Capítulo 2. Fundamentos de investigación

Resumen

En este capítulo primero se exponen los orígenes, especificaciones y arquitectura del Protocolo de Aplicaciones Inalámbricas (WAP), el cual permite a los dispositivos móviles tener acceso a Internet y otros servicios. En segundo lugar se presentan una breve descripción de los lenguajes de marcado, una explicación del por qué son los más adecuados para crear interfaces para dispositivos portátiles, y finalmente un análisis de los lenguajes más importantes de este tipo. Después se analizan XML y sus lenguajes asociados utilizados por dispositivos móviles. Finalmente, se describen brevemente las versiones de Java que se han desarrollado para este tipo de dispositivos.

2.1 WAP (*Wireless Application Protocol*)

En los últimos años la tendencia en tecnología ha sido proveer a los usuarios con servicios de comunicación mediante dispositivos inalámbricos de tamaño de bolsillo. WAP es un protocolo de comunicaciones y ambiente de aplicación que permite a los dispositivos móviles tener acceso a recursos de información, servicios avanzados de telefonía e Internet. Inicialmente fue creado por Phone.com, Ericsson, Nokia y Motorola [Arehart et al. 2000].

Precursores de WAP: HDML e i-Mode

HDML (*Handheld Device Markup Language*)

Este lenguaje fue introducido en 1995 por Unwired Planet (actualmente Phone.com) para definir contenido y aplicaciones para dispositivos portátiles con pantallas pequeñas. Es una versión reducida de HTML, diseñado para extender la infraestructura y protocolos del World Wide Web proporcionando un lenguaje de marcado eficiente para dispositivos inalámbricos y otros aparatos portátiles, tomando en cuenta sus capacidades y limitaciones. Utilizando la infraestructura del Internet, ofrece un medio eficiente para proporcionar contenido a teléfonos celulares, localizadores y PDAs [Hyland 1997].

HDML proporciona un modelo de navegación explícita que no se basa en el contexto visual, introduciendo en sus interfaces de usuario la metáfora del conjunto de tarjetas

[Hyland 1997].

i-Mode

Este servicio fue introducido a principios de 1999 por NTT DoCoMo, la compañía de telecomunicaciones más grande de Japón. Ofrecía navegación por Internet e *email* desde teléfonos celulares [WestCyber Corporation, 2000]. Los celulares que cuentan con el servicio i-Mode utilizan un navegador basado en HTML y un sistema GPRS (*General Packet Radio Switching*), el cual ofrece conexión continua y mayor capacidad. i-Mode se convirtió en una tecnología muy popular, con casi siete millones de usuarios disfrutando de servicios de Internet desde dispositivos móviles [Arehart et al. 2000].

Tanto HDML como i-Mode plantearon a los fabricantes la cuestión de decidir si la mejor tecnología era aquélla que brindaba la mejor solución a un problema específico o la que contaba con mayor número de adeptos. Unwired Planet tomó la iniciativa de involucrar a las compañías manufactureras más grandes de teléfonos celulares en su proyecto y formar el Foro de WAP. Como resultado de esta alianza surgieron las especificaciones de WAP, su arquitectura de aplicación y un nuevo lenguaje de marcado, WML (*Wireless Markup Language*) [Arehart et al. 2000].

2.1.1 Especificaciones de WAP

Las especificaciones de WAP incluyen capas complementarias de aplicación, sesión, transacción, seguridad, transporte y protocolo. Estos protocolos minimizan los problemas relacionados con el uso de los protocolos de Internet para la transmisión inalámbrica de datos, pues utilizan código binario para reducir la cantidad de datos a enviar. Por lo tanto son adecuados para el ancho de banda angosto utilizado en la comunicaciones inalámbricas [Arehart et al. 2000].

Las capas de WAP se ilustran en la Figura 2.1 y se describen brevemente a continuación:

Capa de aplicación: proporciona un ambiente de aplicación destinada al desarrollo y la ejecución de aplicaciones y servicios portátiles. WML pertenece a esta capa.

Capa de sesión: provee métodos para el intercambio organizado de contenido entre las aplicaciones cliente/servidor.

Capa de transacción: proporciona diferentes métodos para efectuar transacciones, en un grado variable de confiabilidad.

Capa de seguridad: es una capa opcional que proporciona autentificación, privacidad, y conexiones seguras entre aplicaciones.

Capa de transporte: es la última capa, y sirve de enlace entre las capas superiores y los servicios de soporte.

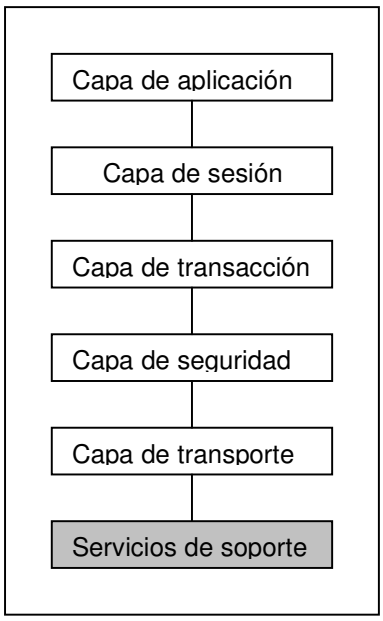

Figura 2.1 Capas de WAP (adaptado de [Arehart et al. 2000])

2.1.2 Arquitectura de Aplicación WAP

Los protocolos WAP se diseñaron de manera parecida a los protocolos de Internet. El objetivo inicial de WAP era aprovechar la infraestructura existente de Internet, pero proporcionando comunicación entre los proveedores de contenido y los dispositivos móviles de manera más eficiente y más rápida de lo que sería si se usaran los protocolos de Internet [Arehart et al. 2000].

Algunos conceptos importantes en la arquitectura WAP son [Arehart et al. 2000]:

Dispositivo WAP: es el dispositivo físico que se usa para tener acceso a las aplicaciones y contenido disponibles en los sitios WAP; puede ser un teléfono móvil,

PDA o una computadora de bolsillo.

Cliente WAP: es la entidad que recibe contenido de Internet mediante un *gateway* de WAP; es generalmente, pero no necesariamente, un navegador de WAP.

Navegador WAP: es el *software* que corre en el dispositivo WAP, interpreta el contenido WAP de Internet y decide cómo desplegarlo en la pantalla. Los navegadores WAP funcionan para todos los dispositivos WAP, y son conocidos también como micronavegadores (*microbrowsers*). Asimismo, existen emuladores de navegadores WAP que pueden ejecutarse en PCs.

Gateway de WAP: es el elemento que actúa como un intérprete entre un dispositivo WAP y el servidor origen, permitiéndoles comunicarse. Generalmente reside dentro de la red de la compañía telefónica, sin embargo cualquiera puede instalar su propio *gateway*.

Servicios de soporte: son las diferentes formas en que un teléfono móvil puede comunicarse con su red inalámbrica. Para enviar y recibir información desde el servidor origen, los teléfonos tienen que establecer un cierto tipo de comunicación con el *gateway* de WAP. El *gateway* debe dividir la información que va a enviarse al teléfono en muchos mensajes pequeños, de manera parecida a como un módem se comunica con el proveedor de Internet.

Servidor origen: es el elemento donde reside el contenido de Internet que se envía a los clientes WAP cuando así lo soliciten. Un servidor Web es un servidor origen que proporciona contenido HTML o bien contenido WAP, si se configura apropiadamente.

Existen dos formas de tener acceso a Internet usando un dispositivo WAP, las cuales se muestran en la Figura 2.2 [Arehart et al. 2000]. WAP heredó de Internet el paradigma de cliente-servidor. La principal diferencia es la presencia de un *gateway* de WAP que traduce entre HTTP y WAP.

Para accesar una aplicación almacenada en un servidor, el cliente inicia una conexión con el *gateway* de WAP, y envía una solicitud de contenido. El *gateway* convierte las solicitudes que llegan del cliente WAP a HTTP y lo manda al servidor origen. De regreso,

el servidor envía el contenido al *gateway*, el cual lo traduce al formato WAP, y entonces lo envía al dispositivo móvil. El *gateway* permite a Internet *hablar* con la red inalámbrica.

Uno de los principales objetivos de WAP es proveer un protocolo capaz de adaptarse a cualquier tipo de dispositivo móvil. La conexión entre el dispositivo WAP y el *gateway* se establece por medio de los servicios de soporte, aunque esto puede afectar la velocidad de conexión.

Al igual que en Internet, en WAP el contenido y las aplicaciones residen en los servidores origen. La diferencia consiste en que se envían a los clientes en forma de archivos WML y WMLScript, no como archivos HTML.

