tener que procesar toda la información de una vez y tener anticipadamente un conocimiento de todos los datos recibidos, sin tener que procesarlos de forma completa. DIME es una forma de ordenar aquellos paquetes, bajo el concepto de “tarjetas DIME”, tarjetas de formato binario, rigurosamente estructuradas.

El paquete completo de información está formado por muchas tarjetas, en donde cada una representa una información independiente. Ahora bien, en el caso que alguna información cuya extensión binaria sobrepase el máximo de una tarjeta, aquella información es particionada en dos o más tarjetas.

Para mantener el orden de las tarjetas, o en otras palabras, para determinar cuál tarjeta es la primera, la última, cuál contiene su información particionada, etc., la tarjeta DIME tiene un encabezado binario con indicadores precisos en los cuales marcar tal información. Por sólo dar un ejemplo, existen los indicadores MB (Message Begin) y ME (Message End) que marcan la primera y la última tarjeta, respectivamente. La Figura 2.4 muestra el formato de estas tarjetas. Nótese que en el encabezado de éstas están los indicadores antes explicados.

Version 00001	M B	M E	C F	Type Format	Reserved	Options Length
ID Length					Type Length	
Data Length						
Options						
ID						
Type						
Data						

Figura 2.4. Formato de una tarjeta DIME

Se puede notar que en el encabezado también se indica la versión DIME de la tarjeta (que hasta el momento es sólo la versión 1), un indicador CF (Chunk Flag) que marca que la información (Data) de la tarjeta prosigue en otra (tarjeta particionada); el formato de la información y un identificador propio de la tarjeta dentro del paquete.

También la tarjeta tiene un sector que indica los largos de los datos que vienen a continuación (Options, ID, Type, Data). Estos valores son útiles para calcular fácilmente el largo de una tarjeta y así moverse rápidamente entre una tarjeta y otra.

Una vez que se ha entendido cómo funciona el empaquetamiento DIME, queda por determinar cómo un mensaje SOAP se mezcla con DIME. Por un asunto de convención y orden, WS-Attachments indica que la primera tarjeta DIME de un paquete corresponde al mensaje SOAP. Ahora queda por determinar cómo referenciar los datos adjuntos al mensaje. Para esto, WS-Attachments utiliza un atributo SOAP llamado “href”, el cual es generalmente usado para apuntar a URLs, pero también se acepta que apunte al id de una tarjeta DIME.

Ya con la noción teórica de lo que propone WS-Attachments, resta por implementar una solución adecuada para dispositivos móviles, para recibir un paquete DIME, obtener el mensaje SOAP correspondiente e implementar la forma de buscar los datos adjuntos correspondientes.

2.2.2. WS-Security

Organizaciones, empresas u otros tipos de proveedores de servicios Web necesitan asegurarse que sus servicios están siendo utilizados o invocados por clientes autorizados. Por otra parte, es importante asegurarse que los mensajes SOAP sean vistos y modificados por personas apropiadas. Es por eso que se necesita seguridad dentro del ambiente de los servicios Web, pero al mismo tiempo, esta especificación debe ser lo suficientemente flexible como para soportar la gran cantidad de tecnologías de autenticación que existen.

WS-Security especifica cómo usar los estándares de seguridad, de firmas y encriptación, para autenticar y asegurar integridad y confidencialidad de los mensajes SOAP. De esta forma, un servicio Web, antes de procesar una petición, debe evaluar primero si los mensajes recibidos siguen la política de seguridad previamente definida.

Lo anterior significa que se deben conocer y aplicar diferentes especificaciones como lo son, por ejemplo, XML Signature, la cual asegura integridad de partes de documentos XML transportados; XML Encryption para asegurar la confidencialidad del mensaje¹; así como los certificados X.509 que se ocupan del tema de la autenticación de usuarios. WS-Security define un marco para incorporar estos mecanismos en los mensajes SOAP.

Anteriormente, en este mismo capítulo, se mencionó que el elemento “Header” del mensaje SOAP era el que hace posible que un mensaje sea extensible; de hecho, es precisamente dentro de

¹ XML Signature y XML Encryption son especificaciones que detallan los elementos que un documento XML debe contener para tener una firma o datos encriptados, de manera que todas las partes negociantes sepan cómo tratar o procesar aquellos elementos.

este elemento en donde se agrega la información que WS-Security aporta, y que se detalla a continuación.

1. *Security*: Es el elemento raíz de la especificación WS-Security, dentro del elemento Header del mensaje SOAP.
2. *UsernameToken*: Elemento que ofrece información útil para un mecanismo simple de autenticación (usuario-clave), para el caso en que un servicio Web utilice un modo de autenticación personalizada.
3. *BinarySecurityToken*: Elemento de utilidad para el uso de certificados X.509. Este elemento debe incluir el certificado propiamente tal.
4. *SecurityTokenReference*: Ofrece información de seguridad anexa, indicadas por un enlace externo.
5. *KeyInfo* y *Signature*: Elementos útiles para el uso de la especificación XML Signature. Ofrece información de la llave y firma de una o más partes del mensaje SOAP.
6. *ReferenceList*: En éste se listan referencias a uno o más elementos encriptados en el elemento Body del mensaje SOAP. Tiene su base en la especificación XML Encryption.
7. *EncryptedKey* y *EncryptedData*: Elementos útiles para el uso de la especificación XML Encryption. Ofrece información encriptada de la llave y datos en el mismo Header de SOAP.

La implementación o uso de cada uno de los elementos indicados anteriormente, depende del nivel de seguridad que se quiera tener en el servicio Web, ya que WS-Security abarca desde una autenticación simple de usuario-contraseña, hasta el uso de llaves públicas y privadas enviando un mensaje cifrado. Por lo tanto, se puede desarrollar una amplia variedad de modelos de seguridad, acomodando los elementos apropiados para dar una solución adecuada a un dispositivo móvil. Por otra parte, el usar encriptación afecta lo que es el tiempo de proceso del mensaje SOAP, sobre todo si se habla de trabajos en dispositivos móviles, así que, en el caso de encriptar, se debe determinar qué elementos del mensaje es necesario encriptar y un algoritmo adecuado a usar para tal efecto.

2.3. Trabajo Relacionado

Como base de trabajo se tiene la implementación de un servidor Web para dispositivos móviles [18]. Éste tiene desarrollado un módulo SOAP, que analiza un mensaje SOAP, obteniendo toda la información que aquel mensaje contiene. Este módulo debe ser especialmente tratado durante el desarrollo del trabajo de título, pues como se ha visto, el mensaje SOAP, gracias a su extensibilidad y aplicación de estándares, vendrá con información que este módulo no tiene implementado y que obviamente es parte del trabajo documentado en esta memoria.

El servidor Web para dispositivos móviles tiene 3 componentes principales, los cuales son CompactListener, MobileWebServer y Plug-ins, ilustrados en la Figura 2.5. *CompactListener* es el responsable de manejar las peticiones de los clientes en un determinado puerto.

MobileWebServer es la base del servidor Web, el cual recibe las peticiones captadas por el *CompactListener*, valida si es necesario y determina los módulos a utilizar. Por último, *Plug-ins* son los módulos que soporta el servidor Web, y que están relacionados con un cierto protocolo (Http, SOAP, etc).

Los Plug-ins o módulos se mantienen debidamente separados del resto de los componentes, para mayor flexibilidad. De esa manera, los usuarios de un servicio Web podrán utilizar el protocolo que más se adecue a sus necesidades. Además, el manejar así los módulos permite adaptarse fácilmente a futuros protocolos. Actualmente, se encuentran los módulos SOAP y Http. El módulo SOAP tiene a su cargo la creación de mensajes que corresponden a la respuesta hacia el cliente que invocó cierto servicio. Por otro lado, también genera el documento WSDL, vital para que los clientes sepan los detalles necesarios de servicios que existen.

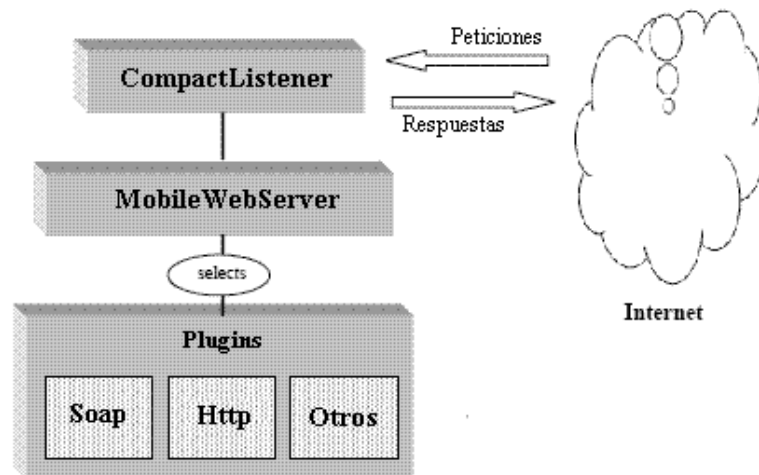


Figura 2.5. Componentes Servidor Web

El desarrollo actual consiste en un servidor que espera peticiones de clientes, aceptando protocolos POST, GET y SOAP. Una vez recibida la petición, el servidor ejecuta el servicio Web requerido para, finalmente, enviar la respuesta del aquel servicio al cliente. Además, existe una GUI en donde activar y desactivar el servidor para que escuche las peticiones de los clientes (ver Figura 2.6). Luego de activar el servidor, éste tiene una página inicial que marca que el servidor está efectivamente trabajando.

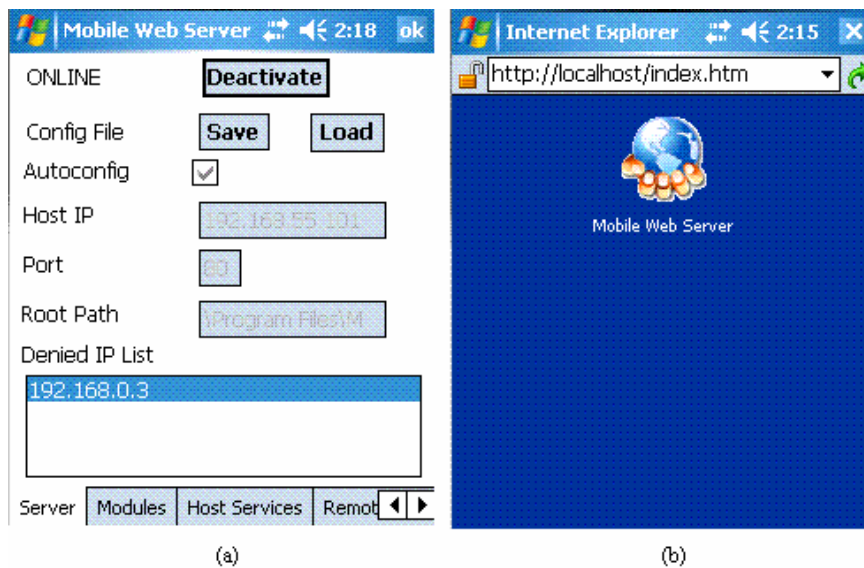


Figura 2.6. (a) El servidor ya está activo. (b) Página index del servidor activo

Este desarrollo existente está basado en Microsoft .NET Compact Framework, el cual es un subconjunto de .NET Framework, diseñado especialmente para dispositivos con recursos de hardware limitados. Este framework es un buen soporte para desarrollar servicios Web.

Las especificaciones WS-Attachments y WS-Security están implementadas en Web Services Enhancements, el cual es un paquete de definiciones que no forman parte del .NET Framework. Su equivalente para dispositivos móviles es OpenNETCF, el cual, a su vez, no forma parte del .NET Compact Framework. Se requiere que el MicroServidor sea dependiente de una sola tecnología en favor a los recursos limitados de los dispositivos móviles. Por eso, las definiciones de OpenNETCF no serán ocupadas por el MicroServidor (solamente las de .NET Compact Framework), pero serán de mucha utilidad al momento de crear clientes para servidor. Por otra parte, OpenNETCF no provee todavía una forma de exponer servicios Web, que es lo que precisamente el MicroServidor hace.

2.4. Resumen

En este capítulo se deja en claro los conceptos básicos que se necesitan para el desarrollo de este trabajo, tales como los elementos de un servicio Web, así como los estándares a aplicar. Obviamente, lo que se ha presentado en este capítulo será debidamente profundizado en el momento que se den a conocer los detalles del diseño de las soluciones creadas, sobre todo en lo que se refiere a los estándares.

Además, queda claro que durante el trabajo se deben utilizar las definiciones ya descritas, como SOAP, WSDL, DIME, etc. para así apegarse a los estándares. La idea es no “reinventar la rueda”, sino seguir los conceptos existentes para lograr una mayor integración de los desarrollos que se hagan en el futuro, tanto para dispositivos móviles como en computadores de escritorio.

Capítulo 3. WS-Attachments

Este capítulo describe el diseño e implementación del protocolo WS-Attachments, con el objeto de dar cumplimiento al primer objetivo definido. Tal como se indicó en el marco teórico, esta especificación tiene relación directa con el protocolo de encapsulación de mensajes DIME, por lo que era necesario un módulo que generara e interpretara este tipo de protocolo. Por otra parte, la aplicación existente, el servidor Web para dispositivos móviles (MicroServidor) sobre el cual había que implementar estas soluciones estándares, tenía que ser modificado para que procesara peticiones y respuestas con archivos adjuntos.

Además, el diseño final debía proveerle simplicidad al desarrollador de servicios Web, en lo que respecta a la recuperación de los datos adjuntos de un cliente y, al mismo tiempo, anexar otros datos a su respuesta. La especificación WS-Attachments no debía ser tratada solamente en el lado del servidor, pues también era necesario crear un intermediario (*proxy*) para que cualquier cliente pudiera interactuar con el MicroServidor, enviando y recibiendo archivos adjuntos de una manera simple. A continuación se detalla cómo cada uno de estos puntos fue desarrollándose, obteniéndose como resultado el estándar WS-Attachments implementado.

