UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

SISTEMA DE EDICIÓN Y EMULACIÓN
DE DOCUMENTOS WML

T E S I S

QUE PARA OBTENER EL TÍTULO DE:
INGENIERO EN COMPUTACIÓN
P R E S E N T A N:

GONZÁLEZ ORTÍZ OSCAR ALBERTO
SANTILLÁN HERNÁNDEZ RICARDO

DIRECTORA DE TESIS: ING. LAURA SANDOVAL MONTAÑO

MEXICO, D.F. 2005

AGRADECIMIENTOS

Generales

A nuestra querida Universidad

Por ser la mejor institución educativa del mundo, aportando a la sociedad profesionistas que cuentan con una educación con calidad excepcional.

A la Facultad de Ingeniería y maestros.

Por dar a los futuros Ingenieros del país la mejor preparación acorde a la vanguardia tecnológica, para enfrentar los retos que México demande.

A nuestra directora de Tesis Laura Sandoval Montaño por su tiempo, paciencia y dedicación.

Oscar Alberto

A mis padres:

Gracias por brindarme su apoyo incondicional, su vida para acompañarme en todo momento, su oído para escucharme, su mano para levantarme ante las adversidades, su sabiduría para compartirla conmigo con cada consejo que me han dado.

Ante todo quisiera agradecerles la mejor herencia que me pudieron haber dado: mi educación.

Gracias por existir.

A mis hermanas:

Por acompañarme a lo largo de mi vida brindándome su amistad, cariño y comprensión. A Cintya por reirse conmigo de la vida y a Leslie por aguantarme las bromas.

A la familia González:

Quiero agradecer a toda la familia González por el apoyo, cariño y comprensión brindado a lo largo de mi vida, sobre todo gracias a mis abuelitos que donde quiera que se encuentren se pueden sentir orgullosos de su nieto.

A la familia Ortíz:

Por estar conmigo en todo momento, por creer en mí y sobre todo por todo el cariño que siempre me han demostrado. A mi abuelito Eduardo, que me enseñó tantas cosas, donde quiera que estés muchas gracias. A mi abuelita Julia que siempre ha estado conmigo apoyándome y consintiéndome.

A la familia Toledo Ortíz:

Quiero agradecer a mis primos Beldar Rosbel y Rosger que hayan crecido conmigo aprendiendo, jugando y divirtiéndonos. A mi tía Rosi por sus sabios consejos y dedicación. Gracias por ser mi segunda familia.

Ricardo

A mis padres:

Por todo el apoyo incondicional y constante que he recibido siempre, así como por todo el amor que día a día me demuestran. Gracias por todos esos años de desvelos y de esfuerzo que no ceja.
Papá, gracias por tus consejos y esas pláticas de toda la vida.
Mamá, gracias por ese ejemplo de lucha y de una vida de esfuerzo constante, pero sobre todo por demostrarme que las adversidades son para vencerse.

A mi hermano:

Fer, por ser mi mejor amigo y aceptar la responsabilidad que esto conlleva: apoyo confianza, cariño, verdad y entereza. Gracias por haber preferido ser mi amigo antes que ser mi hermano.

A mis abuelitos:

Por ser la luz que guío mi camino, pero sobre todo, quiero agradecer a Doña Refugio, por plantar los cimientos sobre los cuales edifiqué carácter, gracias mi negrita.

Agradecimientos especiales:

Alejo, gracias por ser parte de nuestra familia y permitirnos verte como a un hermano. Gracias por tus bromas, tus comentarios y sobre todo por tu cariño.

Jennifer, no tengo palabras para agradecerte por todo el peso que me ayudaste a cargar, así como ese apoyo y motivación que siempre sentí, tu fe me ayudó a salir adelante.

Tío Fernando, gracias por tus consejos y tu ejemplo de vida, pero sobre todo gracias por tu forma de ser. A Dany y Marilú les agradezco el cariño y las porras.

Tío Alejandro, por ese amor que expresas a mi familia.

A la familia Flores Villanueva por ese cariño que siempre nos ha expresado.

A las familias Hernández García, Hernández Villanueva, Hernández Zuñiga, Hernández Pineda, Orozco Moncada, Santillán, Parra Miranda y en general, a todas aquellas personas de las cuales he recibido apoyo, ánimos y comprensión. Gracias, a todos y cada uno de ustedes.

Índice General

INTRODUCCIÓN.