Los archivos WML o WMLScript se envían al cliente WAP mediante un *gateway* de WAP, el cual traduce el contenido al formato apropiado para anchos de banda angostos (ver Figura 2.3). Los clientes WAP contienen un micronavegador que despliega al usuario la información recibida.

2.2 Lenguajes de marcado

Un lenguaje de marcado (*markup language*) permite que un texto sea marcado con símbolos especiales (o *tags*), los cuales le indican a un programa (por ejemplo, a un navegador) cómo desplegar el texto [Harmonia 2000].

Ventajas. Los lenguajes de marcado presentan las siguientes ventajas [Phanouriou 2000]:

- *Proporcionan un alto grado de portabilidad*: esto permite que puedan distribuirse interfaces de usuario multiplataformas en Internet.
- *Pueden ser usados por programadores novatos*: a diferencia de los lenguajes imperativos, los lenguajes de marcado requieren poca experiencia en programación.
- *Pueden crearse con editores gráficos*: de esta manera no es necesario memorizar el vocabulario del lenguaje.
- *No necesitan procesamiento adicional*: a diferencia de los lenguajes imperativos, no requieren compilación.
- *Acceso rápido*: se cargan rápidamente pues necesitan menos espacio de

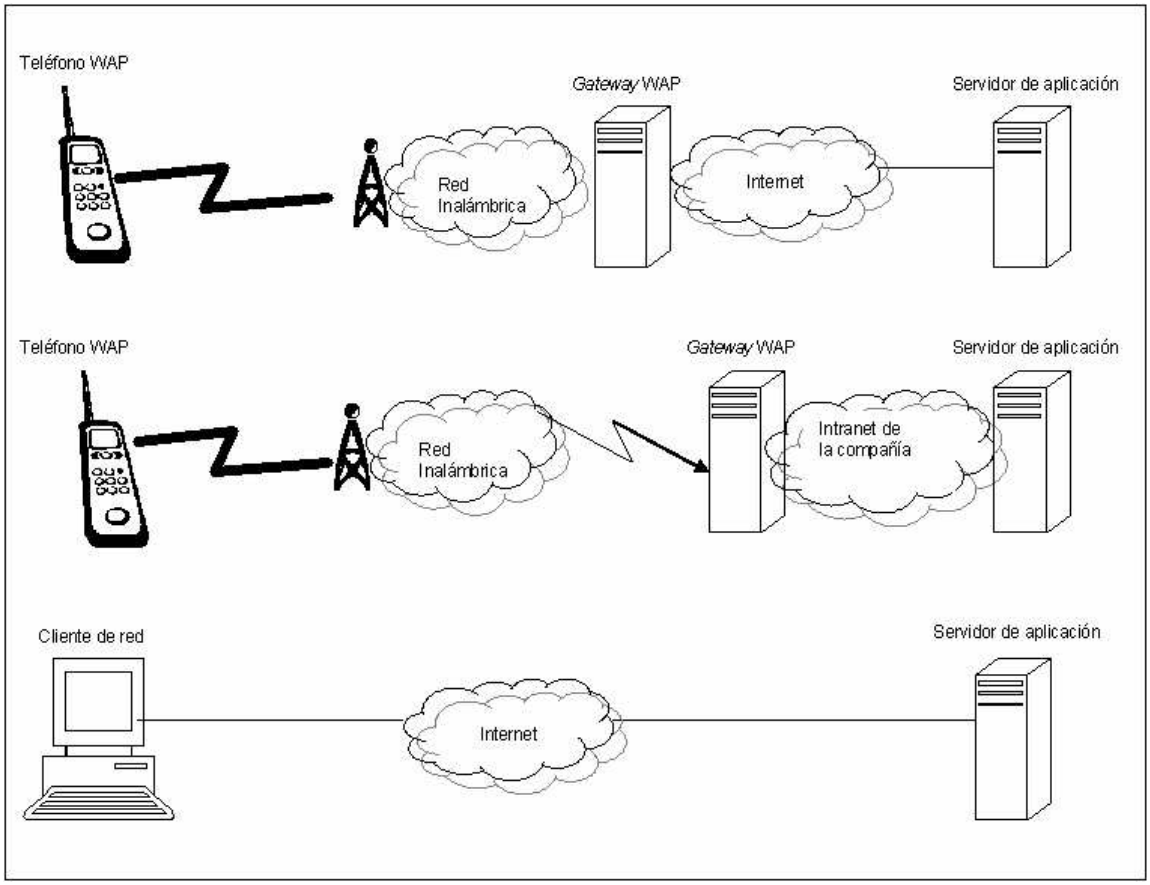

Figura 2.2 Arquitectura WAP (adaptado de [Arehart et al. 2000])

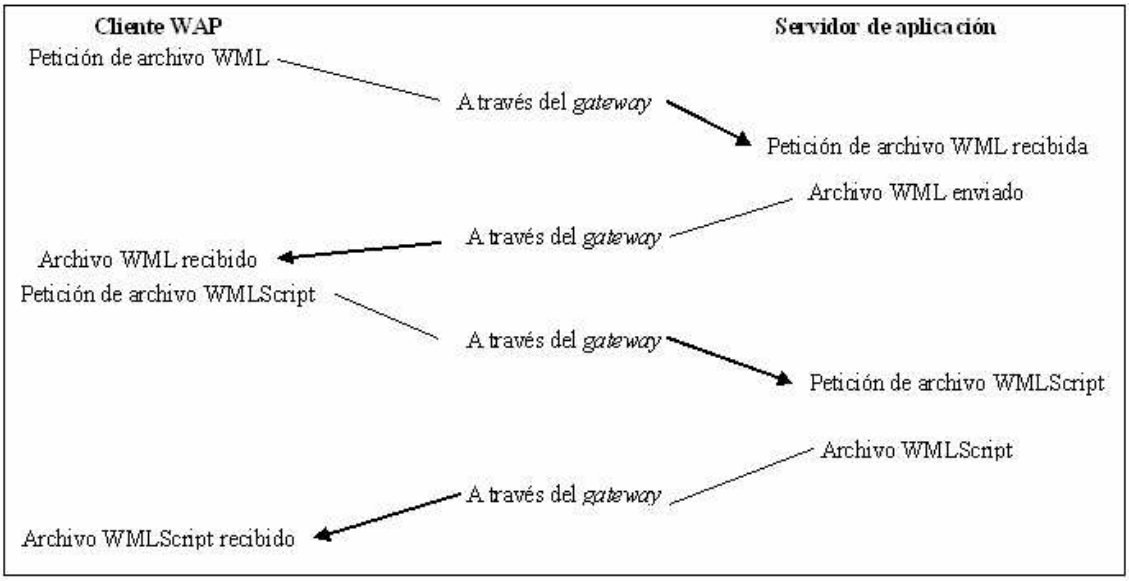

Figura 2.3 WML y WMLScript (adaptado de [Arehart et al. 2000])

almacenamiento que los códigos compilados.

- *Se almacenan como texto y son resistentes a errores de bits:* si se presenta algún error, el daño es muy pequeño y en la mayoría de los casos recuperable.
- *Son independientes del dispositivo:* la información independiente del dispositivo se almacena en un archivo separado llamado *hoja de estilo* (*stylesheet*). Este archivo permite agregar estilo de presentación, por ejemplo tipo de letra, color de fondo, etc.

La principal diferencia entre los lenguajes declarativos y los imperativos es que los imperativos especifican a detalle cómo realizar una tarea; los declarativos, sólo especifican la tarea. Al ser los lenguajes de marcado declarativos, el intérprete que los ejecuta debe decidir cómo desplegar el contenido para una plataforma en particular. Otra diferencia es que los lenguajes declarativos no cuentan con todas las funciones de los imperativos. Además, algunos lenguajes de marcado permiten ligar la descripción de un documento a la semántica en una hoja de estilo. Si se requiere aún más control en el estilo, se puede modificar la semántica del lenguaje para que cuente con características imperativas, o bien para que permita el uso de *scripts*. Por esta razón la mayoría de los lenguajes de marcado permiten la utilización de *scripts* [Phanouriou 2000].

2.2.1 Lenguajes *script*

En [Phanouriou 2000] se presenta un análisis detallado sobre los lenguajes *script* y de marcado. Tomando como referencia este documento, esta sección y la siguiente comentan las características principales de los lenguajes *script* y el uso de los lenguajes de marcado para interfaces de usuario portátiles.