3.1. Adjuntos en el MicroServidor Web

WS-Attachments es parte de un estándar que busca dar interoperabilidad entre servicios Web para enviar o recibir archivos adjuntos. Tal como se ha visto, WS-Attachments se basa en DIME, pero esto no significa que no existan otros protocolos que permitan el manejo de adjuntos. Entre los protocolos alternativos para manejar datos adjuntos, se puede nombrar a SwA (SOAP with Attachments), el cual está basado en MIME; y a MTOM (Message Transmission Optimization Mechanism), un protocolo que busca mejorar la eficiencia del envío de mensajes usando XML, aunque todavía no tiene mucho soporte (lo que dificulta la interoperabilidad) [14].

Lo anterior significa que el MicroServidor Web no tiene por qué depender de la especificación WS-Attachments para manejar adjuntos. Esto quiere decir que el MicroServidor debe ser lo suficientemente modular como para aceptar peticiones con cualquier protocolo de adjuntos.

Es necesario, entonces, crear un puente de comunicación entre el MicroServidor y algún protocolo para adjuntar datos, ya implementado. Este puente puede ser representado como una interfaz¹, de tal manera que el MicroServidor se preocupe solamente de interactuar con los métodos y propiedades de la interfaz, sin saber realmente cuál es el protocolo que la implementa. A continuación se explica el diseño de esta interfaz, para luego ver cómo un módulo DIME la implementa.

3.1.1. Interfaz de Adjuntos

La Figura 3.1 muestra la firma de la interfaz, que se denomina *IAttachment*. Se puede notar que ésta ocupa listas genéricas (*ILIST*) para no depender de ningún protocolo de manejo de adjuntos

¹ Una interfaz contiene sólo las firmas de métodos, con la ventaja de que no se incluye la implementación de cada uno de ellos. La implementación la hace una clase que implementa la interfaz.

en particular. Por otra parte, los recursos que se utilizan como parámetros y retornos, son arreglos y flujos de bytes (*stream*), recursos totalmente genéricos a la hora de analizar un mensaje que sea parte del protocolo de manejo de adjuntos¹.

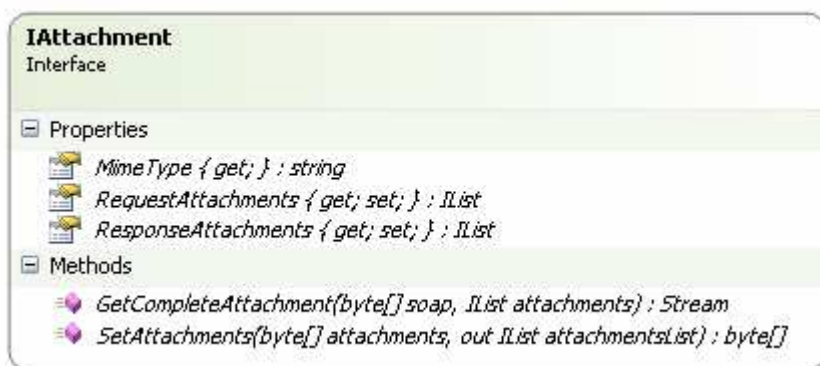


Figura 3.1. Diagrama de la interfaz IAttachment

A continuación se explican las propiedades y métodos de la interfaz:

- *MimeType* es una propiedad que sirve para obtener el nombre MIME correspondiente al protocolo de adjuntos implementado. Es necesario incluirlo en el encabezado “Content-Type” de una petición o respuesta Web.
- *RequestAttachments* y *ResponseAttachments* son listas que, respectivamente, contienen adjuntos de la petición de un cliente y la respuesta de un método Web.
- *GetCompleteAttachment* retorna un *stream* que contiene un mensaje de adjuntos, de acuerdo al protocolo implementado. Como parámetros recibe un documento SOAP (en binario), junto con un listado de los adjuntos que se desean incluir en el mensaje final.
- *SetAttachments* transforma un arreglo de bytes, que corresponde a un mensaje de adjuntos, a una lista de adjuntos. Esta lista es pasada como un parámetro “out” para que sea instanciada dentro del método. Este método retorna un arreglo de bytes para devolver información que no es un adjunto, pero que forma parte de datos anexos, como es el caso del documento SOAP.

De esta forma, el MicroServidor sabe que cuenta con ciertos métodos que son suficientes para trabajar, y los utiliza de forma apropiada. Sin embargo, la funcionalidad se logra cuando un módulo implementa *Attachment*. En la siguiente sección se explica el diseño del módulo DIME, que es el que finalmente implementa esta interfaz.

¹ Al leer “mensaje de adjuntos” entiéndase como un conjunto de bytes (ya sea en un arreglo, una lista o un *stream*) que contiene archivos e información debidamente dispuestos de acuerdo a algún protocolo de adjuntos.

3.2. Módulo DIME

Este módulo tiene tres desarrollos principales que permiten encapsular archivos adjuntos. Estos son una definición de tarjeta DIME, un generador y un intérprete de mensajes de adjuntos, los cuales se explican en las sub-secciones siguientes.

3.2.1. Tarjeta DIME

La tabla siguiente explica detalladamente el conjunto de datos que posee cada tarjeta DIME (ver Figura 2.4).

Campo	Descripción
Version (5 bits)	Indica la versión del mensaje DIME
MB (1 bit)	Indica que la tarjeta es la primera tarjeta de un mensaje
ME (1 bit)	Indica que la tarjeta es la última tarjeta de un mensaje
CF (1 bit)	Indica que el contenido del mensaje ha sido cortado
TYPE_T (4 bits)	Indica la estructura y el formato del campo TYPE
RESERVED (4 bits)	Reservado para un futuro uso
OPTIONS_LENGTH (16 bits)	Indica el largo (en bytes) del campo OPTIONS, excluyendo posibles bytes de relleno ¹
ID_LENGTH (16 bits)	Indica el largo (en bytes) del campo ID, excluyendo posibles bytes de relleno
TYPE_LENGTH (16 bits)	Indica el largo (en bytes) del campo TYPE, excluyendo posibles bytes de relleno
DATA_LENGTH (32 bits)	Indica el largo (en bytes) del campo DATA, excluyendo posibles bytes de relleno
OPTIONS	Contiene información opcional usado por el intérprete DIME
ID	Contiene un identificador único para la tarjeta; el largo de este campo se indica en ID_LENGTH
TYPE	Indica la codificación de la tarjeta (URI, MIME, etc) de acuerdo con la información del campo TYPE_T; el largo de este dato se indica en TYPE_LENGTH
DATA	Contiene el dato del adjunto, cuyo formato depende del tipo TYPE indicado en la tarjeta; el largo de este campo se indica en DATA_LENGTH

Tabla 3.1. Campos de una tarjeta DIME

Al momento de transformar estos conceptos a una clase, se hace un calce directo entre un campo de la tarjeta y una propiedad. Así surge *DimeRecord*, cuyo diagrama se aprecia en la Figura 3.2.

¹ Una restricción para los campos que no tienen un largo predefinido es que sus largos en bytes sean múltiplo de 4. Si esto no se cumple, se agregan bytes de relleno, los cuales obviamente pueden ser hasta tres.

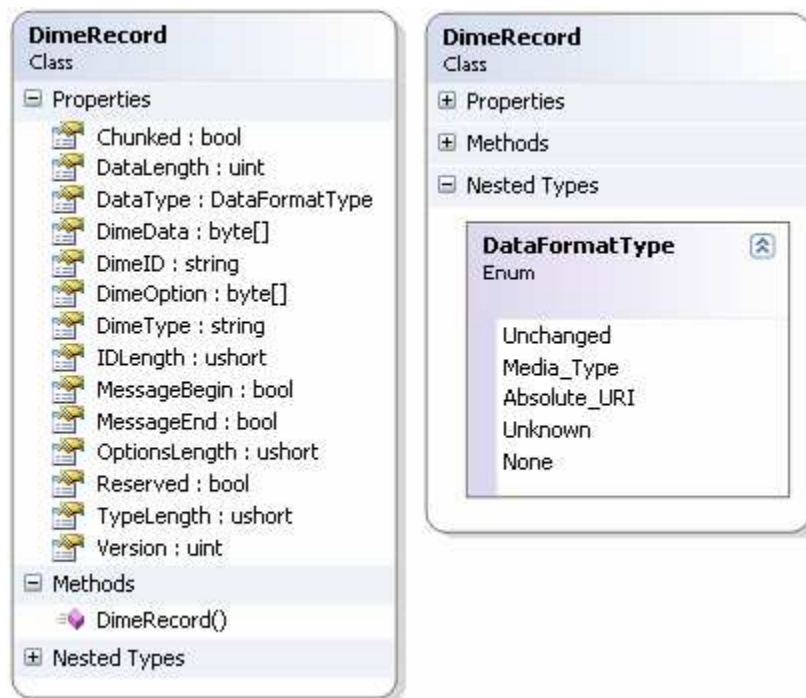


Figura 3.2. Diagrama de la clase DimeRecord

Junto con tener la estructura para una tarjeta DIME, también es necesaria una colección para guardar todas las tarjetas del mensaje. Esta colección se ilustra en la Figura 3.3.



Figura 3.3. Diagrama de la clase DimeRecordCollection

Con estas clases, ya es posible convertir un archivo a una tarjeta DIME. La forma más simple de hacer una tarjeta es instanciar un objeto *DimeRecord* y asignarle el valor *DataType*, que en la mayoría de los casos es *Media_Type*, *DimeType*, como por ejemplo “text/xml” si es un archivo XML, y *DimeData*, como por ejemplo un buffer (arreglo de bytes) de un archivo.

Luego de tener varias tarjetas, o también varios adjuntos, es necesario construir un mensaje de adjuntos, que en este caso se llama “mensaje DIME”, que incluya tantos archivos como se deseen. Para esto está el generador DIME.

3.2.2. Generador DIME

El generador DIME consiste en tomar cada una de las tarjetas ya guardadas en una colección y construir un mensaje completo, ateniéndose a la especificación DIME [4], es decir, respetando los campos que forman un mensaje DIME (ver Tabla 3.1).

Este generador determina por sí solo la primera tarjeta (*MessageBegin*), así como la última (*MessageEnd*), y revisa que las tarjetas cortadas (*Chunked*) estén correctamente configuradas; es decir, que cumpla con las siguientes condiciones, basadas en la especificación DIME:

- La primera tarjeta cortada no tenga en el campo `TYPE_T` el valor `Unchanged`, pues este valor es reservado para tarjetas cortadas que no sean la primera (tarjetas cortadas hijas).
- Las tarjetas cortadas hijas tengan su campo `TYPE_T` con el valor `Unchanged`, por la razón expuesta en el punto anterior.
- Las tarjetas cortadas hijas no posean información en los campos `TYPE` e `ID`, es decir, que el valor de los campos `TYPE_LENGTH` e `ID_LENGTH` sean cero.

Si alguna de las restricciones anteriores no se cumple, entonces el generador no construye el mensaje, lanzando un error (*exception*). En el caso que una tarjeta no tenga un identificador (ID), y a la vez no sea una tarjeta cortada hija, entonces el generador le asigna un identificador único global (GUID).



Figura 3.4. Diagrama de la clase DimeGenerator

El generador escribe de forma secuencial, en un *stream*, los datos correspondientes a cada tarjeta que va capturando de una colección. En la Figura 3.4, se muestran los métodos del generador. Entre estos métodos, algunos tienen como prefijo `Write`, los cuales son los que interactúan directamente con el *stream*, mientras que los métodos *ConfirmChunking* y *ConfirmChildChunk* son lo que verifican las condiciones en las tarjetas cortadas.

La forma común de obtener un mensaje DIME es instanciar un objeto *DimeGenerator*, ir agregando tantas tarjetas DIME como se deseen con el método *AddDimeRecord* y finalmente invocar *GetStream*. Este último método es el encargado de comenzar a construir el mensaje, retornándolo en un *stream*, para enviar su contenido ya sea como una petición o respuesta, según el contexto del proceso.

El generador DIME es vital para la implementación de WS-Attachments, de igual forma como debe serlo un intérprete o, en otras palabras, un proceso que sea capaz de transformar un mensaje DIME a tarjetas; es decir, el trabajo inverso al generador. La siguiente sub-sección explica dicho proceso.

3.2.3. Intérprete DIME

Al igual como desde una colección de tarjetas DIME se genera un mensaje DIME, también se necesita obtener las tarjetas a partir de un mensaje antes generado. Por esto, nace la definición *DimeParser*, cuyo diagrama se ilustra en la Figura 3.5.

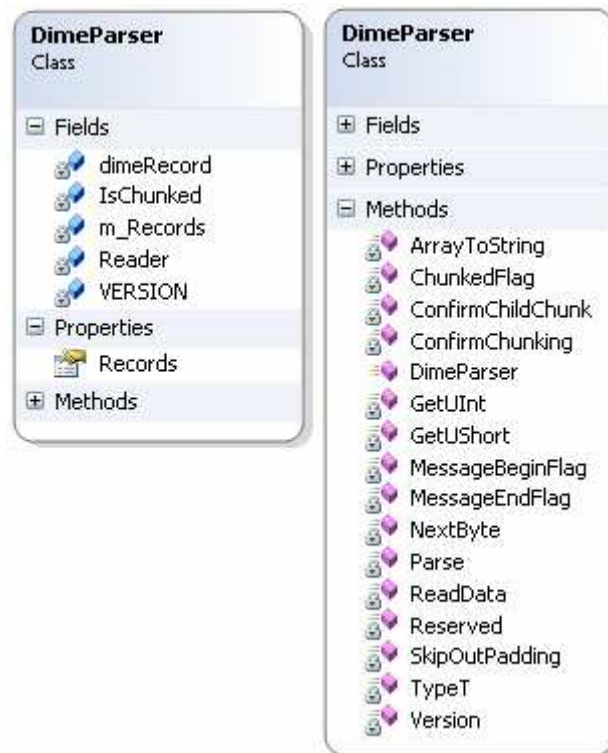


Figura 3.5. Diagrama de la clase *DimeParser*

A medida que el intérprete recorre el mensaje, debe ir guardando las tarjetas que encuentra en una colección, para que al final del análisis se obtenga aquella colección con la propiedad *Records*. Sin embargo, la especificación DIME menciona que aunque exista sólo una tarjeta mal confeccionada, el mensaje entero debe ser rechazado. Por lo tanto, aparte de validar las mismas restricciones que valida el generador con respecto a las tarjetas cortadas, el intérprete también debe ceñirse a las siguientes:

- Todas las tarjetas deben tener la misma versión asignada. Como la especificación actual de DIME es la versión 1, *DimeParser* acepta solamente mensajes con tarjetas que indiquen versión 1.
- El campo *RESERVED* debe tener como contenido ceros.