El objetivo del presente proyecto de tesis es el de crear un sistema para el desarrollo sencillo, rápido y amigable de páginas que utilizan el Lenguaje de Marcación Inalámbrica (WML).

Este sistema ofrecerá a los desarrolladores un entorno para la creación, prueba y ejecución de las aplicaciones WML; una de las ventajas de esta herramienta sobre algunos productos del mercado será la posibilidad de no tener que salir de la aplicación ya que se dispondrán de todos los elementos necesarios para empezar a trabajar, y esto permite al desarrollador no preocuparse por las herramientas debido a que todas las tiene a su alcance, y centrarse en el desarrollo que es, al fin y al cabo, lo más importante.

Para poder realizar dicha herramienta se efectuó un análisis de las herramientas que existen en la actualidad para el desarrollo de páginas WML. A partir de lo observado se diseñó un sistema que resultara amigable para el usuario en el cual no necesitara de leer grandes manuales de usuario para empezar a utilizarlo, ni de muchos conocimientos de programación en el lenguaje WML.

Para lograr nuestro objetivo nos apoyamos en el lenguaje de modelado unificado (UML, por sus siglas en inglés) para el análisis y diseño, y tecnologías basadas en Java para la implementación.

De esta forma la tesis está estructurada en 8 capítulos:

En el capítulo I, *Difusión de la información en dispositivos inalámbricos*, se da una breve descripción de la importancia de la información, del uso de los dispositivos inalámbricos para su transmisión y de las características de los dispositivos móviles e Internet inalámbrico.

En el capítulo II, *Protocolo de Aplicaciones Inalámbricas (WAP)*, se describe el protocolo que se emplea para la comunicación entre dispositivos inalámbricos. Se explica a detalle cada una de las capas que conforman el protocolo y su relación con los equipos móviles.

En el capítulo III, *Lenguaje de Marcación Inalámbrica (WML)*, se explican los elementos básicos del lenguaje de marcación inalámbrica (WML), que se usa en la generación de páginas para equipos inalámbricos como teléfonos celulares y PDA's, por medio de ejemplos y resaltando algunos de los aspectos más importantes.

En el capítulo IV, *Problemática en la edición de documentos WML,* se define la funcionalidad que necesitan los usuarios y la forma útil de proporcionarla, en otras palabras, en este capítulo se abordan los problemas en la edición y emulación de páginas WML, y se ofrece una propuesta para la solución.

En el capítulo V, *Análisis del Sistema de Edición y Emulación de Documentos WML*, se aborda la propuesta de solución por medio del "proceso de desarrollo unificado" y se describen algunos de los aspectos más importantes de las herramientas que se utilizarán para la propuesta de solución.

En el capítulo VI, *Documentación del Análisis y Diseño del Sistema*, se expresa el diseño de la solución por medio del Lenguaje de Modelado Unificado y se presentan también los diagramas propios del análisis. Con este diseño como base para la solución del problema, el siguiente paso es la implementación, la cual se describe en el capítulo VII.

En el capítulo VII, *Desarrollo del Sistema*, se describen las herramientas utilizadas para la implementación del sistema. El desarrollo está basado en herramientas como Java y Swing.

Finalmente en el capítulo VIII, *Pruebas y evaluación*, se explican las pruebas realizadas a la funcionalidad del sistema, basadas en un modelo de pruebas.

CAPÍTULO I.
Difusión de la información en dispositivos inalámbricos

I.1 La importancia de la información y su difusión.

Las palabras "comunicación" e "información" pertenecen al lenguaje cotidiano; se usan y se conoce su significado en forma intuitiva, nadie subestima su importancia, pero pocas personas podrían definirlas en forma precisa.

Desde el punto de vista etimológico, la palabra "comunicación" proviene de la raíz latina *communicare*, es decir, "hacer común" algo. Por otra parte, "información" tiene su origen en las palabras *in* y *formare*, es decir, "instruir hacia adentro". A partir de estas dos palabras, y debido a la importancia que en épocas recientes han cobrado, se ha generado una enorme cantidad de variantes, cada una con un significado muy preciso, aplicable a determinadas situaciones. Por ejemplo, "telecomunicaciones" significa comunicar a distancia, "informática" (que proviene de "información", auto y mática) supone el procesamiento automático de la información; "telemática" es la conjunción de "telecomunicaciones" e "informática", e implica la transmisión y el procesamiento automático de la información.

La información y comunicación tienen una gran cantidad de acepciones, y sus significados pueden ser sorprendentemente distintos, como veremos a continuación.