Los lenguajes de marcado carecen de algunas características importantes como condiciones, ciclos y funciones. Aunque sus elementos tienen significado semántico, y pueden ser interpretados como estatutos imperativos, los lenguajes de marcado no fueron diseñados para ser lenguajes de programación. Los lenguajes *script* son un buen complemento para los lenguajes de marcado y la combinación de ambos proporciona la funcionalidad necesaria para construir la mayoría de las aplicaciones.

Los lenguajes *script* cuentan con muchas de las ventajas de los lenguajes de marcado. Son portátiles, fáciles de usar y no requieren compilación. Por esta razón son ideales para

implementar las funciones usadas por las interfaces de usuario.

2.2.2 Lenguajes de marcado para interfaces de usuario portátiles

Para que un lenguaje garantice portabilidad en dispositivos y sistemas operativos, debe proporcionar un alto nivel de abstracción. Es decir, debe evitar que el programador conozca todos los detalles de implementación, de manera que disminuya la experiencia y tiempo necesarios para desarrollar *software*.

Actualmente las interfaces de usuario se implementan en un número cada vez mayor de tecnologías de *software* y *hardware*. Aunque los lenguajes declarativos son más abstractos que los imperativos, para que sean útiles en el desarrollo de interfaces de usuario deben permitir la transparencia de los siguientes elementos:

- arquitectura usada para desplegar la interfaz (Java o Motif)
- tecnología de presentación (pantalla grande o chica)
- aplicación de la interfaz (*script* o sistema distribuido)
- red que conecta la interfaz con la aplicación (local o inalámbrica)
- sistema operativo

Los lenguajes de marcado son lenguajes declarativos que proporcionan el nivel de abstracción necesario para garantizar portabilidad. Por lo tanto, para que las interfaces de usuario sean portátiles en dispositivos y sistemas operativos, es preferible que se desarrollen utilizando un lenguaje de marcado.

2.2.3 El lenguaje de marcado de hipertexto (*HyperText Markup Language*, HTML)

El lenguaje de marcado para hipertexto (HTML) es un lenguaje simple que revolucionó la publicidad electrónica permitiendo a programadores no profesionales distribuir información en Internet. Se le considera el lenguaje del Internet, pues define un formato de documentos para Internet flexible, portátil y práctico [Kamada 1998].

La cuarta versión de HTML cuenta con las siguientes ventajas [Evans 1998]:

- Permite el formateo de documentos en varios estilos
- Incluye hiperligas que apuntan a otros documentos, archivos o servicios
- Contiene una amplia gama de capacidades de organización de sus componentes

- Permite crear tablas y texto preformateado
- Permite incluir imágenes
- Contiene características interactivas que involucran al usuario
- Hojas de estilo de cascada (*Cascading Style Sheets*, CSS), las cuales sirven para agregar estilo, por ejemplo tipo de letra, color de fondo, etc.

Sin embargo, también presenta dos limitaciones importantes [Evans 1998]:

- Un documento HTML se despliega de manera diferente en cada dispositivo
- Cada tipo navegador despliega una página HTML de manera diferente

Metáfora. Una metáfora es una forma de describir un concepto de manera más accesible y familiar para facilitar la comprensión de un tema desconocido o adaptarse a una nueva situación. En el área de Interacción Humano Computadora se han utilizado varias metáforas para diseñar interfaces se usuario [Preece 1994]. HTML emplea la metáfora de páginas [Evans 1998].

Script. Permite manejar JavaScript y VBScript [Evans 1998].

2.2.4 CompactHTML

CompactHTML es un subconjunto bien definido de HTML 2.0, HTML 3.2 y HTML 4.0, diseñado para aparatos de información pequeños [Kamada 1998].

Metáfora. CompactHTML emplea la metáfora del conjunto de tarjetas. El usuario despliega y/o interactúa con las tarjetas, utilizando un modelo de navegación similar al de HTML, en el cual hay una noción de movimiento "hacia delante" y "hacia atrás". "Adelante" significa desplegar la siguiente tarjeta, y "atrás" desplegar la tarjeta anterior [Arehart et al. 2000].

El sistema de organización en conjuntos de tarjetas y no en páginas como en HTML se debe a la latencia de los dispositivos inalámbricos. Las demoras en la conexión con el servidor llegan a ser inaceptables en un ambiente inalámbrico. Regresando varias tarjetas a la vez, los viajes al servidor se reducen y es posible almacenar contenido adicional localmente en el dispositivo [Arehart et al. 2000].

Script. Permite manejar JavaScript y VBScript [Kamada 1998].

Características. Las características de HTML 4.0 que se han excluído de CompactHTML son [Kamada 1998]:

- Imágenes JPEG (pero sí maneja imágenes GIF)
- Tablas
- Múltiples estilos de letras
- Color e imagen de fondo
- Marcos
- Hojas de estilo

Interfaces. Las interfaces de CompactHTML pueden visualizarse en pantallas pequeñas y monocromáticas. Puede operarse fácilmente, porque CompactHTML está diseñado para que todas las operaciones básicas puedan ejecutarse combinando cuatro botones: cursor hacia adelante, cursor hacia atrás, seleccionar y atrás/detener [Kamada 1998].

Complejidad del lenguaje. CompactHTML está basado completamente en las recomendaciones W3C de HTML. CompactHTML es un subconjunto bien definido de las especificaciones HTML 2.0, HTML 3.2 y HTML 4.0, y por lo tanto hereda la flexibilidad y portabilidad de HTML estándar. Además, fue diseñado para implementarse con poca memoria y bajo poder de CPU [Kamada 1998].

2.2.5 HTML Dinámico (Dynamic HTML, DHTML)

DHTML es el término de mercadotecnia aplicado a una mezcla de estándares incluyendo HTML, hojas de estilo, Modelo de Objetos de Documentos (*Document Object Model*, DOM) y *scripts*. Sin embargo, no hay especificación de W3C que defina formalmente a DHTML [Chisholm 1999].

2.3 Estándares para lenguajes de marcado: SGML y XML

Los dos estándares más populares de sintaxis para los lenguajes de marcado son el Lenguaje de Marcado Estándar Generalizado (*Standard Generalized Markup Language*, SGML) y el Lenguaje de Marcado Extensible (*eXtensible Markup Language*, XML) [Phanouriou 2000].

Lenguaje de Marcado Estándar Generalizado (*Standard Generalized Markup Language*, SGML)

El Lenguaje de Marcado Estándar Generalizado (SGML) es un lenguaje muy complejo que se usa para definir otros lenguajes de marcado, como HTML. Permite que la sintaxis de un lenguaje sea especificada de manera exacta y completa. Sin embargo, no especifica el significado de cada elemento, ni describe cómo se manejan los elementos en un dispositivo que soporte el lenguaje. Por ejemplo, SGML permite especificar que el elemento <table> contiene los elementos <tr> y <td>, en tanto en HTML se especifica que con el elemento <table> el navegador debe desplegar una tabla. El navegador necesita conocer el significado de cada elemento para desplegarlo apropiadamente [Arehart et al. 2000].
SGML surgió a mediados de los 80's, pero nunca recibió gran aceptación debido principalmente a su complejidad [Phanouriou 2000].

XML (*eXtensible Markup Language*)

El Lenguaje de Marcado Extensible (XML) fue desarrollado en 1996 por un grupo de trabajo auspiciado por el consorcio del World Wide Web. El principal objetivo de este grupo fue crear un lenguaje que permitiera el intercambio de información independiente de las plataformas [Wood 1998].

Aunque XML es compatible con SGML, sus elementos no tienen significado intrínseco. El poder de XML proviene precisamente de esta característica: XML es un subconjunto, no una aplicación de SGML. Por una parte puede usarse para contener información, y por otra, como base para aplicaciones que usen XML. La gama de aplicaciones que XML soporta es muy amplia, y sus atributos opcionales son mínimos [Arehart et al. 2000].

2.3.1 Documentos XML

XML describe un tipo de objetos de datos llamados documentos XML, los cuales almacenan información de manera estructurada o semiestructurada. En XML se describe la estructura y la organización de los datos, no la forma como se despliegan. El diseño de un documento XML es formal, conciso y se prepara rápidamente. Un documento XML es legible, claro y fácil de crear, por lo que también es muy sencillo escribir programas que procesen este tipo de documentos [Arehart et al. 2000].

Además del contenido del documento, XML contiene metadatos, es decir, datos que describen el contenido. Se puede crear un documento XML que contenga información usando elementos con el significado que se quiera. Mientras tanto el emisor como el receptor sepan qué significan los elementos, un documento XML puede usarse para almacenar y transmitir información independientemente de las plataformas, sistemas operativos y dispositivos. Además, como un documento XML es sólo texto, puede transmitirse directamente en cualquier tipo de red, incluyendo Internet [Arehart et al. 2000].