- No debe existir ningún tipo de datos después de una tarjeta marcada como última (*MessageEnd*).

En caso que una restricción no se cumpla, el mensaje deja de analizarse, lanzando un error (*exception*). Las definiciones que se han presentado: *DimeRecord*, *DimeGenerator* y *DimeParser*, forman el núcleo para manejar archivos adjuntos. Sin embargo, para unir esta capacidad con las funciones de un MicroServidor, se debe implementar la interfaz *IAttachment*, que se describió en 3.1.1. “Interfaz de Adjuntos”. La siguiente sub-sección muestra cómo se utilizan los recursos DIME ya descritos para implementar la interfaz *IAttachment*.

3.2.4. Atributo DIME

Un atributo es información que se puede añadir a los metadatos asociados al código de un módulo. Es como una información incrustada en un programa, la cual se puede consultar en tiempo de ejecución mediante una técnica denominada reflexión [6]. Más adelante se explicará con más detalle cómo se le añade el atributo DIME a un servicio Web, y cómo el MicroServidor utiliza la reflexión para trabajar con un atributo añadido a un servicio Web.

Lo importante a tratar en esta sección es cómo este atributo implementa la interfaz *IAttachment*. La Figura 3.6 muestra, como es obvio al implementar una interfaz, que la clase *DimeAttribute* posee la definición de las mismas propiedades y métodos que *IAttachment*, con un método anexo que tiene que ver con la extracción de la información SOAP de un mensaje DIME.

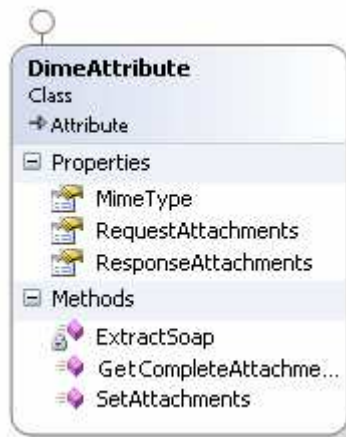


Figura 3.6. Diagrama de la clase *DimeAttribute*

A continuación se especifican algunos detalles de la implementación de la interfaz.

- El *MimeType* de un mensaje DIME es “application/dime”, el cual corresponde al valor Content-Type de un encabezado Web.
- Las listas *RequestAttachments* y *ResponseAttachment* son listas estáticas, para que así, tanto el MicroServidor como un método Web puedan compartir el contenido de ellas.

- *GetCompleteAttachments* utiliza *DimeGenerator* para generar el mensaje de adjuntos, dejando el mensaje SOAP como primera tarjeta, para después agregar los adjuntos de la lista en el mismo orden en el que se encuentran. Este método además corta automáticamente las tarjetas si es necesario, de manera que este trabajo no sea parte del *MicroServidor* ni de un método Web.
- *SetAttachments* utiliza *DimeParser* para analizar un mensaje de adjuntos, dejando los datos adjuntos en la lista referenciada como argumento. Este método además une automáticamente las tarjetas cortadas, de manera que este trabajo tampoco sea parte del *MicroServidor* ni de un método Web.
- Dado que la especificación DIME indica que la información SOAP corresponde a la primera tarjeta del mensaje DIME, *ExtractSoap* extrae esta información dejando a disposición de la aplicación solamente los archivos adjuntos.

De esta forma, con la clase *DimeAttribute*, el módulo *DIME* queda listo para que un *MicroServidor* procese adjuntos de dicho protocolo. Ahora, para esto fue necesario hacer cambios en la implementación existente del *MicroServidor*, lo que se detalla en la siguiente sección.

3.3. Modificaciones en el MicroServidor Web

Un requisito importante a tener en cuenta al momento de modificar el *MicroServidor* Web era que la forma de procesar adjuntos se comportara como un módulo o un plug-in, de manera de no realizar cambios que significaran reingeniería cada vez que se implantara un protocolo nuevo de adjuntos. En esta sección se describe cómo el *MicroServidor* logra este requisito, quedando totalmente dispuesto para enviar y recibir adjuntos.

3.3.1. Recepción de Datos de un Cliente

La Figura 3.7 muestra un leve, pero importante, cambio en la clase *Request*. Originalmente se asumía que los datos enviados por un cliente eran solamente una secuencia de caracteres (*string*) previamente codificadas, por lo que siempre que el servidor recibía datos, los transformaba inmediatamente a *string*. Esta forma de proceder era suficiente para procesar peticiones cuyos datos venían desde la misma URL (ubicación) del servidor o en un archivo SOAP (XML).



Figura 3.7. (a) Antigua definición de Request. (b) Nueva definición de Request

Ahora que el servidor debe estar preparado para aceptar adjuntos, no es lógico que los datos recibidos sean transformados inmediatamente a *string*. Por esta razón, la petición original de un cliente debe mantener una lista de bytes correspondiente a lo recibido, exceptuando el encabezado Web de la petición, el cual se asume que corresponde a una secuencia de caracteres.

3.3.2. Módulo Attachments

El uso de módulos en el MicroServidor no es nuevo. Como se explicó en la sección 2.3 “Trabajo Relacionado”, existen los módulos SOAP y Http. La forma de cargar estos módulos es mediante un archivo XML de configuración del MicroServidor llamado *config.web*, el cual es analizado por la clase *WebServerConfiguration*, que deja los datos en memoria para su uso inmediato.

Aprovechando esta idea existente para cargar módulos, la forma de agregar un nuevo módulo de adjuntos, como por ejemplo el ya descrito módulo *DIME*, es totalmente equivalente. La Figura 3.8 muestra un ejemplo del archivo *config.web*, el cual bajo la nueva etiqueta “attachments” lista un solo módulo. Sin embargo, siguiendo el mismo esquema, pueden agregarse más.

```
<configuration>
...
<modules>
  <module classname="HttpFileProcessor" modulepath="HttpModule.dll">
    <mimeType mime="image/jpeg" extension="jpg"/>
    ...
  </module>
  <module classname="SoapProcessor" modulepath="SoapModule.dll">
    <mimeType mime="text/xml" extension="asmx"/>
  </module>
</modules>
<attachments>
  <module classname="DimeAttribute" modulepath="AttachmentModule.dll">
    <mimeType mime="application/dime" />
  </module>
</attachments>
</configuration>
```

Figura 3.8. Ejemplo del archivo de configuración del MicroServidor

Igualmente, en la clase *WebServerConfiguration* tuvo que crearse la definición *AttachmentModule* (ver Figura 3.9), debido a que no se podía utilizar directamente la forma de cargar los módulos SOAP y Http (*WebModule*). Esto se debía a que ellos son referenciados por la interfaz *IHttpModule*, que un módulo de adjuntos no debe implementar.

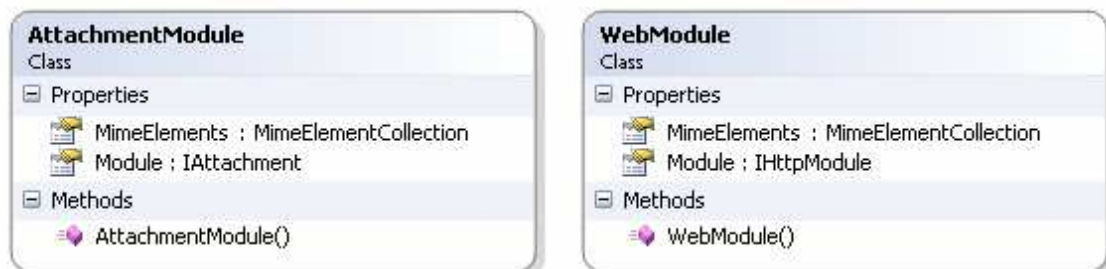


Figura 3.9. Diagrama de las clases AttachmentModule y WebModule

3.3.3. Módulo SOAP

El módulo SOAP tuvo que ser modificado en sólo dos de las 12 clases que lo componen. Estas clases son:

- *SoapProcessor*, el cual da inicio a la solicitud SOAP de un cliente, obteniendo los datos provenientes en el archivo SOAP.
- *SoapDetails*, el cual hace una invocación al servicio Web requerido por un cliente.

Las modificaciones realizadas no fueron sustanciales, más bien corresponden a nuevas decisiones frente a los datos recibidos o por enviar. Una forma útil de ilustrar estas nuevas decisiones es mediante un diagrama de secuencia, el cual puede apreciarse en la Figura 3.10.

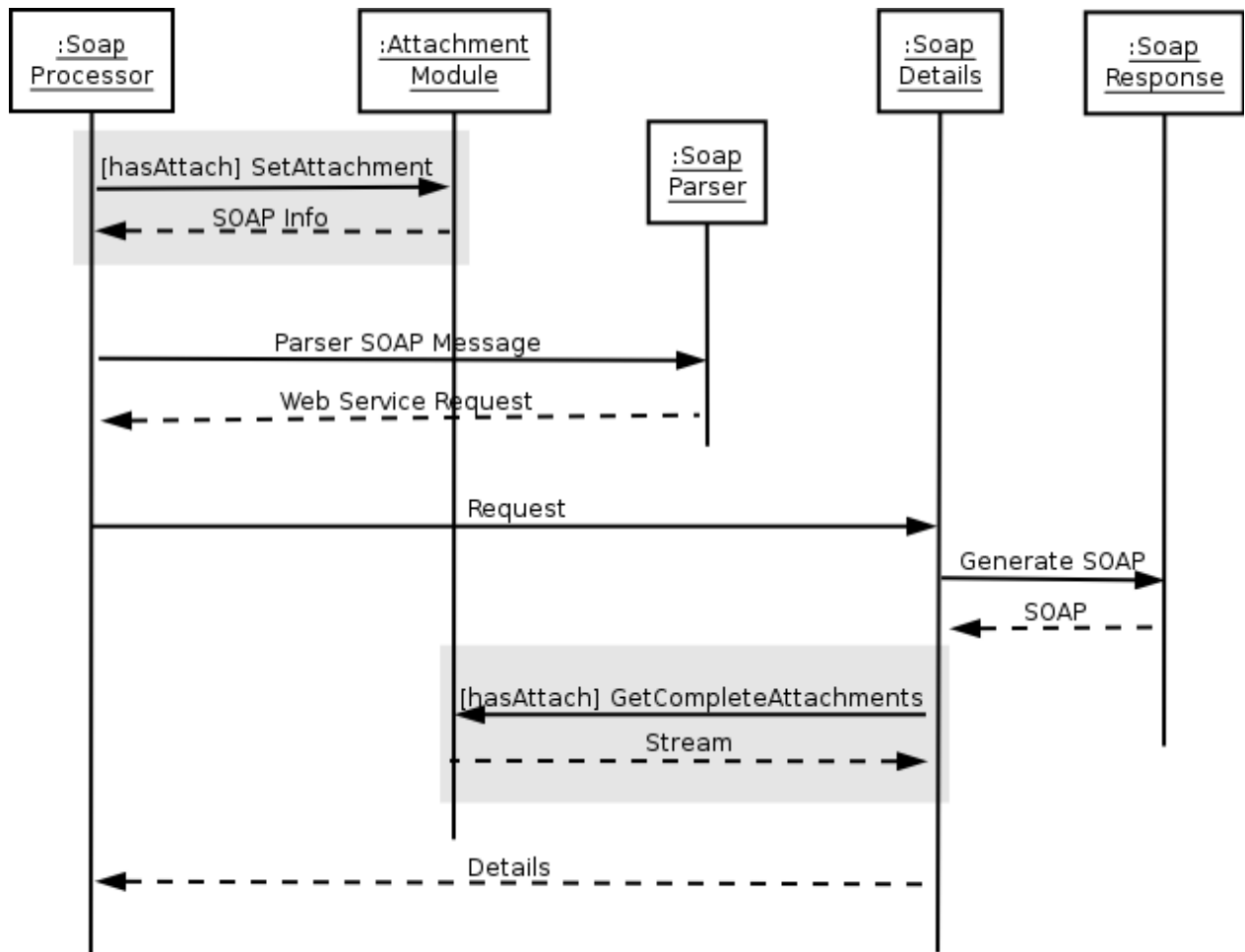


Figura 3.10. Diagrama de Secuencia módulo SOAP.

Sutilmente se han colocado sobre el diagrama recuadros grises que indican el cambio efectuado en las dos clases antes mencionadas. Nótese que ambas clases tienen una modificación similar, pues primero se verifica la existencia de adjuntos y, en el caso que existan, se procesan con un módulo *Attachment*.