La información es coleccionable, almacenable o reproducible. Se utiliza para tomar decisiones, conduce también a conclusiones acertadas o equivocadas, puesto que puede ser interpretada de diversas formas por distintos individuos, dependiendo de muchos factores subjetivos y del contexto en que se encuentre la persona que la recibe e interpreta. Así como es posible comunicar una noticia, también se comunican los estados de ánimo, opiniones o conocimientos.

Todo lo relacionado con las comunicaciones, es decir, las técnicas, la ciencia, la tecnología, se ha visto fuertemente impulsado por las necesidades militares de cada época. Una infinidad de hechos históricos documentan el derrumbe de personajes, la derrota de ejércitos y la pérdida de enormes fortunas, porque alguna de las partes en pugna contaba con información estratégica que las otras partes no poseían.

La mayor influencia sobre las comunicaciones la tuvo la Segunda Guerra Mundial: en esa época la humanidad ya se encontraba en la frontera de la revolución tecnológica, misma que las actuales generaciones hemos tenido la oportunidad de presenciar desde hace algunos años. Muchos de los sucesos que condujeron a la conclusión de la guerra, con el resultado que todos conocemos, estuvieron relacionados con la disponibilidad de información oportuna o con la intercepción ingeniosa de información del enemigo. Los requerimientos de comunicaciones instantáneas, seguras y privadas de esa época fueron determinantes para que las comunicaciones sean lo que son hoy en día.

Se sabe de muchos escándalos financieros en los cuales las personas que poseen información confidencial antes que otras, la usan a su favor, y ganan grandes capitales (este uso personal de información confidencial es ilegal en muchos países).

En estos días es difícil pensar que alguien niegue conscientemente que la información tiene un valor; la información ha ido ganando importancia conforme la gente que toma decisiones está convencida de que ésta se puede asociar a un valor real, frecuentemente ligado a un valor material o económico. Esto es distinto de lo que ocurría en otras épocas, en que predominaban otros bienes y servicios, que tenían mayor valor económico. A las épocas de grandes cambios en la historia de la humanidad, se les han asignado nombres especiales: el Renacimiento, la Ilustración, Revolución Industrial... En nuestros días, es de tal importancia poseer, administrar y transmitir información, que toda la humanidad se ve y se seguirá viendo afectada, influida y posiblemente dominada por quienes tienen, administran y transmiten este recurso, razón por la cual a esta época se le han impuesto los calificativos de "sociedad de la información" o de "revolución electrónica", este último debido a la facilidad con que se transmite la información por medio de los sistemas modernos basados en dispositivos electrónicos.

Uno de los aspectos más abstractos e importantes de la información es que su valor puede disminuir a lo largo del tiempo. Es decir, en un momento determinado a alguien le puede interesar contar con cierta información, pero ese interés puede decrecer o incluso desaparecer algún tiempo después. Por otra parte, es necesario que la información sea de interés para el individuo que la adquiere o recibe, quien, además, no debe conocer *a priori* su contenido; en caso contrario, dicha información le resultará irrelevante. Es evidente que este estado de incertidumbre no necesariamente tiene que ser consciente ni voluntario.

La información se origina en una fuente y se hace llegar a su destinatario por medio de un mensaje a través de un canal de comunicación; el destinatario generalmente se encuentra en un punto geográfico distante, o por lo menos, separado de la fuente. La distancia entre fuente y destinatario puede variar desde pocos centímetros (al hablar frente a frente a un volumen normal) hasta cientos y aun miles de kilómetros (como es el caso de transmisiones telefónicas intercontinentales o de transmisiones desde y hacia naves espaciales).

Esto constituye precisamente el problema central de las telecomunicaciones, ya que al haber una fuente que genera información en un punto y un destinatario en otro punto geográfico distante del primero, se trata de saber cuál es la mejor manera de hacer llegar al destinatario la información generada por la fuente, de manera rápida (por la dependencia temporal de la importancia de la información), segura (para garantizar que la información no caiga en manos de alguien que haga mal uso de ella, o a quien simplemente no estaba destinada), y veraz (para garantizar que en el proceso de transmisión no se alteró el contenido de la información). En nuestros días, influidos fuertemente por aspectos de tipo económico, intervienen además otros factores, tales como el costo de hacer llegar la información de la fuente a su destino. Si el factor de costos no fuera determinante, con seguridad conversaríamos telefónicamente con amistades o parientes en otros países sin importar la duración de las llamadas.