2.3.2 Estructura de un documento XML

Un documento XML tiene una estructura lógica y una física [Wood 1998]:

- *Físicamente*, el documento está compuesto de unidades de almacenamiento llamadas entidades, las cuales pueden hacer referencia a otras entidades para que sean incluidas en el documento. Un documento empieza en una "raíz" o entidad de documento.
- *Lógicamente*, el documento está compuesto de declaraciones, elementos, comentarios, referencias de caracteres e instrucciones de procesamiento.

Tanto la estructura lógica como física están indicados en el documento mediante etiquetas (*tags*) explícitas. XML provee un mecanismo para imponer restricciones sobre las estructuras. Ambas estructuras deben anidarse correctamente para ser *parseadas* exitosamente y el documento pueda considerarse "bien formado" (*well-formed*) [Wood 1998].

Un documento XML bien formado debe cumplir con las siguientes características [Martin et al. 2000]:

- Contar con al menos un elemento
- Contener un elemento raíz único que incluya a todos los demás elementos
- Todas las etiquetas deben cerrarse, es decir, cada <tag> debe tener su correspondiente </tag>
- Los elementos dentro del elemento raíz deben están correctamente anidados, esto es, sólo puede cerrarse la última etiqueta abierta (no se permite la superposición de etiquetas).

Además de estar bien formado un documento XML debe ser válido. Para definir las reglas que permiten revisar la validez de un documento XML, es necesario tener un medio para describir la estructura del documento. Existen dos formas de describirlo: mediante una Definición de Tipo de Documentos (*Document Type Definition*, DTD) o bien por medio de los Esquemas XML [Wood 1998].

DTD (*Document Type Definition*)

La técnica original para definir la estructura de un documento XML era mediante un DTD que proporcionaba información adicional sobre su estructura utilizando la sintaxis de SGML. Un DTD describe las etiquetas que pueden estar dentro de un documento, la manera como pueden anidarse e información adicional. El *parser* de XML usa la información del DTD para verificar que el documento tenga la estructura correcta, en cuyo caso se considera un documento válido [Arehart et al. 2000].

Sin embargo, los DTDs presentan las siguientes desventajas [Arehart et al. 2000]:

- Son inflexibles.
- No proporcionan el control necesario sobre la estructura y los valores permitidos de los elementos y atributos.
- No manejan el concepto de tipos de datos.
- No están escritos en un formato compatible con XML.
- No pueden definir elementos y atributos que tengan alcance local.
- No permiten la herencia de otros esquemas que tengan elementos y atributos comunes a varios documentos.

Esquemas XML

Para contrarrestar las desventajas de los DTDs se crearon esquemas escritos en formato XML y que permiten herencia y definición de atributos de alcance local. Permiten agregar descripciones a los elementos y atributos de manera que otras personas o herramientas puedan leer, construir y manipular los esquemas [Arehart et al. 2000].

Otra característica importante de los esquemas es que son compatibles con XML. Pueden crearse, *parsearse* y modificarse usando las mismas técnicas que funcionan con los

documentos XML [Arehart et al. 2000].

2.3.3 Procesamiento de documentos XML

Un *parser* de XML es un programa que toma un documento XML como entrada y hace que sus elementos y atributos puedan ser procesados por otros programas. Por ejemplo, un *parser* XML de Java convierte a los componentes de un documento XML en componentes disponibles para un programa Java. Actualmente existen *parsers* XML de diferentes compañías como Sun, IBM, Microsoft y Oracle [Wood 1998].

Un *parser* XML procesa un documento XML de dos formas [Wood 1998]:

- *En el tiempo*: mediante el API Simple para XML (*Simple API for XML*, SAX), el cual asocia un evento con cada etiqueta y con cada bloque de texto.
- *En el espacio*: con el Modelo de Objetos de Documentos (*Document Object Model*, DOM), el cual es una interfaz de programas de aplicación (API) para documentos HTML y XML. Define la estructura lógica de los documentos como un árbol completo. También define la manera como se debe tener acceso y manipular los documentos.

2.3.4 Transformación de XML a otros formatos: el Lenguaje de Hojas de Estilo Extensible (*eXtensible Stylesheet Language*, XSL)

El lenguaje de Hojas de Estilo Extensible (*Extensible Stylesheet Language*, XSL) es una aplicación de XML. Permite transformar el contenido de cualquier documento XML a otro formato, aplicando la información del estilo. En una hoja de estilo XSL se especifica el estilo, la distribución y paginación de un documento XML o de un archivo de datos, los cuales pueden desplegarse en un navegador, un dispositivo manual o en las páginas físicas de un catálogo, reporte, panfleto o libro [Adler 2001]. Está formada por plantillas (*templates*) que se cotejan con los elementos del documento XML para determinar el formato del documento final [Martin et al. 2000].

XSL consta de dos partes, el lenguaje de Transformación XSL (XSLT), y un vocabulario para especificar el formato. El vocabulario es una colección de elementos especializados

llamados objetos de formateo, los cuales especifican los detalles de la presentación del documento [Clark 1999].

El lenguaje de Transformación de XSL (*XSL Transformation Language*, XSLT)

XSLT es un lenguaje que define un conjunto de elementos XML que pueden usarse para crear hojas de estilo que transformen documentos XML en otros documentos XML. XSL especifica el estilo de un documento XML usando XSLT para describir cómo el documento se transforma en otro documento XML que use el vocabulario de formateo [Clark 1999].

Para realizar la transformación es necesario un *procesador XSLT* (XSLT *engine*). Los procesadores XSLT no manipulan documentos; manipulan la estructura del documento [Martin et al. 2000].

El procesador XSLT necesita la estructura de árbol del documento XML y la hoja de estilo XSL. Analiza las plantillas del archivo XSL y los nodos del árbol XML; cuando coinciden, procesa el nodo conforme a las reglas contenidas en la plantilla, generando otro árbol como salida. Finalmente, traduce el árbol resultante a un documento XML, HTML, texto o el formato que se requiera [Martin et al. 2000]. Este procesamiento se ilustra en la Figura 2.4.

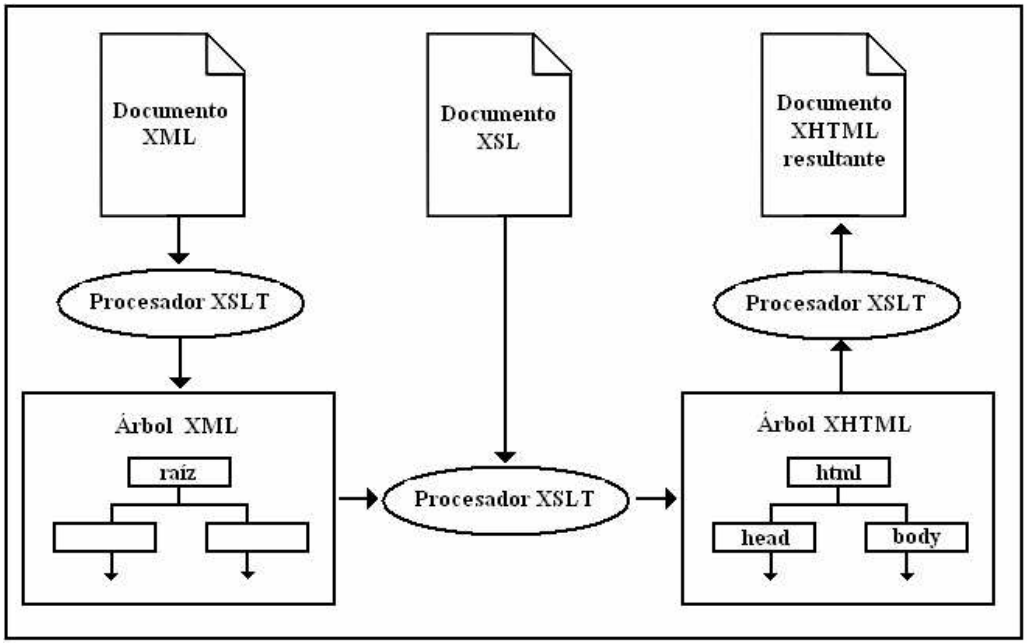

Figura 2.4 Transformación XSLT de un documento XML a XHTML

2.4 Lenguajes basados en XML para el diseño de interfaces

Casi todos los lenguajes de marcado que han surgido últimamente son aplicaciones de XML. XML proporciona la sintaxis general; cada lenguaje, el vocabulario y la semántica. Los lenguajes de marcado compatibles con XML funcionan en una amplia gama de aplicaciones, como bases de datos, desarrollo en Internet, búsquedas e interfaces de usuario [Phanouriou 2000].