La forma de preguntar si existen adjuntos en la petición o respuesta se hace utilizando estrategias diferentes, dependiendo del origen de los adjuntos. Estas estrategias son las siguientes:

- En *SoapProcessor*, el origen de los adjuntos es el cliente, por lo tanto, entre los encabezados Web de la petición del cliente, éste indicó el tipo de contenido de sus datos. En el caso de DIME, el cliente debe indicar el encabezado “Content-Type: application/dime”. Con este encabezado, y utilizando la información en memoria de *WebServerConfiguration*, al servidor le resta obtener un módulo que le permita manejar el tipo de contenido indicado, y así entregar los datos del cliente a aquel módulo. En caso de que el módulo no exista, los datos se tratan como una simple petición SOAP.
- En *SoapDetails*, el origen de los datos es un método de un servicio Web. Si el método maneja adjuntos, debe tener asignado obligatoriamente el atributo *DIME* (véase 3.4.2 Adjuntos en un servicio Web). Gracias a la técnica *reflexión*, en tiempo de ejecución se pueden determinar los atributos de un método; en particular, el MicroServidor puede saber si el método Web que invocará posee el atributo *DIME*. Si es así, toma el atributo y recupera los adjuntos de la respuesta del método Web.

Un detalle importante que hace posible que el servidor pueda asignar y recuperar adjuntos del módulo *Attachments*, es que este módulo tiene implementado una lista de adjuntos que es estática; es decir, es una lista cuyos datos pertenecen al módulo y no a un objeto de éste. Esto significa que los datos de la lista se mantienen siempre, aunque se inicialice el módulo muchas veces. Obviamente, el servidor se encarga de borrar los adjuntos de la lista, una vez que ya no los necesita.

Finalmente, el MicroServidor tiene la capacidad de procesar adjuntos, alcanzando el primer objetivo del trabajo de título. Aun así, queda por explicar cómo hacer un servicio Web que esté implementado de acuerdo a los procedimientos del MicroServidor y, además, cómo un cliente puede hacer sus peticiones enviando adjuntos.

3.4. Usando WS-Attachments

En esta sección, se da por concluido el trabajo sobre el MicroServidor para explicar cómo se realiza un ciclo completo de comunicación, entre un cliente y un servicio Web, teniendo ahora como alternativa, el envío de datos adjuntos.

3.4.1. Crear un adjunto

Hasta el momento, la única manera de formar un adjunto es tener el binario del dato a adjuntar y asignarlo a un objeto *DimeRecord*. Para flexibilizar la forma de hacer un adjunto, se creó una derivación de *DimeRecord* llamada *DimeAttachment*. La Figura 3.11 muestra que para construir un adjunto ahora también puede asignarse una ruta de localización de un archivo o un *stream*, haciendo más amigable trabajar con adjuntos.



Figura 3.11. Diagrama de la clase DimeAttachment

3.4.2. Adjuntos en un servicio Web

Para hacer uso de adjuntos en un método de un servicio Web, al método debe asignársele el atributo *DIME*, y luego, en la implementación del método, se debe hacer uso de las listas de *DimeAttribute* agregando elementos *DimeAttachment* a ellas. Para ver la simpleza del procedimiento, la Figura 3.12 muestra una porción de código de un método Web.

```

14 [WebMethod]
15 [Dime]
16 public string SampleMethod()
17 {
18     DimeAttribute att = new DimeAttribute();
19
20     foreach (DimeAttachment da in att.RequestAttachments)
21         Process(da);
22     //...
23     att.ResponseAttachments.Add(new DimeAttachment("file1.xml",
24                                                     "text/xml"));
25     att.ResponseAttachments.Add(new DimeAttachment("file2.xml",
26                                                     "text/xml"));
27     return "OK";
28 }

```

Figura 3.12. Ejemplo de implementación de un método Web que usa adjuntos

En las líneas de código 14 y 15 están los atributos del método *SampleMethod*. Con estos atributos, el MicroServidor Web puede saber que el método es un método Web y que usa DIME como protocolo de adjuntos. Con un objeto *DimeAttribute* se acceden a las listas *RequestAttachments* y *ResponseAttachments* con las cuales se puede obtener los adjuntos del cliente y agregar otros a la respuesta.

3.4.3. Adjuntos en un cliente

Existen tecnologías que permiten tener un intermediario (*proxy*) que construya peticiones de un cliente, usando el protocolo de adjuntos DIME. Sin embargo, depender de tecnologías que no sean parte de Microsoft Compact .Net Framework (CF), que es lo utilizado hasta el momento, no es lo ideal. Por esta razón, se decidió aprovechar una directiva que provee el CF, la cual permite modificar los encabezados Web y el contenido de una petición en el mismo proxy que las genera.

La Figura 3.13 muestra, en primer lugar, una definición para que un *proxy* se comporte de acuerdo al protocolo DIME. Extiende *SoapHttpClientProtocol*, que es el que genera e interpreta mensajes SOAP, y se le agregan las listas pertenecientes a *DimeAttribute*.



Figura 3.13. Diagrama de la clase *MobileWebServerClientProtocol*

Ahora que un cliente tiene una lista para agregar adjuntos a su petición, se necesita modificar el archivo SOAP generado por *SoapHttpClientProtocol*. Para esto existe la directiva del CF llamada *SoapExtension*, a través de la cual se tiene acceso al mensaje SOAP en sus diferentes etapas de creación y recepción en un *proxy*.

En la etapa de creación, cuando el mensaje SOAP está listo para ser enviado a un servidor, este mensaje es cambiado a un mensaje DIME. Para esto, se tiene a disposición el mensaje SOAP y la lista *RequestAttachments*, recursos suficientes para invocar el método *GetCompleteAttachments* de *DimeAttribute* con el cual se obtiene un mensaje de adjuntos para ser enviado.

Por otra parte, en la etapa de recepción, justo antes que un mensaje esté listo para analizarse, se invoca el método *SetAttachments* de *DimeAttribute*, para obtener los adjuntos de la respuesta y dejar solamente el mensaje SOAP para que sea analizado.

Estas modificaciones se agrupan en una definición llamada *MobileWebServerExtension*, la cual a su vez debe asignarse como atributo a un método que quiera utilizar adjuntos en sus peticiones. De esta forma, son tres los pasos que se deben hacer para tener un cliente que utilice el protocolo de adjuntos:

1. Crear un *proxy* que utilice el protocolo *MobileWebServerClientProtocol*.
2. Crear un método en el *proxy* que invoque un servicio Web. Si el método envía y/o recibe adjuntos, agregarle el atributo *MobileWebServerExtension*.
3. Crear el cliente, utilizando las listas de adjuntos que el *proxy* provee, con la misma simplicidad con la que se puede hacer un método Web (ver Figura 3.12).

3.5. Resumen

El trabajo realizado en el MicroServidor Web sobre WS-Attachments deja un fundamento para que nuevos protocolos de adjuntos sean implementados, sin la necesidad de re-codificar lo existente en el MicroServidor. Esto es así porque se supo dejar totalmente independiente el

módulo DIME, que es el que construye mensajes DIME, y el MicroServidor. De hecho, la independencia funcional del módulo DIME quedó demostrada a la hora de implementar *MobileWebServerExtension*, pues dicha extensión está totalmente fuera del contexto de un servidor. Los pasos a seguir para agregar un nuevo protocolo de adjuntos al MicroServidor pueden resumirse en tres:

1. Crear un módulo que genere e interprete mensajes del nuevo protocolo de adjuntos.
2. Agregar al módulo un atributo que implemente la interfaz *IAttachment*.
3. Agregar al archivo de configuración del MicroServidor. los datos del nuevo módulo creado.

Además, el diseño realizado permite que un desarrollador implemente con facilidad métodos Web y clientes, dejando en claro que el objetivo a lograr no fue tan sólo dejar un MicroServidor funcional, sino dar la oportunidad de aprovechar bien las capacidades que este servidor tiene, facilitando su uso a futuros desarrolladores.

De esta forma, el MicroServidor avanza un paso más a la integración que se busca en los servicios Web, logrando interoperabilidad, ya no tan sólo al procesar peticiones y respuestas SOAP, sino también al aplicar el estándar WS-Attachments como protocolo de adjuntos.

Capítulo 4. WS-Security

La seguridad en servicios Web busca cubrir muchos aspectos, entre los cuales se destacan la autenticación de usuarios, y la integridad y confidencialidad de los mensajes enviados por un cliente hacia un servidor. Dado que el MicroServidor no tenía ningún tipo de seguridad asociado, fue necesario añadirle una capa de seguridad que marcara el punto de partida para que, de forma incremental, fueran agregándose más capas en un futuro.

Este capítulo trata precisamente de cómo el MicroServidor autentica usuarios, de tal manera de asegurar que sus servicios están siendo utilizados o invocados por clientes autorizados. A pesar de que no es una capa que logra integridad ni confidencialidad de los mensajes, se dará un especial enfoque a cómo lo realizado deja las puertas abiertas para implementar nuevas capas de seguridad, sin la necesidad de rediseñar el flujo de decisiones del MicroServidor.

Para empezar, se verá cuáles son los datos que el estándar WS-Security dispone para que un servidor autentique usuarios. Luego se mostrará el diseño creado en base al estándar, para luego demostrar cómo el MicroServidor termina autenticando usuarios. Junto con esto, este capítulo describe el desarrollo de un cliente, similar a lo que se hizo con Attachments.

4.1. Datos de Autenticación

Para autenticar usuarios, WS-Security provee datos definidos en indicadores o *tokens* de seguridad. Existen dos tipos de *tokens*: *BinarySecurityToken* y *UsernameToken*. Debido a que entre los objetivos de incluir WS-Security en el MicroServidor Web está el abrir paso a la seguridad para futuras implementaciones, se optó por desarrollar *UsernameToken*, teniendo en cuenta que el trabajo tenía que quedar lo suficientemente extensible para *BinarySecurityToken*, con el cual se autentica usuarios usando certificados X.509 y vales Kerberos¹. La siguiente tabla muestra cómo se define un *UsernameToken* según el estándar WS-Security [24]:

Campo	Descripción.
Id	Identificador del <i>Token</i> .
Username	Nombre de usuario del cliente
Password	Clave secreta del usuario. Cuando corresponde a un valor hash, se encuentra en base 64.
Tipo de password	Existen tres tipos de passwords: <i>PasswordDigest</i> , cuando el campo Password corresponde a un valor hash; <i>PasswordText</i> , cuando el Password es simplemente su valor plano (original); y <i>None</i> , cuando el password provisto debe ser ignorado.
Nonce	Cadena de caracteres que representa a un número cualquiera convertido a base 64, usado para generar un valor hash a partir de la contraseña original
Created	Fecha de creación del <i>token</i> de seguridad, en formato UTC (Coordinated Universal Time)

Tabla 4.1. Definiciones de UsernameToken

¹ Debido a los objetivos planteados, y para no desviar la atención del diseño y desarrollo realizado, no se analizará en profundidad en qué consisten los certificados X.509 y vales Kerberos.

En particular, el campo *Nonce* es de utilidad solamente cuando la clave secreta es enviada como un valor hash. Con la existencia de este campo, cada vez que el mensaje SOAP viaja hacia el servidor, el valor hash de la clave secreta del usuario nunca es la misma, pues siempre se genera un *Nonce* diferente (aleatorio) con el cual se obtiene a su vez el valor hash de la clave secreta. Esto evita que una persona malintencionada capture el mensaje para reutilizar la clave secreta. WS-Security detalla que el password como hash, se obtiene como se indica en la Figura 4.1:

```
PASSWORD_HASH = BASE64 (SHA-1 (PASSWORD_PLAIN + NONCE + CREATED))
```

Figura 4.1. Construcción del valor hash de una clave secreta

De esta forma, se tienen los conocimientos suficientes para que un servidor pueda autenticar al cliente. A su vez, el mensaje SOAP agrega un *Timestamp* a la seguridad global del mensaje, el cual tiene los datos mostrados en la Tabla 4.2:

Campo	Descripción.
Id	Identificador del <i>Timestamp</i> .
Created	Fecha de creación de la seguridad en el mensaje.
Expires	Fecha en los que todos los <i>tokens</i> definidos en el mensaje SOAP pierden su validez. En un mensaje pueden venir más de un <i>token</i> , y no necesariamente del mismo tipo.

Tabla 4.2. Definiciones de Timestamp

La Figura 4.2 muestra un fragmento de un mensaje SOAP que utiliza un *UsernameToken* para autenticar.

```
<wsse:Security>
  <wsu:Timestamp wsu:Id=" d521b9f7-c0cc-4520-b5c9-9ea42ad0e77a">
    <wsu:Created>2006-10-23T15:15:33Z</wsu:Created>
    <wsu:Expires>2006-10-23T15:20:33Z</wsu:Expires>
  </wsu:Timestamp>
  <wsse:UsernameToken wsu:Id="cff20dc4-f5bd-4501-a52d-cda4a2f3682c">
    <wsse:Username>user</wsse:Username>
    <wsse:Password Type="PasswordText">secret</wsse:Password>
    <wsse:Nonce>g0HNnoBv1pM2Br+YD0IXIQ==</wsse:Nonce>
    <wsu:Created>2006-10-23T15:15:33Z</wsu:Created>
  </wsse:UsernameToken>
</wsse:Security>
```

Figura 4.2. Ejemplo de WS-Security usando Usenametoken

A continuación se muestra cómo, dadas las especificaciones anteriores, nace el diseño para implementar *tokens*.

4.2. Módulo Seguridad

Debido que un *token*, independiente de su tipo, es definido solamente en un fragmento SOAP, que a la vez es un fragmento XML, es natural pensar que un *token* puede ser reconstruido a un objeto a partir del fragmento XML y viceversa. Es decir, que a partir de un objeto se obtenga un

fragmento XML con su respectiva definición de *token*. Este concepto se denomina Serialización en XML, y es precisamente esta técnica la que se ocupará para trasladar los datos de un *token* en el mensaje SOAP, a un objeto dentro del MicroServidor.

Para lograr esto, en el .NET Compact Framework existe una interfaz llamada *IXmlSerializable*, que permite realizar esta operación. Esta interfaz define los métodos apropiados para que un objeto se construya a partir de un fragmento XML y viceversa. Por lo tanto, un factor común para cualquier *token* es que sea serializable. Además, otro factor común que tienen los *token* es el *Timestamp* de seguridad. De esta forma, nace la interfaz *ISecurityToken* (Figura 4.3), con la cual deben implementarse todos los *token* de seguridad que se requieran.