El problema central de las telecomunicaciones fue definido con claridad por Shannon [Shannon, 1949], con una sencillez asombrosa, quien estableció que un sistema de comunicaciones consiste en cinco componentes:

1) Una fuente de información
2) Un transmisor de información cuya función consiste en depositar la información proveniente de la fuente en un canal de comunicaciones
3) Un canal de comunicaciones, a través del cual se hace llegar la información de la fuente al destino
4) Un receptor que realiza las funciones inversas del transmisor, es decir, extrae la información del canal y la entrega al destinatario
5) Un destinatario.

Se puede entender mejor en la figura I.1

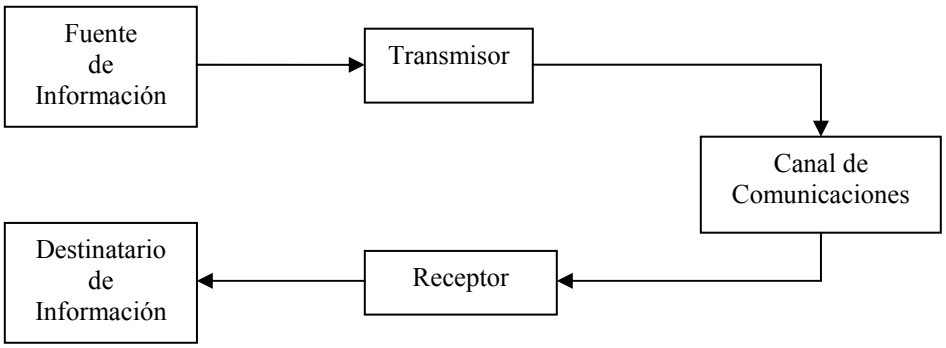


Figura I.1 *Componentes de un Sistema de Comunicaciones*

Un mensaje se usa para hacer llegar información de fuente a destino, y no es lo mismo un mensaje a la información que éste contiene. Considérese el siguiente ejemplo: Una persona (A) desea enviar cierta cantidad de dinero por medio de un giro telegráfico a otra persona (B). En este caso, A es la fuente, B el destinatario. La información es aquello necesario para conocer la cantidad de dinero y para originar la entrega del mismo a B, y el mensaje es el conjunto de palabras o símbolos telegráficos necesarios para que B conozca la intención de A y para que B pueda disponer del dinero que A le envía.

Desde los orígenes de la humanidad, la forma natural en que la información se transmite entre personas es a través del lenguaje oral. (En la actualidad, también existe la necesidad de transmitir información entre máquinas). Debido a la naturaleza efímera de los mensajes orales (hay que recordar el dicho popular de que "las palabras se las lleva el viento"), siempre existió el deseo y la necesidad de que la información no varíe en el transcurso del tiempo. Ello dio origen a los mensajes escritos, los cuales han evolucionado desde las pinturas rupestres, la escritura cuneiforme, los pictogramas, los jeroglíficos y el lenguaje fonético de los fenicios en el siglo XI. a. de C., hasta los distintos conjuntos de símbolos con que hoy se cuenta. Los precursores de las memorias electrónicas, magnéticas u ópticas de la actualidad son precisamente el papel y los muros de las cavernas. A lo largo del proceso, para pasar de los mensajes escritos a los

símbolos codificados, el hombre inventó y perfeccionó sistemas que son frecuentemente utilizados en la actualidad, tales como la imprenta y la fotografía.

Desde la Antigüedad se reconocía la necesidad de transmitir información a distancia. Desde entonces, las soluciones a este problema han estado íntimamente relacionadas con el desarrollo cultural, social y político de la humanidad. Para transmitir información entre dos puntos, primero debe ser "envasada" en un “contenedor", que posteriormente se enviará a través de un canal; dicho proceso es tan abstracto como el de la misma información, pero se explica con la ayuda de algunos ejemplos: si la información consiste en ideas, decisiones o estados de ánimo, las maneras de enviarla a distancia por medio de palabras, texto impreso, imágenes, ondas acústicas, ondas electromagnéticas o señales intermitentes de humo, por mencionar sólo algunas, y los canales de comunicación para cada uno de ellos son respectivamente el aire, el correo, un cable de televisión y la atmósfera; en todos los casos se observa que el medio o canal a través del cual se transmite la información es un elemento que impone restricciones sobre los "contenedores" de la información: una onda acústica sólo puede ser transmitida por un canal que conduzca ondas acústicas y una eléctrica, por medio de un conductor de señales eléctricas. Afortunadamente, hoy en día, con ayuda de la tecnología, es posible solucionar estas limitaciones y convertir señales de un tipo a otro: el precursor de esto es el micrófono, por medio del cual se convierte una señal acústica en eléctrica.