2.4.1 Lenguaje de marcado de hipertexto extensible (*eXtensible HyperText Markup Language*, XHTML)

XHTML es un lenguaje basado en XML que maneja documentos y módulos que reproducen y extienden HTML 4 [Pemberton 2000]. Se le considera el sucesor de HTML, pues fue diseñado para trabajar en una amplia gama de plataformas de navegadores, incluyendo celulares, televisiones, carros, comunicadores de tamaño de bolsillo y computadoras de escritorio [Masayasu 2002].

Metáfora. XHTML Utiliza la metáfora de páginas, al igual que HTML [Phanouriou 2000].

Características. Tres aspectos de XHTML promueven la compatibilidad e interoperabilidad [Arehart et al. 2000]:

- Provee todos los efectos de desplegado usando un conjunto reducido de elementos de HTML estándar; esto facilita a los fabricantes, desarrolladores de código y usuarios la construcción de aplicaciones que sean completamente compatibles con el estándar XHTML.
- Los documentos XHTML requieren un estricto conjunto de reglas de sintaxis basadas en XML, lo que evita que los dispositivos tengan que lidiar con documentos que no sean bien formados; de esta manera es fácil desarrollar dispositivos y aplicaciones que los soporten.
- Divide el conjunto de elementos disponibles de HTML en módulos o grupos. Todos los dispositivos deben soportar un conjunto estándar de módulos "base"; también existen módulos opcionales. Esta modularización facilita a los desarrolladores y fabricantes de dispositivos la tarea de especificar o descubrir las capacidades exactas de un

dispositivo.

Scripts. Los documentos XHTML pueden utilizar *scripts* o *applets* que cumplan con los requerimientos DOM de HTML o de XML [Pemberton 2000].

Interfaces. XHTML permite utilizar los siguientes elementos en sus interfaces [Baker 2000]:

- Texto
- Hiperligas
- Formas
- Tablas
- Imágenes
- Metadatos

Complejidad. No todos los elementos de XHTML pueden ser requeridos por todas las plataformas; ciertos dispositivos pueden soportar sólo un subconjunto de elementos de XHTML. El proceso de modularización divide XHTML en una serie de conjuntos más pequeños de elementos que pueden recombinarse para satisfacer las necesidades de diferentes comunidades. Esta modularización cuenta con las siguientes ventajas [Pemberton 2000]:

- Proporciona un mecanismo formal para subdividir y extender HTML
- Simplifica la transformación entre tipos de documentos
- Promueve la reutilización de módulos en nuevos tipos de documentos

En XHTML es muy fácil introducir nuevos elementos o atributos adicionales a través de módulos. Estos módulos permiten la combinación de características nuevas con existentes cuando se desarrolla contenido y se diseñan nuevas implementaciones que procesen y recuperen documentos XHTML [Pemberton 2000].

2.4.2 El lenguaje extensible de interfaces de usuario (*eXtensible User interface Language*, XUL)

XUL es un lenguaje basado en XML que describe las interfaces de usuario del proyecto XPFEToolkit de Mozilla. Este proyecto intenta crear aplicaciones multiplataformas, como

navegadores y clientes de correo electrónico, a partir de un conjunto de herramientas diseñadas específicamente para ese propósito. Su intención no es implementar una estructura para aplicaciones multiplataformas, lo que intenta es proveer un subconjunto de funcionalidad multiplataforma adecuado para construir aplicaciones de red como navegadores [Mozilla Org. 1999].

Características. Permite construir todos los elementos de control de ventanas y de diálogos, como barras de tareas, árboles, barras de porcentaje de avance y menúes. Se utiliza para simplificar el desarrollo de interfaces para nuevas aplicaciones que se ejecuten en un navegador Netscape [Phanouriou 2000].

Interfaces. Con XUL la descripción de las interfaces de usuario se crea independientemente de la aplicación, en un archivo aparte. Las ventanas se despliegan y manejan por el mismo motor de organización que manipula el contenido HTML en los navegadores [Mozilla Org. 1999].

Complejidad del lenguaje. XUL es una aplicación de XML, de hecho, es sólo XML adicionado con la definición de unos cuantos elementos [Mozilla Org. 1999].

2.4.3 Lenguaje de Marcado Inalámbrico (WML, *Wireless Markup Language*)

El Lenguaje de Marcado Inalámbrico (WML) forma parte del Protocolo de Aplicación Inalámbrico (WAP). WAP es una iniciativa para definir una especificación para el desarrollo de aplicaciones que operen en redes de comunicación inalámbricas. El propósito de WML es proporcionar contenido e interfaces de usuario a dispositivos con pantallas pequeñas y ancho de banda limitado, como celulares y localizadores [Phanouriou 2000].

WML está basado en etiquetas, comparte algunos elementos de HTML, y se define con un documento XML. Como WML se diseñó para desplegar principalmente páginas de texto y es un lenguaje de XML, está definido por y debe seguir las reglas de un DTD [WMLOrg. 2001].

Metáfora. WML usa la metáfora del conjunto de tarjetas. Cada documento WML es un conjunto de una o más tarjetas. Cuando un usuario accede un sitio WAP, el sitio regresa un conjunto de tarjetas; el usuario observa la primera tarjeta, lee su contenido, posiblemente

introduzca información, y se mueve a otra tarjeta de su elección. Esta metáfora permite a WML dividir una interfaz compleja en una secuencia de pantallas simples. De esta manera puede desplegar interfaces en aparatos con pantallas de resolución limitada [Phanouriou 2000].

Características. Aunque WML cuenta con capacidades limitadas en comparación con HTML, cuenta con una amplia gama de características [Arehart et al. 2000]:

- *estilo de texto*: maneja negritas, itálicos, subrayados, tablas y fin de línea.
- *imágenes:* utiliza un nuevo formato, WBMP (*wireless bitmap*), en blanco y negro.
- *entrada de datos*: las tarjetas pueden contener elementos que permitan al usuario introducir texto, escoger en una lista de opciones, navegar entre las tarjetas o ir a ligas específicas.
- *variables*: pueden incluirse en el código WML para mantener información oculta y manipular la entrada de datos del usuario.
- *navegación:* incluye facilidades para navegar y registro de historia.
- *soporte internacional*: el conjunto de caracteres que maneja es UNICODE, el cual usa 16 bits para representar cada caracter.
- *optimización para banda angosta:* WML ha sido diseñado para adaptar las características del ancho de banda angosto y de latencia alta de las redes inalámbricas. Las conexiones con el servidor origen se evitan mediante el uso de varias soluciones: variables que conserven su valor en más de una tarjeta, agrupación en conjuntos de tarjetas, y validación de los datos de entrada con WMLScript.
- *Manejo de errores*: difiere de HTML. Los navegadores de Internet comúnmente permiten errores en el código HTML; los navegadores WAP, no [Arehart et al. 2000].

Interfaces dinámicas. WML ofrece las siguientes funcionalidades [Phanouriou 2000]:

- Etiquetado dinámico de los elementos de la interfaz (aunque los dispositivos no lo soporten). No obstante, no permite el cambio de tipos de elementos.
- Una vez que el código de una tarjeta se ha bajado al dispositivo, solamente las etiquetas de los elementos pueden cambiarse; todo lo demás requiere una llamada al

servidor origen.

- Las tarjetas se direccionan por medio de URLs, las cuales pueden estar ligadas a archivos CGI o JSP. Esto permite la personalización.
- El formato de las tarjetas depende del navegador [WMLOrg. 2001].

Interacción con el usuario. La interacción en una interfaz WML se realiza de la siguiente manera [Phanouriou 2000]:

- Los usuarios seleccionan alguno de los elementos de una tarjeta y después navegan a la siguiente.
- Los usuarios sólo pueden interactuar con una tarjeta a la vez, no pueden saltar a otra de manera arbitraria, a menos que la tarjeta actual contenga una liga a esa tarjeta.
- El navegador mantiene el estado de WML durante una sesión, lo cual incluye almacenar variables de datos, el registro de la historia, e información del usuario y del dispositivo.
- WML cuenta con un comando para inicializar el contexto actual y e iniciar uno nuevo.