Figura 4.3. Diagrama de la interfaz ISecurityToken

De esta manera, al implementar el objeto *UsernameToken*, se debe implementar la interfaz *ISecurityToken*, tal como muestra la Figura 4.4.



Figura 4.4. Diagrama de la clase UsernameToken

Se nota en la Figura 4.4 que las propiedades calzan directamente con todos los datos mencionados en 4.1 “Datos de Autenticación”. Entre los métodos, los que corresponden a la serialización XML son los siguientes:

- *ReadXml*, para construir un objeto *UsernameToken*, a partir de un XML.
- *WriteXml*, para generar un XML a partir un objeto *UsernameToken*.
- *GetSchema*, que provee el esquema de los XMLs leídos y generados.

El método público *Authenticate* sirve para comparar la clave secreta del *UsernameToken*, con una provista por un medio externo, como lo es el *MicroServidor*. En el caso que la clave tenga un valor hash, el método *GetDigest* ayuda a reconstruir el hash a partir del password provisto por el medio externo, el *Nonce* y fecha de creación del *UsernameToken*. Nótese que el password puede ser un valor hash, lo cual significa que no existe un proceso inverso que entregue el valor original del password, razón por la cual debe reconstruirse, tal como se muestra en la Figura 4.1. Ahora que se tiene el diseño del *UsernameToken*, se continúa mostrando cómo el *MicroServidor* ocupa tal diseño para aplicar seguridad en sus servicios.

4.3. Seguridad en el Microservidor Web

Para lograr aplicar seguridad en el *MicroServidor*, se necesitaron varias modificaciones y nuevas configuraciones, las cuales se detallan a continuación.

4.3.1. Herramienta de Seguridad

Un servicio importante que el *MicroServidor* necesita, es aquel que mantiene un repositorio de datos en donde se encuentra el listado con los usuarios y sus respectivas contraseñas. Este repositorio puede ser un archivo de texto, un archivo XML o una base de datos. Para que el *MicroServidor* no se relacione directamente con alguno de estos recursos se creó la interfaz *ISecurityUtils* (Figura 4.5), en la que se definen los métodos que el *MicroServidor* necesita, dejando aparte lo que es su implementación. De esta forma, el *MicroServidor* sólo se preocupa de invocar aquellos métodos, sin saber si los datos están siendo obtenidos y/o guardados en un archivo XML o en una base de datos.

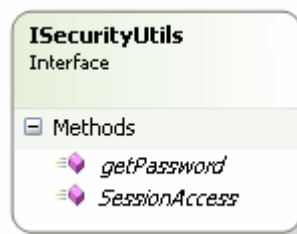


Figura 4.5. Diagrama de la interfaz *ISecurityUtils*

La idea es que con el tiempo, la cantidad de métodos definidos en esta interfaz aumenten de acuerdo a las futuras necesidades del *MicroServidor*. Por el momento, de acuerdo al *token* implementado, se necesita el método *getPassword*, para que el *MicroServidor* obtenga el password que el usuario tiene asignado, que después será comparado por el provisto por un cliente¹.

Aprovechando la flexibilidad que se obtiene al crear esta interfaz, cada servicio Web puede proveerle al *MicroServidor* su propia herramienta de seguridad, indicándola en el archivo de configuración del servicio Web correspondiente.

¹ Con respecto a la definición *SessionAccess* véase 5.7 Trabajo a Futuro

4.3.2. Archivo de Configuración de un Servicio Web

En el desarrollo actual del MicroServidor, éste verifica si el servicio Web que el cliente requiere tiene un archivo de configuración asociado. Esta capacidad fue aprovechada para agregar datos de seguridad al servicio Web. En primer lugar, en el archivo de configuración de un servicio Web, el cual es un archivo XML, se agregó una nueva etiqueta, llamada “security”. Esta etiqueta sirve como un indicador que determina si el servicio Web usa seguridad. A su vez, se indica también la herramienta de seguridad asociada.

Un ejemplo de archivo de configuración de un servicio Web es el que se ilustra en la Figura 4.6. Se aprecia en la línea 3 que la herramienta de seguridad es la clase “MWSUtility.MyUtility” que se encuentra en el ensamblado MWSUtility.dll. Aquella clase debe ser una que implemente la interfaz *ISecurityUtils*. En la línea 4 se indica el *token* que el servicio Web acepta.

```
1 <system.web>
2   <webservices/>
3   <security Utility="MWSUtility.MyUtility, MWSUtility">
4     <usernameToken/>
5   </security>
6 </system.web>
```

Figura 4.6. Archivo de configuración de un servicio Web

De esta forma, se obtienen las siguientes ventajas:

- La seguridad de un servicio Web se activa o desactiva agregando o quitando la etiqueta “security” en el archivo de configuración del servicio Web.
- Cada servicio Web provee al MicroServidor su propia herramienta de seguridad. Esto quiere decir, por ejemplo, que cada servicio Web puede tener su propio repositorio de usuarios que estén autorizados a invocar el correspondiente servicio Web. En el caso de omitir el atributo “Utility”, el MicroServidor utilizará una herramienta por omisión y, por lo tanto, un repositorio global.
- Cada servicio Web decide el *token* (o los *tokens*) de autenticación que acepta. Por ahora, sólo está implementado *UsernameToken*.

Ahora que el archivo de configuración de un servicio Web ha cambiado, el MicroServidor, y en particular el módulo SOAP, debe modificarse para leer los nuevos elementos.

4.3.3. Modificaciones en el Módulo SOAP

Varias modificaciones se hicieron en este módulo. Para empezar, la clase *ClassDetails* es la encargada de tener los datos de la clase equivalente al servicio Web a invocar, tales como su namespace y su nombre. Es útil agregar entonces en *ClassDetails* los datos del archivo de configuración correspondiente. Para ello se creó la clase *WebServiceConfiguration* que obtiene todos los datos que rescate del archivo de configuración (ver Figura 4.7).

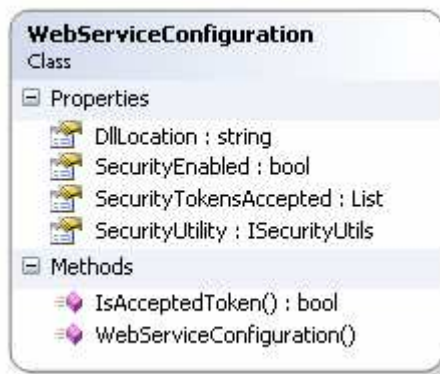


Figura 4.7. Diagrama de la clase `WebServiceConfiguration`

Al momento de leer el archivo de configuración de un servicio Web y al notar que se encuentra la etiqueta “security”, se cambia el valor de `SecurityEnabled` a verdadero, se instancia en `SecurityUtility` la herramienta de seguridad indicada en el atributo “Utility”, y finalmente se agregan a la lista `SecurityTokensAccepted` todos los `tokens` que el servicio Web acepta para autenticar.

Tal como se ha explicado anteriormente, la información de un `token` viene en el mensaje SOAP que corresponde a la petición de un cliente. WS-Security detalla que esta información debe ser parte del encabezado SOAP. De esta forma, se hizo necesario que la clase `SoapParser` procesara los encabezados del mensaje, pues hasta el momento, sólo procesaba el cuerpo de éste.

La modificación en `SoapParser` consistió en que se guardara el encabezado SOAP del mensaje, para poder utilizarlo en un proceso posterior; específicamente cuando el MicroServidor ya tiene listo el servicio Web a invocar, junto con su configuración. En ese momento, justo antes de invocar un método de un servicio Web, en `SoapDetails`, se comienzan a utilizar todos los recursos previamente descritos en este capítulo. El procedimiento es el siguiente:

1. Verificar que el servicio Web tenga indicado seguridad en su configuración.
2. Verificar que la herramienta de seguridad exista.
3. Obtener el encabezado SOAP de seguridad que `SoapParser` guardó.
4. Leer el encabezado SOAP de seguridad encontrando `tokens`.
5. Al encontrar uno, preguntar si el `token` está entre los aceptados en la configuración del servicio Web.
6. Obtener un objeto del `token` (como `UsernameToken`) usando el fragmento SOAP correspondiente a la definición del `token` que se está leyendo.
7. En el caso de `UsernameToken`, obtener el password que provee la herramienta de seguridad, para finalmente autenticar.

El diagrama de secuencia de la Figura 4.8 muestra cómo queda el módulo SOAP con las modificaciones. Al igual que en el diagrama de secuencias del capítulo anterior, se han colocado sutilmente recuadros grises que indican los cambios principales en el MicroServidor.

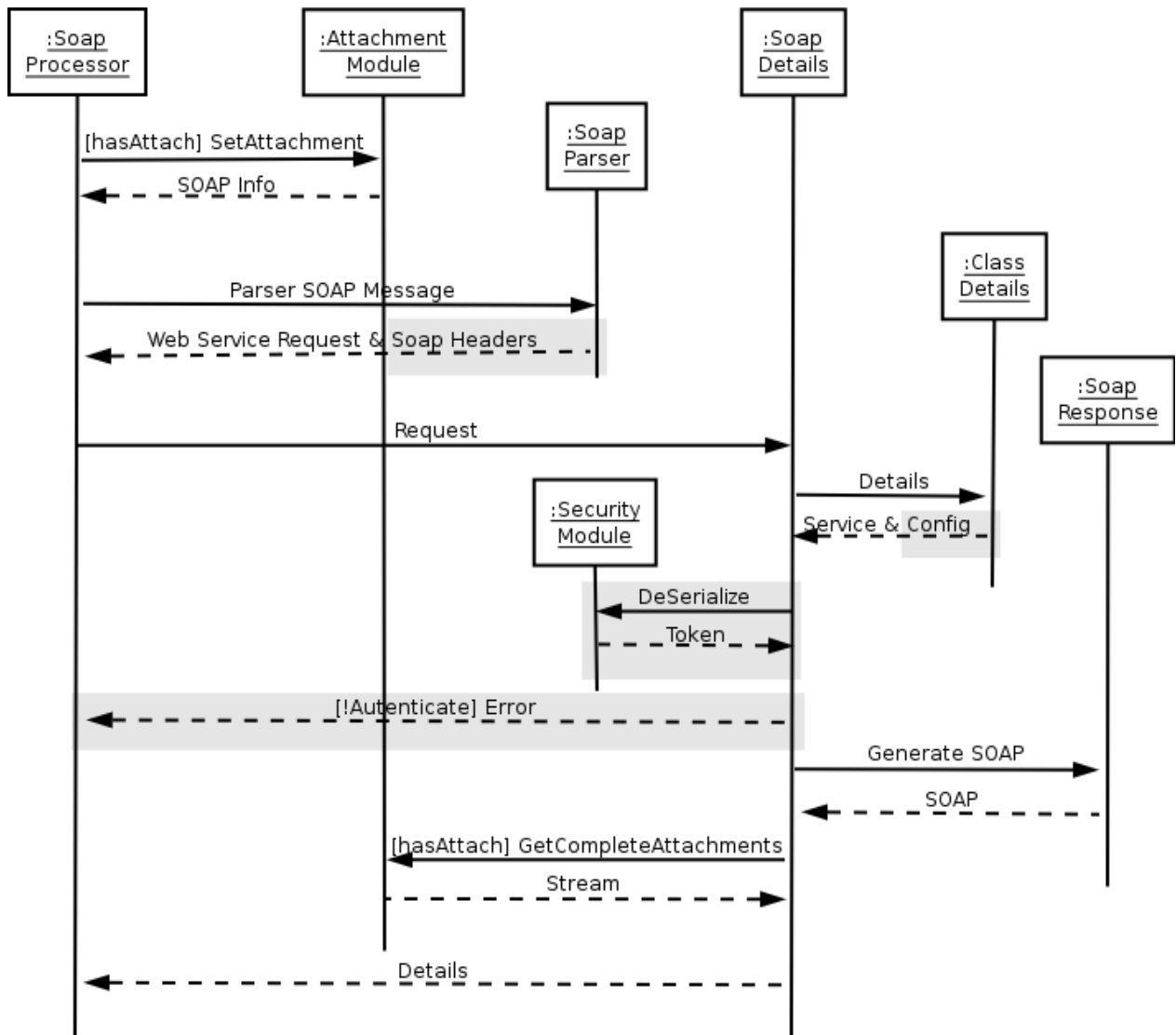


Figura 4.8. Diagrama de Secuencia del módulo SOAP

De esta forma, el MicroServidor queda listo para autenticar usuarios, permitiendo la invocación de servicios solamente para usuarios autorizados. Sin embargo, también se hizo necesario construir un cliente para que dispositivos móviles pequeños puedan comunicarse con el MicroServidor, siempre adecuándose a los estándares existentes.

4.4. Seguridad en un Cliente

En 3.4.3 “Adjuntos en un cliente”, se describió un protocolo de comunicación llamado *MobileWebSeverClientProtocol*, que deriva de *SoapHttpClientProtocol*, el cual está encargado de generar mensajes SOAP y establecer comunicación con un servidor. Este protocolo SOAP permite también agregar tantos encabezados SOAP como se quieran, definiendo clases que

deriven de *SoapHeader*. Esta última es una clase abstracta que provee el módulo Web Services del .NET Compact Framework.

De esta forma se creó la clase *SecurityHeader*, que contiene una lista de *tokens* de seguridad. Por otra parte, a la clase *MobileWebSeverClientProtocol* se le agregó el método *AddToken* para agregar un *token* a la lista del *SecurityHeader* (ver Figura 4.9). Luego, cuando se invoque un método Web, *SoapHttpClientProtocol* busca los objetos públicos que deriven de *SoapHeader* y los serializa. Como *SecurityHeader* implementa *IXmlSerializable*, de forma personalizada se obtiene (para el caso de *UsernameToken*) un *Nonce* y se calcula el hash de la contraseña, si es necesario.