El mensaje fue creado por el hombre para comunicarse, es decir, para hacer común algo que en este caso específico es la información. Esto es una muestra palpable del ingenio humano: la creación de un mensaje forzosamente implica la necesidad de codificar la información para que sea susceptible de ser enviada o transmitida; no sería posible transmitir una idea sino se utilizara el lenguaje oral, el corporal, el escrito, o algún otro; estos lenguajes son precisamente las versiones codificadas de la información. Es posible explicar las funciones del codificador de la siguiente manera: así como no se puede enviar una carta (es decir, un sobre de papel que contiene otros papeles en su interior, cuyos símbolos o texto contienen la información que se desea transmitir) a través de un canal telefónico o de la atmósfera, tampoco es posible enviar señales de humo utilizando para ello un sobre de papel. Por tanto, es indispensable adaptar el mensaje que contiene la información al canal por el que será transmitido. Ésta es precisamente la función de un codificador. Para que se complete el proceso de comunicación, se requiere que tanto el que origina el mensaje como el que lo recibe conozcan la forma en que fue codificada la información (esto es, el código que fue empleado); en otras palabras, para que dos personas se comuniquen por la vía oral, es indispensable que ambas hablen el mismo idioma, y para que dos personas se comuniquen por vía telefónica, se requiere que, además de hablar el mismo idioma, ambas tengan a su disposición un aparato telefónico y que ambos estén unidos por medio de conductores de señales (véase figura I.2).

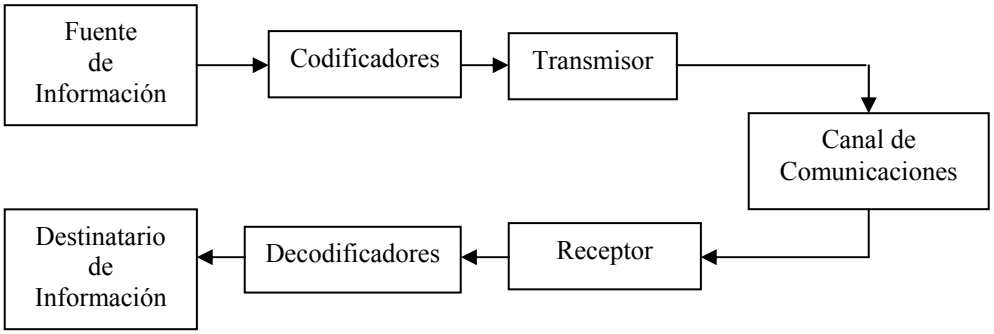


Figura I.2 *Sistema de Comunicaciones con Codificadores.*

El hombre, al querer cubrir distancias cada vez mayores, empezó a utilizar sistemas cada vez más complejos, conforme se lo permitían los avances científicos y tecnológicos. Como consecuencia, también comenzó a usar sistemas de codificación tan abstractos como la escritura misma: símbolos basados en señales intermitentes de humo, o en diversas combinaciones de señales de fuego generadas por medio de antorchas. Éstos fueron los precursores de la codificación de la información. El historiador griego Polibio (204-122 a. de C.) [Oberliesen, 1982] relata que la manera en que se codificaban las 24 letras del alfabeto griego era colocando cada una de ellas en una retícula cuadrada de 5 x 5 unidades: por ejemplo, el código de la letra "alfa", colocada en el primer espacio, era "primer renglón, primera columna". Se puede afirmar que también fue Polibio quien diseñó el primer sistema digital de comunicaciones sincronizadas. En este caso, se trabajaba en la misma línea visual, de una isla a otra, con dos recipientes cilíndricos de igual tamaño llenos de agua. Ambos tenían un pequeño orificio por donde salía un chorro de agua. Dentro de los recipientes se contaba con una regla que tenía un conjunto de símbolos convencionales: "necesito refuerzos", "necesito alimento", "manden barcos", etc. Por medio de una antorcha se señalizaba (se informaba) de una isla a otra el instante en que debía ser abierto el orificio, y por medio de otra antorcha se señalizaba el instante en que debía ser cerrado. El mensaje transmitido era precisamente aquel que se encontraba a la altura del agua en el momento de cerrar los orificios. Por supuesto que la sincronía era un factor extremadamente crítico; si ésta fallaba podían recibir, por ejemplo, refuerzos de caballería cuando lo que en realidad necesitaban eran alimentos.