Complejidad del lenguaje. En lo que se refiere a la complejidad, WML cuenta con las siguientes características [Phanouriou 2000]:

- Contiene 35 elementos, 25 obligatorios y 10 opcionales; los navegadores WML deben soportar los obligatorios y pueden ignorar los opcionales.
- Soporta manejo de eventos, parametrización de cadenas y manejo de estados.
- Permite formatear el texto.
- No necesita otro lenguaje para presentar la interfaz, pues la interfaz está escrita completamente en WML. Las páginas Web existentes deben convertirse a WML antes de que se desplieguen en un navegador WML.
- No usa hojas de estilo, porque el estilo es fijo e integrado a la descripción de la interfaz.
- Mezcla el contenido y la presentación de la interfaz. El código WML puede estar en un archivo o dividido en varios. Cada archivo puede contener sólo un conjunto de tarjetas, pero el conjunto puede tener una o más tarjetas.
- Las tarjetas se relacionan mediante URLs.

***Script*.** WML permite el manejo de eventos, y por lo tanto permite la ejecución de WMLScript en su código [Arehart et al. 2000].

WMLScript es un lenguaje *script* procedural sencillo, basado en la versión estandarizada de JavaScript. Proporciona un conjunto de librerías (para operaciones matemáticas, manipulación de cadenas, etc.), colabora con WML y evita conexiones innecesarias con el servidor. En particular, WMLScript proporciona al programador [Arehart et al. 2000]:

- Capacidad para revisar y validar los datos entrada del usuario antes de enviarlos al servidor, impidiendo así la transmisión de información inválida.|
- Acceso a las facilidades de los dispositivos, como directorio, calendario y lista de mensajes.
- Métodos para interactuar con el usuario, como mensajes de error y avisos.

A diferencia de los archivos HTML, que pueden contener *scrip*t inscrustrado, los archivos WMLScript están separados del código WML que los invoca. Si las tarjetas contienen archivos WMLScript, éstos no son enviados al cliente con los archivos WML. Comúnmente, los archivos de WMLScript sólo se envían al cliente cuando éste intenta acceder alguna función contenida en ellos [Arehart et al. 2000].

2.4.4 VoiceXML

El desarrollo actual de la tecnología inalámbrica, servidores para el procesamiento de voz y procesadores más rápidos para celulares y PDAs, ha hecho posible la interacción con el usuario usando entrada y salida de audio. VoiceXML es un lenguaje que consiste en una serie de reglas que detallan cómo describir una transacción de voz usando un lenguaje de marcado [Arehart et al. 2000].

Para los proveedores de servicios telefónicos cada vez es más fácil brindar nuevos servicios a sus clientes explotando el crecimiento de la tecnología de Internet. VoiceXML permite a los proveedores proporcionar servicios de voz automatizados usando interfaces de voz. Además, con VoiceXML los desarrolladores pueden construir estos servicios usando la misma tecnología que usan para crear sitios visuales de Internet, reduciendo significativamente el costo de construcción [VoiceXML Org. 2001].

Características. Se utiliza para especificar aplicaciones de voz interactivas que permiten [Phanouriou 2000]:

- diálogos de audio que efectúen síntesis de voz
- digitalización de audio
- reconocimiento de voz
- entrada Multifrecuencia de Doble Tono (Dual-Tone MultiFrequency, DTMF)
- grabación de entrada de voz
- telefonía
- conversaciones de iniciativa mixta

Las ventajas de VoiceXML son [VoiceXML Org. 2001]:

- Minimiza las interacciones cliente/servidor, pues especifica múltiples interacciones por documento.
- Evita que los desarrolladores tengan que lidiar con los detalles de bajo nivel específicos de la plataforma.
- Separa la interacción con el usuario de la aplicación.
- Promueve la portabilidad de servicios a través de diferentes plataformas.
- Es fácil para desarrollar tanto interacciones simples como diálogos complejos.

Metáfora. Las interfaces de voz no pueden diseñarse efectivamente usando las metáforas de escritorio o de tarjetas. Con la voz la máquina sólo puede dar entrada o salida a un elemento de información a la vez, no hay respuestas concurrentes. Sin embargo, las respuestas no tienen que estar sincronizadas, el usuario puede responder una pregunta actual, pasada, futura o implícita (como "salir") [Phanouriou 2000].

VoiceXML usa la metáfora conversacional. Esta metáfora permite al usuario involucrarse en un diálogo con la máquina de manera similar a como se involucra en un diálogo con humanos. La metáfora conversacional resulta familiar a las personas ya que están acostumbradas a responder preguntas. También es apropiada para operaciones que requieran manos libres [Arehart et al. 2000].

***Script*.** Permite invocar *scripts* CGI del servidor, pero no permite invocar *scripts* del cliente. Soporta algunas expresiones del tipo de ECMAScript, como operaciones

aritméticas, operadores lógicos y de comparación, el operador especial ternario [(a>b)?a:b] y operaciones de cadenas [Phanouriou 2000]. Los *scripts* son útiles para validar la entrada del usuario y proporciona contenido dinámico [Arehart et al. 2000]

Interfaces dinámicas. VoiceXML puede recibir gramáticas de reconocimiento de voz dinámicamente. La información de la gramática puede estar contenida en los elementos o estar direccionada mediante identificadores de recursos uniformes (*Uniform Resource Identifiers*, URIs). De esta manera se puede reutilizar dinámicamente la misma interfaz para diferentes personas cambiando únicamente la gramática [Phanouriou 2000].

VoiceXML divide la interfaz de usuario en una serie de documentos. Cada documento cuanta con su propio URI, por lo que un servidor puede repartir documentos personalizados en cada transición [Phanouriou 2000].

Una vez que se descarga un documento VoiceXML, no puede modificarse. Sin embargo, el documento puede solicitar la gramática de forma dinámica, afectando el reconocimiento y la síntesis de voz [Phanouriou 2000].

Interacción con el usuario. En una interfaz de VoiceXML el usuario responde a comandos verbales generados por el sistema a partir de voz sintetizada o grabada. El usuario puede responder verbalmente o presionando un botón del teléfono. El sistema puede grabar la respuesta del usuario o pasarla a un motor de reconocimiento de voz [Phanouriou 2000].

Dos aspectos importantes a considerar en las interacciones de VoiceXML son [Arehart et al. 2000]:

> *Iniciativa mixta:* en una iniciativa mixta tanto el usuario como la computadora pueden afectar la conversación. VoiceXML permite iniciativas mixtas.
>
> *Interfaces multimodales:* las interfaces multimodales permiten, al usuario utilizar deferentes medios de interacción, y al diseñador presentar contenido integrando diferentes tipos de interfaces, como gráficas y de voz [Arehart et al. 2000].

Actualmente no existen navegadores que soporten el cambio de un lenguaje de marcado de texto, como WML, a uno basado en voz, como VoiceXML. Sin embargo, es posible que en el futuro existan navegadores que soporten la combinación de ambos tipos de lenguajes. De

esta manera se podrían desarrollar interfaces multimodales usando VoiceXML [Arehart et al. 2000].

La tecnología multimodal sería particularmente adecuada para las comunicaciones inalámbricas, aunque este tipo de aplicaciones presentarían las mismas limitaciones de los dispositivos inalámbricos. En las aplicaciones basadas en WAP los usuarios podrían posicionarse en algún campo usando el teclado, y llenar el contenido usando su voz [Arehart et al. 2000].

Complejidad del lenguaje. VoiceXML contiene 47 elementos y soporta el manejo de eventos. Los eventos pueden ser ejecutados por las plataformas, por el intérprete, o explícitamente por el elemento <throw> del lenguaje. VoiceXML también permite utilizar variables y evaluar expresiones simples. El manejo de eventos se utiliza para invocar a scripts CGI o accesar nuevos documentos VoiceXML. VoiceXML describe la estructura y el comportamiento de la interfaz de usuario, pero necesita una gramática externa para reconocer y generar voz. El estilo se especifica en otro documento. Las descripciones de estilo pueden estar en uno o varios archivos. Cada archivo puede contener sólo un elemento, pero el documento puede tener uno o más diálogos. Una aplicación puede abarcar varios documentos [Phanouriou 2000].

2.4.5 Lenguaje de marcado para interfaces de usuario (UIML, *User Interface Markup Language*)

UIML es un lenguaje declarativo compatible con XML creado en 1997 por Harmonia Inc. para definir interfaces de usuario. El principal objetivo de UIML es describir interfaces de manera independiente a los dispositivos. Los dispositivos pueden ser computadoras personales, manuales, teléfonos de escritorio, teléfonos celulares, o cualquier otro aparato con el que un humano pueda interactuar [Harmonia 2000].

Características. Introduce el modelo de Meta-Interfaz para separar la interfaz de usuario de la aplicación (ver Apéndice A). También introduce varias abstracciones nuevas que simplifican el desarrollo de interfaces de usuario [Phanouriou 2000]:

- Permite a los programadores describir separadamente la estructura, contenido y comportamiento de la interfaz de usuario.