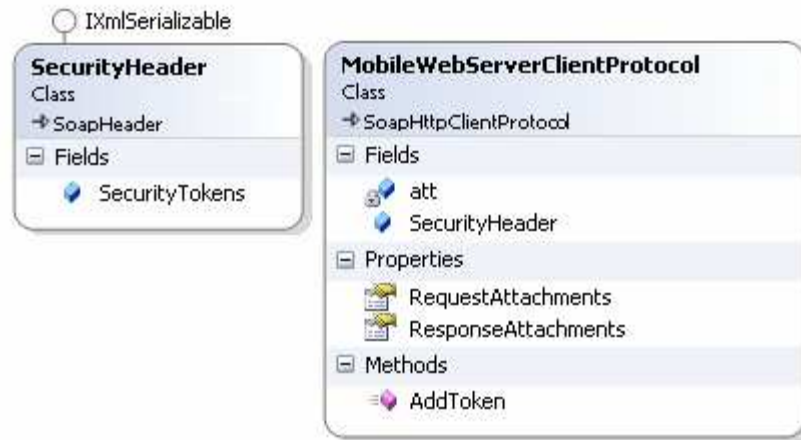


Figura 4.9. Diagramas de las clases *SecurityHeader* y *MobileWebServerClientProtocol*

Para ilustrar la facilidad de utilizar seguridad en el cliente, la Figura 4.10 muestra un extracto de código, en el que a un *proxy* (instancia de una clase que deriva de *MobileWebSeverClientProtocol* y que contiene los métodos para invocar a un servicio Web) se le asigna un *token* de seguridad.

```
1 MWSPProxy proxy = new MWSPProxy();
2 //Crear un token
3 UsernameToken uToken = new UsernameToken("user",
4                                           "secret",
5                                           PasswordOption.Hashed);
6 //Agregar al proxy
7 proxy.AddToken(uToken);
8
9 //Invocar metodo Web
10 proxy.TestMethod();
11
12 //Continuar...
```

Figura 4.10. Extracto de código de un cliente

Entre las líneas 3 y 5 está la creación del *UsernameToken* del módulo de Seguridad descrito en 4.2 “Módulo Seguridad”, demostrando así que el módulo de seguridad trabaja independiente de lo que es el MicroServidor Web.

4.5. Resumen

Al término de la implementación de la seguridad en el MicroServidor, se pueden distinguir al menos tres logros importantes, que dan por satisfecho el trabajo realizado.

1. *Interoperabilidad*: Cumple con el estándar WS-Security. No fue necesario agregar más información, lo que permite que cualquier cliente que cumpla con el estándar pueda autenticarse en el MicroServidor.
2. *Flexibilidad*: Se logró una buena flexibilización de la configuración de seguridad. Esto fue porque un servicio Web puede tener o no seguridad, simplemente cambiando valores en su archivo de configuración. Además, indicar en dicho archivo la herramienta de configuración hace que cualquier repositorio de datos de contraseñas de usuarios sirva, pues lo importante es que la herramienta implemente la interfaz *ISecurityUtils*. Por último, en el archivo de configuración se puede elegir qué *token* (o *tokens*) acepta el servicio Web.
3. *Soporte para Desarrollos Futuros*: Lo más próximo a hacer sobre esta plataforma es implementar *BinarySecurityToken*. Tal como se mencionó en el inicio de este capítulo, la implementación que se haría de seguridad dejaría la puerta abierta para que se implementaran nuevas capas de seguridad. Según lo realizado, en el futuro, a la hora de implementar *BinarySecurityToken*, se haría lo siguiente:
 - a. Crear una clase *BinarySecurityToken* que implemente la interfaz *ISecurityToken*.
 - b. Agregar a la interfaz *ISecurityUtils* métodos y/o propiedades necesarias para proveer datos a la autenticación del nuevo *token*.
 - c. Crear un módulo que implemente la interfaz modificada *ISecurityUtils*¹, para agregarla en el atributo “Utility” de la etiqueta “security” del archivo de configuración del servicio Web.
 - d. Implementar la autenticación correspondiente.

De esta forma, el MicroServidor ya tiene una base para empezar a agregar más capas de seguridad a través del tiempo, basándose en estándares. De esa manera, poco a poco se puede alcanzar un mayor nivel de interoperabilidad, que es uno de los objetivos principales de los servicios Web.

¹ Para evitar redefinir todas las clases que ya implementan *ISecurityUtils* es conveniente crear una clase abstracta que implemente la interfaz y que la herramienta de seguridad derive de aquella clase abstracta, redefiniendo sólo los métodos que se estimen conveniente.

Capítulo 5. Resultados, Conclusiones y Trabajo a Futuro

Durante el desarrollo de este trabajo de título se hicieron constantes pruebas que abarcaban varios aspectos, como la velocidad de respuesta del MicroServidor, integridad de los datos adjuntos y autenticación correcta. Además, en dichas pruebas se utilizaron diferentes tecnologías, basadas en .NET Framework, para crear clientes que interactuaran con el MicroServidor. Las tecnologías utilizadas fueron las siguientes:

- *System.Web.Services*, definiciones propias de .NET Framework (.NET) para crear clientes sencillos; es decir, que no necesitan capacidades de adjuntos ni seguridad. También existen definiciones equivalentes para el .NET Compact Framework (.NET CF).
- *System.Web.Services2*, definiciones de Web Services Enhancement 2.0 (WSE) que agregan a *System.Web.Services* de .NET capacidades para manejar datos adjuntos y seguridad.
- *OpenNETCF.Web.Services2*, definiciones de OpenNETCF que agregan a *System.Web.Services* de .NET CF las capacidades necesarias para manejar datos adjuntos y seguridad en la versión reducida del framework.

A pesar de que durante el trabajo de título la construcción de clientes se implementó a través de una definición propia, basada en .NET CF, es posible crear clientes utilizando otras tecnologías. Esto sería particularmente útil para demostrar que el diseño del MicroServidor está basado en estándares y que, en realidad, cualquier cliente es aceptado mientras se comporte de acuerdo a lo que estos estipulan¹.

Los medios de prueba fueron PDAs (Personal Digital Assistants) y un emulador de SmartPhone. Sin embargo, no se tuvo la capacidad de obtener el protocolo de Internet (IP) del dispositivo emulado, lo que impidió la realización de pruebas. Solamente se pudo probar resultados invocando el MicroServidor desde un navegador Web del dispositivo emulado (protocolo GET). A pesar de obtener respuestas correctas, el protocolo era insuficiente para probar los trabajos de adjuntos y seguridad.

En este capítulo se muestran diferentes pruebas realizadas sobre el MicroServidor, usando las tecnologías mencionadas anteriormente². Además, se muestran aplicaciones que sacan provecho de las facilidades de tener un MicroServidor Web en un dispositivo móvil. Con estas pruebas, y con el trabajo descrito en este documento se exponen luego las conclusiones y el trabajo a futuro.

5.1. Pruebas de Desempeño

Las pruebas de desempeño, en el contexto del MicroServidor, corresponden a peticiones que se hacen de forma consecutiva; es decir, se envía una petición e inmediatamente después de recibir una respuesta y asegurar su exactitud, se envía otra. Alrededor de 50 peticiones son enviadas a

¹ Por otra parte, implementar una definición propia de clientes sirve para no depender de OpenNETCF para realizar peticiones al MicroServidor.

² Todas las pruebas se realizaron en el laboratorio de Microsoft del Departamento de Ciencias de la Computación (DCC) de la Universidad de Chile, usando Wi-Fi que el mismo Departamento provee.

diferentes servicios Web que el MicroServidor ofrece. Es de notar que las peticiones esperan una amplia variedad de resultados: cadena de caracteres, valores de verdad, objetos complejos, arreglos y documentos XML. Los protocolos de comunicación usados son POST, GET y SOAP.

Originalmente estas pruebas de desempeño se hacían desde un computador de escritorio. Para hacerlas desde un dispositivo móvil, durante el trabajo de título se diseñó un programa ad-hoc. Las pruebas de desempeño en ambas plataformas, fueron ejecutadas con éxito, cada una alcanzando un promedio de 12 segundos en realizar las 50 peticiones. Lo anterior no incluye las nuevas características de seguridad y adjuntos en el MicroServidor, solamente es útil para verificar que los cambios que se realizan no alteran los actuales beneficios de éste.

Para probar las nuevas características, se tiene un servicio Web que envía al cliente todos los archivos que tenga el servidor a disposición, previa autenticación del usuario. El cliente, por su parte, espera los archivos adjuntos, solamente cuando ha enviado un *token* de seguridad correcto.

Más allá de decir que el MicroServidor autentica de forma correcta al cliente, la Figura 5.1 muestra el tiempo que demora una prueba, la cual consiste en enviar 6 peticiones, en las que 2 de ellas tienen autenticación correcta. La variable es la cantidad de kilobytes de todos los adjuntos.

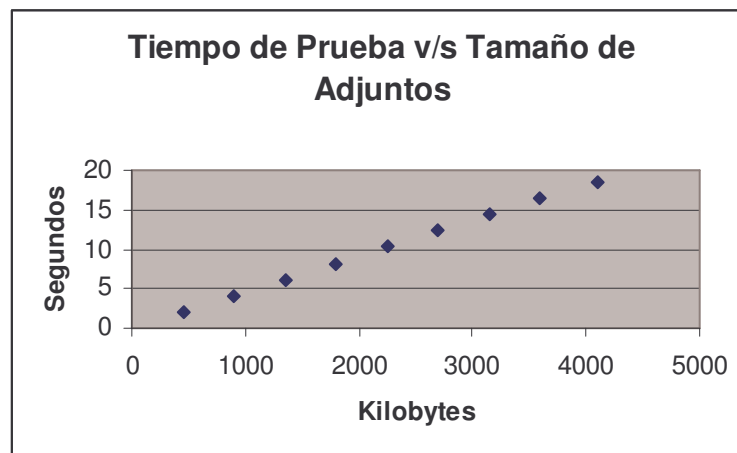


Figura 5.1. Gráfico Tiempo de prueba v/s Tamaño de adjuntos

Se nota, gracias al gráfico de arriba, que existe linealidad entre el aumento de adjuntos y el tiempo de respuesta total de cada prueba, lo cual implica que el módulo de adjuntos trabaja un tiempo proporcional al tamaño de adjuntos que está procesando. Esto es una buena señal con respecto a los recursos que el MicroServidor está utilizando.

5.2. Pruebas Concurrentes

Para probar concurrencia en el MicroServidor, se realizaron las mismas pruebas descritas en el punto anterior, pero a través de ejecuciones concurrentes. El MicroServidor respondió satisfactoriamente, aunque el tiempo de respuesta aumentó debido a que se procesaron más de una solicitud a la vez. El gráfico de la Figura 5.2 muestra lo anteriormente dicho. En este caso la variable graficada es la cantidad de clientes haciendo solicitudes, sobre una cantidad de adjuntos de 1 Megabyte.

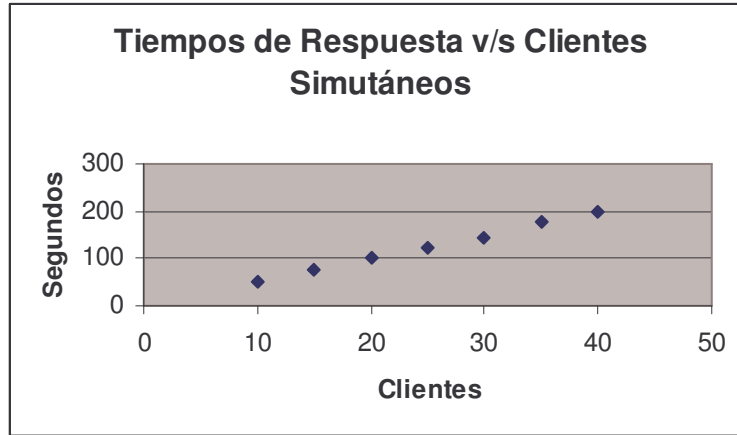


Figura 5.2. Gráfico de Tiempos de Respuestas v/s Clientes Simultáneos

No es de extrañar que el gráfico muestre tiempos de espera superior a dos minutos, pues no se debe olvidar que cada tiempo corresponde a una prueba y cada prueba tiene dos peticiones que reciben adjuntos de 1 Megabyte. El que el MicroServidor responda a 40 clientes de forma simultánea, enviando 1 Megabyte de información a cada uno de ellos, no es para nada despreciable, pues se está trabajando en un dispositivo móvil pequeño, lo que implica capacidades limitadas de recursos de hardware.

5.3. Adjuntos v/s Parámetros

Durante el trabajo de título, se agregó una característica nueva al MicroServidor para que éste aceptara peticiones de clientes que invocaban métodos de servicios Web, cuyos argumentos y retornos eran documentos XML (*XmlDocument*). Esta nueva característica es ahora de mucha utilidad para, por ejemplo, determinar qué tan bueno es enviar documentos XML como adjuntos en vez de enviarlos como parámetros.

Para esto, se implementó un servicio Web que recibe un documento XML, lo modifica y lo envía de vuelta al cliente. El servicio Web contiene dos métodos: uno que recibe el documento como parámetro y lo envía como un objeto de retorno; y el segundo que recibe y envía el documento como adjunto. La Figura 5.3 muestra los tiempos de respuesta de ambos métodos, según el tamaño del documento XML.