- Describe el comportamiento de ejecución en términos abstractos (eventos dependientes de la plataforma y tareas genéricas).
- Permite un diseño basado en componentes dividiendo la descripción de la interfaz en una o más platillas reusables.

Metáfora. UIML describe una interfaz de usuario en términos abstractos y no está atado a ninguna metáfora en particular. En términos de interfaces de usuario, UIML es un lenguaje independiente de metáforas. Los usuarios pueden seleccionar su propia metáfora y darle un estilo de descripción. De esta manera, con UIML se puede desplegar la descripción de una interfaz usando múltiples metáforas, como la de escritorio y la del conjunto de tarjetas, sin que se requiera programación adicional [Phanouriou 2000].

Scripts. UIML es un lenguaje declarativo, por lo que una operación imperativa requiere una definición externa. Aunque UIML permite asignaciones y comparaciones lógicas, no permite realizar cálculos de evaluación ni aritméticos. En su lugar, UIML permite la declaración de tareas genéricas que se llaman durante la ejecución, las cuales en conjunto describen el comportamiento de la interfaz. El mapeo de la declaración de cada tarea a su definición actual se hace dinámicamente. UIML permite acceder *scripts* del cliente y *scripts* o programas ejecutables del servidor. De esta manera una tarea puede mapearse a un *script* del cliente durante una sesión y a un objeto ejecutable remoto en otra. La abstracción de tareas permite a UIML crear interfaces de usuario para una amplia gama de aplicaciones, lo cual incluye *scripts*, aplicaciones de servidor basadas en componentes y sistemas distribuidos [Phanouriou 2000].

Interfaces dinámicas. UIML proporciona varios mecanismos que permiten crear interfaces de usuario dinámicas [Phanouriou 2000]:

- Maneja el concepto de plantillas, las que permiten el diseño de componentes de interfaces reusables, y por lo tanto customizables para usos particulares [Interface Technologies Inc. 2000]. Este mecanismo de plantillas permite que algunas partes de la descripción de interfaz residan en archivos diferentes, identificados mediante un URI. Las plantillas se pueden descargar de un servidor dinámicamente, permitiendo así un alto grado de personalización. UIML permite declaraciones múltiples de cada uno de los elementos de la interfaz (estructura, estilo, contenido y

comportamiento). El usuario puede decidir cuál declaración usar al momento de la ejecución.

- Un documento UIML se representa como un árbol de XML-DOM. Este árbol puede modificarse mediante un programa permitiendo así que la aplicación altere cualquier aspecto de la interfaz durante la ejecución, incluyendo su estructura, estilo y comportamiento.
- Para crear una interfaz, se escribe un documento UIML, que incluye el estilo de presentación adecuado para los dispositivos en los cuales se va a desplegar la interfaz. UIML se mapea automáticamente al lenguaje usado por el dispositivo.
- UIML permite definir la posición y diseño los botones, menúes, listas y otros controles que permiten a un programa funcionar en una interfaz gráfica como Windows o Motif. También define las acciones que deben ejecutarse cuando suceden ciertos eventos. Los usuarios crean los eventos cuando interactúan con la interfaz oprimiendo una tecla o moviendo el ratón [Harmonia 2000].
- El contenido de la interfaz -texto, imágenes, sonido- no está incrustado en UIML, pues se recoge de fuentes externas (archivos, bases de datos, directorios). La definición de la interfaz de usuario contiene referencias (o variables) que almacenarán contenido cuando el documento se despliegue [Phanouriou 2000].

En UIML la interfaz es un conjunto de elementos con los que interactúa el usuario final. Cada elemento de la interfaz se conoce como *parte* (*part*). Estas partes o secciones pueden organizarse de manera diferente para diferentes categorías de usuarios finales y diferentes familias de dispositivos. Cada sección tiene un contenido (sonidos, imágenes), que proporciona información al usuario. Las secciones también pueden recibir información del usuario mediante los elementos con los que cuente el dispositivo [Phanouriou 2000]. En el Apéndice B se detallan los componentes de cada sección de UIML.

En [Aragón 2002] se encuentran los resultados de una investigación orientada a establecer lineamientos para el desarrollo de interfaces en UIML independientes de plataforma y dependientes del dispositivo cliente, basándose en una metodología fundamentada en conceptos de Interacción Humano Computadora.

Interacción con el usuario. UIML separa la descripción de la interacción con el usuario

(comportamiento) del resto de la interfaz (estructura, estilo y contenido). UIML describe el comportamiento en la ejecución como un conjunto de eventos abstractos. Para cada evento, UIML especifica qué elemento de la interfaz se asocia a cada evento, la condición necesaria para que el evento se genere y las tareas que deben realizarse cuando el evento se dispare. Al momento de la ejecución, cada evento abstracto se mapea a un evento real dependiente de la plataforma. De manera similar, en la ejecución cada tarea abstracta se mapea a un *script* o aplicación del cliente [Phanouriou 2000].

Complejidad del lenguaje. UIML permite manejo de eventos y es un lenguaje simple con un vocabulario reducido. Contiene 29 elementos organizados en una estructura de árbol (ver Apéndice B), que son sensibles a las mayúsculas/minúsculas y necesitan parámetros adicionales.

Desventajas. UIML no cuenta con etiquetas específicas a las plataformas. Captura los elementos que son comunes a cualquier interfaz de usuario a través de elementos genéricos. La sintaxis de UIML también define elementos del lenguaje que se mapean a una plataforma en particular. Sin embargo, los elementos de los lenguajes de las plataformas específicas no forma parte de UIML, pues sólo aparecen como valores de los atributos. Por lo tanto, las personas que desarrollen interfaces de usuario en UIML necesitan conocer los lenguajes de las plataformas en las que deseen desplegar sus interfaces. Esta constituye su principal desventaja, pues los programadores deben aprender, aparte de UIML, el lenguaje dependiente de la plataforma para poder describir los mapeos del estilo de la interfaz [Phanouriou 2000].

Según Mayora-Ibarra, este enfoque de construir tantas versiones de la interfaz genérica como lenguajes objetivos se requieran limita la generalidad en el diseño de interfaces de usuario. Además, otra desventaja de UIML es que actualmente sólo es compatible con JDK 1.3.x [Mayora- Ibarra et al. 2003]. En [Nichols et al. 2002] y en [Rees 2001] también se menciona esta desventaja.

2.5 Java

Si bien la mayoría de los lenguajes que se han creado para dispositivos móviles son lenguajes de marcado basados en XML, también se han creado versiones más ligeras de

Java para desarrollar aplicaciones que pueden ejecutarse en estos dispositivos. A continuación se mencionan brevemente las dos propuestas más sobresalientes.

2.5.1 J2ME de Sun Technologies

La plataforma Java 2 MicroEditon™ (J2ME), es un ambiente optimizado de ejecución de Java. La tecnología J2ME está dirigida principalmente a las aplicaciones en red que corren en dispositivos personales pequeños como localizadores, PDAs o *smart phones*. Esta plataforma incluye el ambiente de aplicaciones PersonalJava™ (PJAE, *PersonalJava™ application environment*), el cual es un ambiente de aplicaciones que ejecuta software escrito en el lenguaje de programación Java [Java Technologies 2002].

PersonalJava es el lenguaje de desarrollo de las aplicaciones y servicios que se ejecutan en el sistema operativo Symbian (Symbian OS), el cual es propiedad de Nokia, Ericsson, Motorola, Matsushita (Panasonic) y Psion; otras licencias incluyen Philips, Sanyo, Siemens, Kenwood y Sony Symbian. OS contiene una implementación desarrollada completamente en la plataforma J2ME, la cual incluye todas las características obligatorias y la mayoría de las opcionales definidas en la especificación del PJAE [Allin 2001].

2.5.2 Jeode™ de Insignia

La edición PDA de Jeode™ es una implementación de Insignia, una compañía líder en el desarrollo de máquinas virtuales de Java (JMV, Java Virtual Machine). La JVM de Jeode fue creada para usarse en Pocket PCs, PDAs y demás dispositivos manuales que cuentan con recursos de memoria y requerimientos de navegación limitados. Jeode es completamente compatible con la especificación de PersonalJava y soporta todas las clases de PersonalJava 2.1, incluso las opcionales. Además, la JMV de Jeode puede ejecutar aplicaciones de Java en un tiempo entre seis y diez veces menor que otras JVMs con casi la misma cantidad de memoria [Insignia 2003].