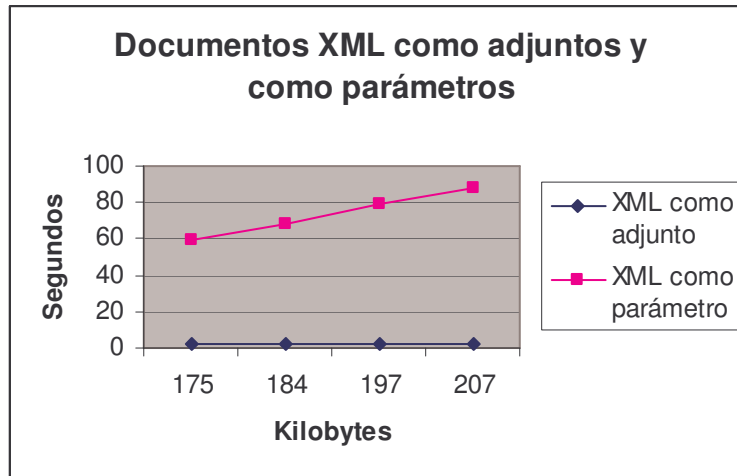


Figura 5.3. Gráfico de Documentos XML como adjuntos y como parámetros

Claramente, el uso de adjuntos es inmensamente superior, por lo que esta nueva característica en el MicroServidor aumenta la eficiencia en las respuestas y justifica la necesidad de crear el módulo de adjuntos en el trabajo de título.

De forma más general, en vez de utilizar archivos XML en las pruebas, puede probarse el envío de secuencias de bytes. Para esto, se implementó un servicio Web que recibe una secuencia de bytes y la guarda a disco. El servicio Web realiza esto recibiendo la secuencia como parámetro o como adjunto. La Figura 5.4 muestra los resultados obtenidos.

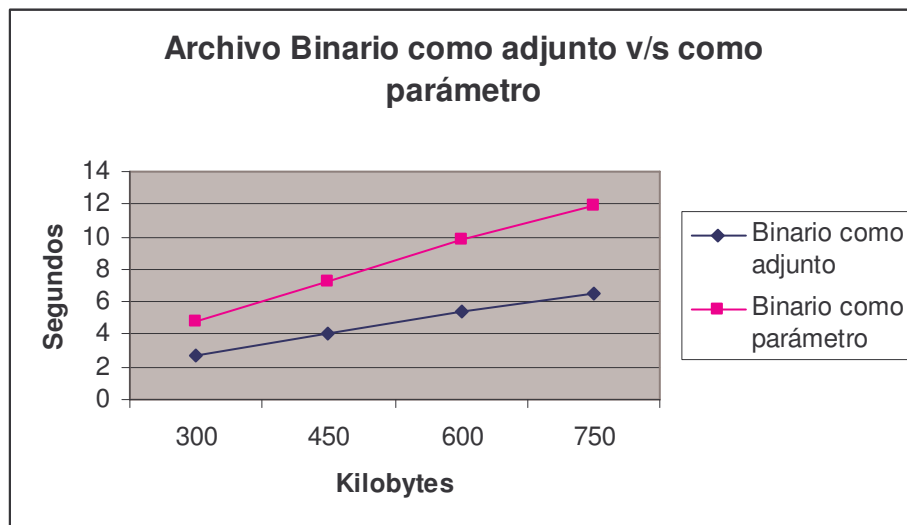


Figura 5.4. Gráfico de Archivo Binario como adjunto y como parámetro

A pesar que los tiempos de respuesta no son tan superiores como los obtenidos usando archivos XML, igualmente se nota que el uso de adjuntos provee mejores tiempos de respuesta, corroborando la utilidad del nuevo módulo de adjuntos.

5.4. Mapa Móvil

Mapa Móvil [22] es una aplicación que permite, a grandes rasgos, visualizar mapas en una PDA e insertar puntos de interés sobre los mapas. Es una aplicación muy útil bajo el contexto de zonas de catástrofes.

El MicroServidor se convirtió en una herramienta que le agregó valor a la aplicación, puesto que hizo posible el compartir, mediante servicios Web, puntos de interés entre diferentes PDAs. Por dar un ejemplo, en un escenario de catástrofes, un bombero puede insertar los puntos de interés de la zona afectada. Luego, cuando una nueva compañía de bomberos llegue a la fuerza de tareas, estos puntos son puestos a disposición de ellos, para que todos puedan tenerlos en sus propios dispositivos y apoyar sus decisiones con esta información.

Mapa Móvil se convierte así en la primera herramienta que obtiene funcionalidades de valor gracias al MicroServidor, señalando que el impacto que puede tener el MicroServidor se traducirá en mejores herramientas de colaboración y, por lo tanto, aplicaciones de mayor valor.

5.5. Mobile File Sharing

Mobile File Sharing es una aplicación que sirvió, durante el trabajo de título, para comprobar lo correcto y eficiente que estaban resultando los diseños de adjuntos y seguridad. La aplicación consiste en que un usuario puede visualizar archivos de los directorios de una PDA remota y también puede descargarlos, pero esto último puede realizarse si hay una autenticación previa (ver Figura 5.5).

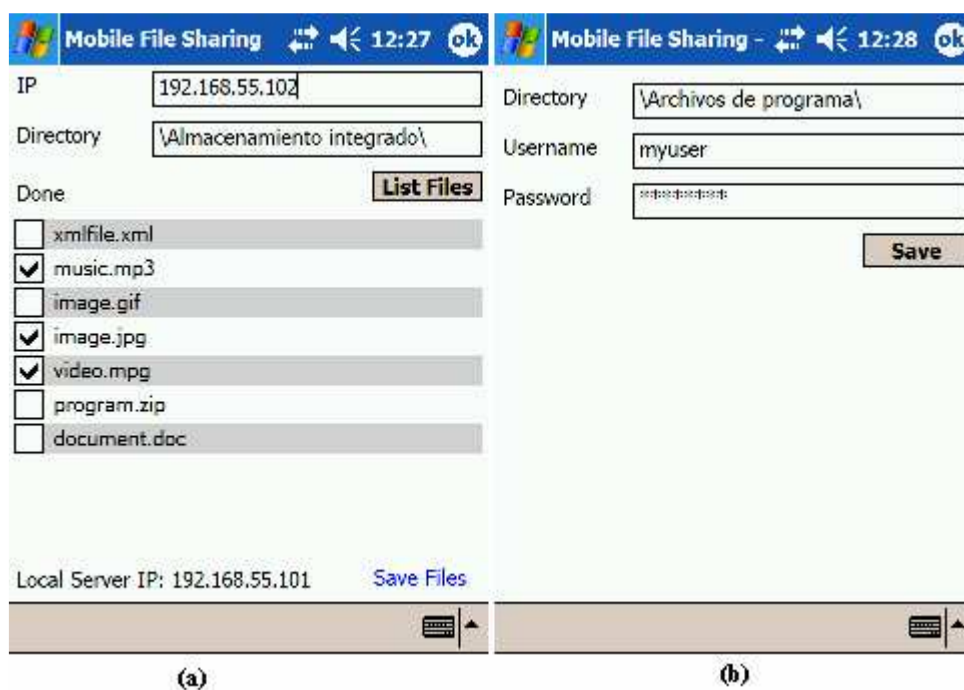


Figura 5.5. Programa Mobile File Sharing (a) Obtención y selección de los archivos de la carpeta remota Almacenamiento integrado. (b) Descarga de archivos seleccionados previa autenticación

Con esta aplicación se logra saber los tiempos de respuesta del MicroServidor, autenticación correcta y exactitud de datos adjuntos, todos estos aspectos relevantes dentro del trabajo realizado. Por otra parte, se entiende que el MicroServidor está preparado para ser utilizado por aplicaciones colaborativas que necesiten servicios Web, adjuntos y autenticación, herramientas suficientes para agregar valor a muchas de estas aplicaciones.

5.6. Conclusiones

Trabajar con servicios Web aumenta la posibilidad de integrar sistemas independientes. Sin duda, las mejoras desarrolladas sobre el MicroServidor, sobre adjuntos y seguridad, capacitan a esta plataforma para abarcar muchos más escenarios en los cuales integrar sistemas.

Esta integración es factible gracias a la existencia de estándares, especificaciones que en este trabajo de título tomaron un papel importante dentro del diseño de los módulos de adjuntos y seguridad. El hecho de que el MicroServidor aceptara peticiones de clientes construidos con distintos tipos de tecnologías, demuestra que este trabajo siguió de forma correcta los estándares propuestos, además de proveer una experiencia que motiva a utilizar estándares siempre que sea posible.

Una ventaja importante del diseño del trabajo realizado, es que ahora se tiene la habilidad para extender el MicroServidor según nuevas especificaciones que surjan. En cada capítulo se expuso un resumen de los pasos a seguir para cuando se quieran agregar más características a los módulos creados. Durante el trabajo de título siempre se tuvo presente que el MicroServidor Web es una herramienta continuamente extensible y, por lo tanto, era vital que los diseños hechos sirvieran de base para las constantes mejoras que éste tendrá.

Uno de los resultados que deja muy satisfecho es cómo el MicroServidor es utilizado por la aplicación Mapa Móvil. Dicha aplicación corresponde a un trabajo totalmente independiente al MicroServidor; donde al momento de integrar ambas herramientas, los resultados que se obtuvieron fueron exitosos. Esto es una señal que demuestra que el MicroServidor es una utilidad que puede ser ocupada por una amplia variedad de aplicaciones que necesiten colaboración e intercambio de información.

Estas ventajas significan, para los llamados “usuarios móviles”, mejores alternativas de uso para sus dispositivos móviles, permitiendo mejorar sus capacidades de trabajo y ofreciéndoles mejores servicios.

5.7. Trabajo a Futuro

Las extensiones que se pueden realizar sobre el MicroServidor, y en realidad sobre un servidor Web en general, son muchas. Por ejemplo, si de estándares se trata, WS-Attachments y WS-Security son apenas dos de al menos 18 que existen para los servicios Web. Sin embargo, con respecto a las dos especificaciones tratadas en este trabajo de título, aún quedan por diseñar algunos detalles que enriquecen las capacidades del MicroServidor. Entre ellas se puede mencionar a las siguientes:

- Implementar otros tipos de protocolos de adjuntos, para no depender el protocolo DIME.
- Agregar otros *tokens* de seguridad, así los usuarios tendrán mejor disposición para usar servicios Web que requieran seguridad.
- Mantener en memoria solamente los módulos que el MicroServidor ocupa con frecuencia, para no exigir al dispositivo móvil el uso de recursos innecesarios.

El MicroServidor tiene un constructor WSDL para los servicios Web, sin embargo, éste tiene que ser actualizado para que describa de forma correcta los servicios Web que aceptan adjuntos y/o requieran seguridad. Francisco Castro [5] desarrolló un módulo administrador de sesiones, que entre otras cosas, permite a un usuario ser parte de una sesión y compartir recursos de su dispositivo móvil, con otros usuarios de su misma sesión. Si consideramos los servicios Web como recursos, es útil que el MicroServidor tenga una herramienta que le provea información acerca de las sesiones existentes, para así determinar si un cliente debe o no tener acceso al servicio Web que quiera invocar. A pesar de que existe una definición de acceso a sesiones en el MicroServidor, en el corto plazo, ésta debe alinearse con la información que el administrador de sesiones provea.

Como se mencionó al inicio de esta sección, las mejoras que se puedan hacer a un servidor Web son muchas. A medida que el MicroServidor evolucione, siempre se debe tener en cuenta los recursos que se ocupan y que los diseños que se hagan sean lo más modulares posibles, para así facilitar la incorporación de las inminentes modificaciones futuras.

Bibliografía y Referencias

1. Beatty, J., Kakivaya, G., Kemp, D., Lovering, B., Roe, B., Schlimmer, J., Simonnet, G., Weast, J. Web Services Dynamic Discovery (WS-Discovery). Microsoft Press. 2004.
2. Brugnoli, M.C., Davide, F., Slagter, R. The future of mobility and of mobile services. Report of MOSAIC Project, <http://mosaic-network.org>. 2006.
3. Buszko, D., Lee, W., Helal, A. Decentralized Ad-Hoc Groupware API and Framework for Mobile Collaboration. Proc. of International Conference on Supporting Group Work (GROUP), ACM Press, Colorado, USA, 2001.
4. Butek, R., Nash, S., Nielsen, H., Sanders, H. Direct Internet Message Encapsulation (DIME). Internet-Draft. <http://xml.coverpages.org/draft-nielsen-dime-02.txt>. Última visita: Octubre, 2006.
5. Castro, Francisco. Manejo de Sesiones Cerradas para Ambientes Colaborativos Móviles. Universidad de Chile. 2007.
6. Duffy, J. Professional .NET Framework 2.0 (Programmer to Programmer). Wrox Press. 2006.
7. Dustdar, S., Gall, H., Mehandjiev, N., Tata, S. DMC – Distributed and Mobile Collaboration Workshop Report. Proc. of the 14th IEEE Internacional Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE'05). 2005.
8. Foggon, D., Maharry, D., Ullman, C., Watson, K. Programming Microsoft .NET XML Web Services. Microsoft Press, 2004.
9. Gehlen, G., Pham, L. Mobile Web Services for Peer-to-Peer Applications. Proc. of the 2nd IEEE Consumer Communications and Networking Conference, (CCNC), Las Vegas, USA. pp. 427-433. 2005.
10. GotDotNet. Global XML Web Services Architecture. http://www.gotdotnet.com/team/XMLwebservice/gxa_overview.aspx. Última visita: Junio, 2006.
11. Guerrero, L.A., Pino, J.A., Collazos, C.A., Inostroza, A., Ochoa, S.F. Mobile Support for Collaborative Work. Group Decisions and Negotiations Journal. In press.
12. Jorstad, I., Dustdar, S., Thanh, D.V. A Service Oriented Architecture Framework for Collaborative Services. Proc. of the 14th IEEE Internacional Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE'05). 2005.
13. Meissner, A., Babu Musunoori, S. Group Integrity Management Support for Mobile Ad-Hoc Communities. Proc. of the 1st Workshop on Middleware for Pervasive and Ad Hoc Computing. 2003.

14. Microsoft Corp. Web Services and Other Distributed Technologies. <http://msdn.microsoft.com/webservices/webservices/>. Última visita: Octubre, 2006.
15. Neyem, A., Ochoa, S., Guerrero, L., Pino, J. Sharing Information Resources in Mobile Ad-hoc Networks. Proc. of International Workshop on Groupware (CRIWG), LNCS 3706, Springer-Verlag. Pp. 351-358. Sept., 2005.
16. Organization for the Advancement of Structured Information Standards (OASIS). URL: <http://www.oasis-open.org/home/index.php>. Última visita: Octubre del 2006.
17. Pashtan, A. Mobile Web Services. Cambridge Univertisty Press, 2005.
18. Pratistha, Putera. Exposing Web Services on Mobile Devices. Monash University. 2002.
19. Schaffers, H., Brodt, T., Pallot, M., Prinz, W. (eds). The Future Workplace - Perspectives on Mobile and Collaborative Working. Mosaic Consortium. Telematica Instituut, The Netherlands. 2006.
20. Sen, R., Handorean, R., Roman, G.C., Gill, C. Service Oriented Computing Imperatives in Ad Hoc Wireless Settings (Book Chapter). Service-Oriented Software System Engineering: Challenges And Practices. 2004.
21. Sycara, K., Paolucci, M., Ankolekar, A., Srinivasan, A. Automated discovery, interaction and composition of Semantic Web services. Web Semantics: Science, Services and Agents on the World Web, 1(1), pp. 27-46. 2003.
22. Villarroel, Manuel. Navegación y actualización de información de Cartografía, utilizando dispositivos handheld. Universidad de Chile. 2007.
23. Wagner, M., Balke, W., Hirschfeld, R., Kellerer, W. A Roadmap to Advanced Personalization of Mobile Services. Proc. of the International Federated Conferences DOA/ODBASE/CoopIS (Industry Program). 2002.
24. Web Services Security. UsernameTokenProfile 1.1. Oasis Standard Specification, 1 February, 2006. <http://www.oasis-open.org/committees/download.php/16782/wss-v1.1-spec-os-UsernameTokenProfile.pdf>. Última visita: Noviembre, 2006.

Apéndice A: Código Fuente

A continuación se presenta parte del código fuente de los principales componentes implementados en esta memoria.

A.1. Código Fuente WS-Attachment

Interfaz IAttachment

```
/// <summary>
/// This interface must implement the attachment modules.
/// The method listed are used from the Mobile Web Server
/// </summary>
public interface IAttachment
{
    /// <summary>
    /// Gets the Mime Type of the attachment module.
    /// It is needed to set the Content-Type web header.
    /// </summary>
    string MimeType
    {
        get;
    }

    /// <summary>
    /// Transforms a set of bytes (maybe from a remote connection)
    /// to a list of attachments.
    /// </summary>
    /// <param name="attachments">Bytes to parse to fount attachments</param>
    /// <param name="attachmentsList">List in wich store each
    /// attachment</param>
    /// <returns>A soap envelope if it is in the bytes to parse, else returns
    /// a 0 length array </returns>
    byte[] SetAttachments(byte[] attachments,
        out IList attachmentsList);

    /// <summary>
    /// Process a soap envelope and attachment to get a complete response
    /// or request to send
    /// </summary>
    /// <param name="soap">The soap envelope of the request or
    /// response</param>
    /// <param name="attachments">List to get the attachments to make a
    /// complete request or response</param>
    /// <returns>A stream which stores the complete data ready to
    /// send</returns>
    System.IO.Stream GetCompleteAttachment(byte[] soap,
        IList attachments);

    /// <summary>
    /// Gets or sets a list of attachments of a request
    /// </summary>
    IList RequestAttachments
    {

```

```

        get;
        set;
    }

    /// <summary>
    /// Gets or sets a list of attachments of a response
    /// </summary>
    IList ResponseAttachments
    {
        get;
        set;
    }
}

```

DimeAttribute

```

/// <summary>
/// Attribute needed in a web method to manage request and response
attachments
/// </summary>
[AttributeUsage(AttributeTargets.Method, AllowMultiple = false, Inherited =
false)]
public class DimeAttribute : Attribute, IAttachment
{
    #region Static Collections

    //The following two collections are static because the server must set the
//request attachments by reflection to pass to a web method.
//In the other hand, the web method must set the response attachments and
//the server, with reflection, get those attachments, thanks the static
property
//of the collections

    /// <summary>
    /// Collection to store response attachments
    /// </summary>
    static DimeAttachmentCollection _ResponseAttachments = new
DimeAttachmentCollection();

    /// <summary>
    /// Collection to store request attachments
    /// </summary>
    static DimeAttachmentCollection _RequestAttachments = new
DimeAttachmentCollection();

    #endregion Static members

    /// <summary>
    /// Gets the soap envelope from a collection and removes it in the
collection
    /// </summary>
    /// <param name="records">Collections where find the soap envelope</param>
    /// <returns>If soap envelope exists in collection returns this soap
envelope
    /// as an array of bytes, else returns a 0 length array</returns>
    private byte[] ExtractSoap(ref DimeRecordCollection records)
    {
        // default return

```

```

        byte[] soap = new byte[0];

        // Dime specification says that the soap envelope must be the first
record
        if (records.Count > 0 &&
            records.Item(0).DimeType ==
"http://schemas.xmlsoap.org/soap/envelope/")
        {
            soap = records.Item(0).DimeData;
            // then remove the soap
            records.RemoveAt(0);
            // It fails if soap envelope is part of a chunk record
        }
        return soap;
    }

    public IList RequestAttachments[...]

    public IList ResponseAttachments[...]

    public string MimeType
    {
        get { return "application/dime"; }
    }

    public Stream GetCompleteAttachment(byte[] soapResult,
                                        IList attachments)
    {
        DimeGenerator dimeGenerator = new DimeGenerator();

        // add soap file as first attachment
        DimeRecord record = new DimeRecord();
        record.DataType = DimeRecord.DataFormatType.Absolute_URI;
        record.DimeType = "http://schemas.xmlsoap.org/soap/envelope/";
        record.DimeData = soapResult;

        dimeGenerator.AddDimeRecord(record);

        // then add the attachments
        foreach (DimeAttachment dimeAttachment in attachments)
        {
            record = dimeAttachment;
            record.DataType = DimeRecord.DataFormatType.Media_Type;
            record.DimeData = dimeAttachment.AttachmentBinary;

            while (dimeAttachment.IsChunked)
            {
                record.Chunked = true;
                dimeGenerator.AddDimeRecord(record);
                record = new DimeRecord();
                record.DimeData = dimeAttachment.AttachmentBinary;
                // Dime specifications says that in chunks record (except
first)

                // DataType must be Unchange and ID_Length 0
                record.DataType = DimeRecord.DataFormatType.Unchanged;
                record.DimeID = string.Empty;
            }
            record.Chunked = false;
            dimeGenerator.AddDimeRecord(record);
        }
    }

```

```

    }
    return dimeGenerator.GetStream();
}

public byte[] SetAttachments(byte[] attachments,
                            out IList attachmentsList)
{
    DimeParser parser = new DimeParser(new MemoryStream(attachments));

    DimeRecordCollection collection = parser.Records;

    // Extract the soap and store to return it later
    byte[] soap = ExtractSoap(ref collection);

    attachmentsList = new DimeAttachmentCollection();

    // List filled with bytes of chunks records
    List<byte> chunkByteList = new List<byte>();
    string chunkType = string.Empty, chunkId = string.Empty;

    foreach (DimeRecord dimeRecord in collection)
    {
        if (dimeRecord.Chunked)
        {
            chunkByteList.AddRange(dimeRecord.DimeData);
            // The next record(s) has DataType Unchange and a 0 length Id
            // so we store the current data
            if(dimeRecord.DataType != DimeRecord.DataFormatType.Unchanged)
            {
                chunkId = dimeRecord.DimeID;
                chunkType = dimeRecord.DimeType;
            }
        }
        else if (chunkByteList.Count > 0)
        {
            // the record isn't chunk anymore so the attachment is
            complete
            chunkByteList.AddRange(dimeRecord.DimeData);
            attachmentsList.Add(new
DimeAttachment(chunkByteList.ToArray(),
                                                         chunkType,
                                                         chunkId));

            chunkByteList.Clear();
            chunkId = string.Empty;
            chunkType = string.Empty;
        }
        else
        {
            // the record has the complete attachment
            attachmentsList.Add(new DimeAttachment(dimeRecord.DimeData,
                                                         dimeRecord.DimeType,
                                                         dimeRecord.DimeID));
        }
    }
    // returns the soap extracted from the original collection
    return soap;
}
}

```

A.2. Código Fuente WS-Security

Interfaz ISecurityToken

```
/// <summary>
/// All tokens must implement this interface.
/// The main aspect is the IXmlSerializable interface
/// </summary>
public interface ISecurityToken : System.Xml.Serialization.IXmlSerializable
{
    //This properties are about the timestamp of any token

    string TimestampId[...]

    string SecurityCreated[...]

    string SecurityExpires[...]
}
```

UsernameToken

```
public class UsernameToken : ISecurityToken
{
    [Fields & Properties]

    public UsernameToken() { }
    public UsernameToken(string username,
                        string password,
                        PasswordOption option) [...]

    #region IXmlSerializable

    /// <summary>
    /// Read the data of an xml and asign values to the properties
    /// of the token
    /// </summary>
    /// <param name="xmlReader">Reader with soap fragment</param>
    public void ReadXml(XmlReader xmlReader)
    {
        while (xmlReader.Read())
        {
            if (!xmlReader.IsStartElement())
            {
                continue;
            }

            switch (xmlReader.LocalName.ToLower())
            {
                case "security":
                    _UsernameTokenId = xmlReader.GetAttribute("wsu:Id");
                    break;
                case "username":
                    _Username = xmlReader.ReadString();
                    break;
                case "password":
```

```

        _PasswordType = _SetPasswordOption(
            xmlReader.GetAttribute("Type"));
        _Password = xmlReader.ReadString();
        break;
    case "nonce":
        _Nonce = xmlReader.ReadString();
        break;
    case "created":
        _Created = xmlReader.ReadString();
        break;
    default:
        break;
    }
}

public System.Xml.Schema.XmlSchema GetSchema() [...]

public void WriteXml(XmlWriter writer) [...]

#endregion

private PasswordOption _SetPasswordOption(string passwordType) [...]

public bool Authenticate(string realPassword) [...]

/// <summary>
/// Calculate the hash value of the password using also
/// the nonce and created values
/// </summary>
/// <param name="password">A plain text password</param>
/// <returns>A code base 64 string with hash value</returns>
public string GetDigest(string password)
{
    System.Security.Cryptography.SHA1 sha =
        System.Security.Cryptography.SHA1.Create();

    byte[] nonceBinary = Convert.FromBase64String(this._Nonce);
    byte[] createdBinary = Encoding.UTF8.GetBytes(this._Created);
    byte[] serverPasswordBinary = Encoding.UTF8.GetBytes(password);

    byte[] total = new byte[nonceBinary.Length +
        createdBinary.Length +
        serverPasswordBinary.Length];

    Array.Copy(nonceBinary, total, nonceBinary.Length);
    Array.Copy(createdBinary, 0,
        total, nonceBinary.Length, createdBinary.Length);
    Array.Copy(serverPasswordBinary, 0,
        total, nonceBinary.Length + createdBinary.Length,
        serverPasswordBinary.Length);

    byte[] serverDigest = sha.ComputeHash(total);

    return Convert.ToBase64String(serverDigest);
}
}

```

A.3. Mejoras Realizadas en el MicroServidor

A continuación se lista una serie de modificaciones realizadas al MicroServidor, algunas orientadas a mejorar el uso de recursos en dispositivos móviles pequeños. La gran mayoría no tiene relación directa con los objetivos de este trabajo de título, pero fueron vitales para el correcto funcionamiento del MicroServidor. Considérese que algunas mejoras tuvieron errores asociados, los que provocaron retardos en la agilidad de diseño e implementación de los objetivos del trabajo de título.

- Arreglar de la lectura del SOAP del cliente (*SoapParser*), pues no procesaba bien archivos SOAP que contenían etiqueta vacías (<etiqueta/>).
- Modificar los encabezados Web de respuesta, pues habían clientes que no aceptaban una respuesta que no tuviera el encabezado *Content-Length* aunque su valor fuese cero.
- Leer correctamente datos desde clientes que usaban protocolo de versión anterior a HTTP/1.1, pues el MicroServidor asumía que los clientes eran de protocolo HTTP/1.1, el cual tiene diferentes características en lo que a transferencia de datos se refiere.
- Aceptar invocaciones de métodos Web que tuviesen parámetros complejos, entre los que se encuentran *XmlDocument* y *DataSet*. Inicialmente, el MicroServidor procesaba métodos Web de parámetros primitivos (string, int).
- Aceptar invocaciones de métodos Web que tuviesen retornos *bool* y *void*.
- Mejorar la forma de leer los datos desde un cliente. Inicialmente se leía una sola vez y se procesaba la petición. Ahora se lee cuantas veces sea necesario, según el *Content-Length* que el cliente envía como encabezado de su petición.
- Mejorar el excesivo uso de objetos, cambiado listas de objetos, por listas genéricas y especializadas.
- Arreglar el método de registro de eventos, pues ante concurrencia provocaba errores (dos procesos escribiendo el mismo archivo).
- Agregar definiciones del ensamblado System.Web de .Net para codificar y decodificar caracteres especiales de los archivos XML.
- Quitar variables globales del módulo Soap, las cuales, erróneamente, se compartían en diferentes peticiones, provocando errores cuando había concurrencia.
- Serializar objetos dependiendo de la plataforma de trabajo. Por razones desconocidas, la serialización XML de algunos objetos en .NET es diferente entre las plataformas de Windows y Windows Mobile.