Estos ambientes de programación y ejecución de Java presentan las siguientes ventajas y

desventajas:

Ventajas [Allin 2001]:

- proporciona un ambiente de ejecución seguro y robusto
- estandarización
- rápido desarrollo de aplicaciones
- recolección automática de basura

Desventajas [Allin 2001]:

- Las aplicaciones de Java pueden ser muy lentas y no tienen acceso completo a todos los recursos de una plataforma.
- Hasta el momento la especificación de PersonalJava incluye todas las librerías awt, pero no las librerías swing.

2.6 Lineamientos generales de diseño de interfaces para dispositivos móviles

2.6.1 Usabilidad

El grado de usabilidad de un sistema indica hasta qué punto permite a sus usuarios completar de una manera razonablemente fácil las tareas que pretenden realizar [Arehart 2000].

Para determinar la usabilidad de una aplicación móvil se deben considerar principalmente dos aspectos [Arehart 2000]:

- Las habilidades intelectuales y tiempo requeridos para aprender a usarla
- El nivel de frustración que experimenta el usuario al utilizarla

Asimismo es importante tomar en cuenta que los usuarios de este tipo de dispositivos no están sentados inmóviles frente a su aparato, como ocurre con las PCs. Generalmente se encuentran en movimiento, caminando, en su automóvil o rodeados de mucha gente. Por lo tanto, las interfaces para estos dispositivos deben ser lo más intuitivas posibles y presentar la mayoría de las opciones a lo más a un *click* de distancia.

A continuación se presentan una serie de recomendaciones para diseñar aplicaciones móviles con un alto grado de usabilidad [Arehart 2000]:

- *Identificar las acciones más importantes*: cuando se migra una aplicación Web a

una aplicación móvil, sólo se deben incluir las actividades que la mayoría los usuarios realmente usarán en este tipo de dispositivos. Además, la aplicación debe efectuar estas operaciones lo más rápido posible.

- *Diseñar un árbol jerárquico*: una vez identificadas las actividades que la aplicación permitirá realizar, debe construirse su árbol jerárquico.
- *Minimizar la entrada de los datos*: debe reducirse al mínimo la cantidad de datos que el usuario deba proporcionar, y sólo solicitar datos alfabéticos cuando sea absolutamente necesario.
- *Personalizar*: la aplicación debe recordar información previamente proporcionada por el usuario.
- *Desplegar sólo mensajes concisos*: el texto debe ser directo, informativo y corto.
- *Incluir en todas las páginas las opciones "atrás", "inicio" y "salir":* en todas las interfaces debe ser posible regresar a la pantalla anterior, a la inicial o bien salir del sistema.
- *Evitar redundancia*: si bien en las aplicaciones Web es muy común encontrar varias formas de realizar una misma actividad (por ejemplo, un mensaje puede eliminarse al dar *click* en el icono del *trash* o bien seleccionando la opción *eliminar* del menú), en las aplicaciones móviles esto sólo agrega complejidad al desarrollo y puede ocasionar confusión al usuario.
- *Probar inmediatamente el prototipo*: deben construirse prototipos lo antes posible para que la usabilidad de la aplicación sea evaluada de inmediato.

Una aplicación móvil tiene un alto grado de usabilidad si cuenta con las siguientes características [Arehart 2000]:

- *Intuitiva*. Cuando un usuario pierde tiempo tratando de usar una herramienta, pierde la concentración y experimenta frustración. Las interfaces intuitivas son más fáciles de vender y animan a los usuarios a usarlas una y otra vez. Actualmente ya nadie está dispuesto a leer un manual antes de usar una aplicación; todos esperan entender su funcionamiento fácil y rápidamente.

- *Eficiente*. La mayoría de los usuarios móviles son impacientes, por lo que se debe reducir al máximo la cantidad de datos que se piden al usuario.
- *Fácil de recordar*. Una vez que un usuario ha entendido cómo usar la aplicación, le debe ser fácil recordarlo la siguiente vez que la use.
- *Tolerante.* Las aplicaciones deben desplegar mensajes de confirmación cuando el usuario esté a punto de efectuar una acción destructiva como cancelar o borrar.
- *Consistente*. A menudo los usuarios deben ejecutar la misma operación en diferentes etapas de la aplicación. La consistencia en la ejecución de estas operaciones es vital para que el usuario se sienta a gusto y no se confunda. Además, facilita el trabajo del desarrollador.

2.6.2 Lineamientos de diseño con respecto a los mecanismos de entrada/salida

La diversidad de características de *hardware* y *software* que presentan los dispositivos móviles complica el diseño de interfaces amigables. Asimismo, presentan algunas desventajas con respecto a las computadoras tradicionales, pues tienen un ancho de banda muy pequeño; su capacidad de almacenamiento es muy limitada; generalmente tienen pantallas pequeñas en donde sólo pueden desplegarse pocas líneas de texto; y algunos utilizan reconocimiento de voz, no el teclado tradicional, como mecanismo de entrada/salida. Debido a estas diferencias, las interfaces diseñadas para ejecutarse en computadoras tradicionales no funcionan correctamente en estos dispositivos.

[Aragón 2002] presenta un estudio detallado sobre las características de los modelos más populares de dispositivos móviles. También propone una serie de recomendaciones para el diseño de interfaces móviles, las cuales se mencionan a continuación:

- *Reducir al mínimo los datos de entrada alfabéticos*: si bien la mayoría de las PDAs cuentan con mecanismos como *touch screen* o *point and click*, los celulares sólo disponen de teclados *multi-tab*, es decir, teclados en donde hay más de un caracter por botón. Esto complica la captura de texto.
- *Pedir datos numéricos como entrada siempre que sea posible:* los números son la

primera opción en los teclado *multi-tab*.

- *Aprovechar al máximo las teclas de selección, listboxes, radio-buttons y link-lists:* de esta manera la selección pueden realizarse fácilmente mediante teclas de *scroll*.
- *Evitar el scroll horizontal:* no está resuelto en la mayoría de los dispositivos; en cambio, el *scroll* vertical sí.
- *Usar abreviaciones comunes en los mensajes*
- *Dividir la información en módulos que puedan visualizarse por completo en una pantalla:* de esta manera se evita el *scroll.*
- *Filtrar la información que se presenta en base a las necesidades del usuario:* así se reduce la cantidad de texto que se envía y se despliega en el dispositivo. No se debe olvidar que estos dispositivos disponen de un ancho de banda muy pequeño comparado con el de Internet.
- *Considerar el mínimo común denominador capacidades de pantalla*: es decir, cuatro renglones de 15 a 18 caracteres. De esta manera, las aplicaciones también se desplegarán bien aún en pantallas más grandes, como PDAs [Arehart 2000].

2.6.3 El enfoque genérico

Si bien la usabilidad es uno de los grandes retos del desarrollo de aplicaciones para dispositivos móviles, las diferencias entre los dispositivos móviles generan otro gran problema: el navegador de cada dispositivo despliega la misma interfaz de manera distinta. La principal interrogante es, ¿cúal es el mejor enfoque para desarrollar aplicaciones que deban desplegarse en múltiples micronavegadores?

Por supuesto, una solución consiste en desarrollar múltiples versiones de una aplicación para cada uno de los navegadores: se le conoce como el enfoque de *fuerza bruta*. Otra solución consiste en escribir código genérico que funcione bien en los diferentes dispositivos.

La idea básica detrás del enfoque genérico consiste en identificar los lenguajes que son interpretados por la mayoría de los navegadores de los dispositivos móviles, así como

definir el mínimo común divisor de las características de los micronavegadores [Arehart 2000].

En los celulares, de manera general pueden identificarse dos tipos de micronavegadores [Arehart 2000]:

- *UP.Browser de Phone.com*: es utilizado por Motorola, Siemens, Alcatel, Philips y Samsung, entre otros; es el navegador más popular en el mercado estadounidense.
- *El browser de Nokia*: es el más popular en Europa.

En las PDAs, los navegadores que predominan son [Arar 2003]:

- *Pocket Internet Explorer:* es el navegador de Microsoft y lo utilizan las iPAQs y Jornadas de Hp-Compaq, las Dell Axim, las series e800 de Toshiba, las PocketPCs de ViewSonic y todas las PDAs que funcionen con el sistema operativo Windows Mobile (antes Windows CE).
- *PalmSource™ Web Browser:* es usado por todas las Palm y Sony Clie.

Una vez escrito, el código genérico debe convertirse a los lenguajes que interpretan los micronavegadores de los diferentes dispositivos. Esta conversión puede realizarse con las tecnologías XSL mencionadas en la Sección 2.3.4 [Arehart 2000].

El capítulo siguiente presenta el trabajo relacionado con la construcción de interfaces genéricas.