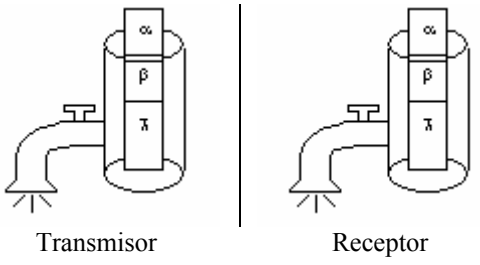


Figura I.3 *Codificador de Polibio*

La documentación que existe acerca del desarrollo de la transmisión de información, es decir, de las telecomunicaciones, principalmente en sus orígenes, es más escasa que aquella referente a la evolución de los lenguajes escritos; de hecho, la primera existe gracias a los segundos.

Es probable que entre los primeros sistemas de los cuales se valió el hombre para transmitir información a distancia fuera el de los mensajeros humanos. Sin embargo, cuando la distancia era mayor que la que podía recorrer un mensajero en el tiempo requerido para que el destinatario no perdiera interés en la información o para que ésta no llegara demasiado tarde, surgió el sistema denominado de "relevos". Esta nueva evidencia del ingenio humano está documentada en fuentes históricas sobre las comunicaciones en el Imperio romano: ahí se menciona la existencia de "mutaciones" y "mansiones", o sea estaciones para cambio de cabalgadura y para descansar, respectivamente.

Un mensajero que en aquella época tenía que recorrer largas distancias estaba también forzado a salvar todas las asperezas topográficas propias de la región. El hombre se percató, entonces, del hecho de que las señales ópticas podían recorrer mayores distancias y más rápidamente que las señales de tipo acústico. "Esta nueva tecnología" podía, además fácilmente salvar obstáculos, como barrancos, cerros, ríos o lagos.

Describir la forma en que las comunicaciones han evolucionado desde aquellas épocas hasta nuestros días sería equivalente a hacer un relato histórico de la humanidad misma. Así, pues, partiendo del hecho de que siempre ha existido la necesidad o simplemente el deseo de transmitir información a distancia en forma rápida y confiable, a continuación exponemos la evolución de esta rama del conocimiento, explicando también el funcionamiento de algunos sistemas usados en la actualidad.

A pesar de que en el pasado remoto fueron ideados los precursores de las telecomunicaciones modernas, cuando al principio pequeños grupos de individuos y después países enteros construyeron sus propias infraestructuras para satisfacer la necesidad de transmitir información a distancia, los fundamentos técnicos del área datan del pasado y del presente siglo.

Fue necesario el descubrimiento de muchos fenómenos elementales de la física, tales como la electricidad y el magnetismo, para modificar los sistemas de comunicación. Esto ocurrió gracias a los trabajos pioneros de los físicos [Bunch, 1993] A. Volta (1745-1827); G. S. Ohm (1787-1854); J. C. Maxwell (1831-1879); A. M. Ampere (1775-1836); J. Henry (1797-1878); M. Faraday (1791-1867); H. C. Oersted (1777-1851); C. Wheatstone (1802-1875); K. F. Gauss (1777-1855). Sus trabajos fueron aplicados exitosamente a los primeros sistemas de telecomunicaciones por H. Baudot (1845-1903); C. Chappe (1767-1805, quien fue el primero en utilizar la palabra "telégrafo" para identificar el sistema que usaba para enviar mensajes a su hermano mientras estaba en la escuela); S. Morse (1791-1872); G. Marconi (1874-1937); A. G. Bell (1847-1922) y H. Hertz (1857-1894). Después del telégrafo óptico se originó el telégrafo eléctrico en su versión alámbrica (1844), la cual evolucionó a su versión inalámbrica (1874). Posteriormente, en 1876, surgió el sistema telefónico.

El teléfono se convirtió en el sistema predominante, debido a que existía un abismo enorme en sus características con respecto al sistema postal e incluso al telegráfico, gracias a su velocidad, su confiabilidad, su bidireccionalidad y su privacía.

Durante estos 125 años de vida del teléfono, se han desarrollado de manera paralela otras tecnologías complementarias y, por añadidura, acontecimientos muy importantes promovieron aún más el acelerado desarrollo del área. Ya desde la Primera Guerra Mundial se reconoció plenamente el valor estratégico de la transmisión de la información, y se le dieron además algunos elementos que hasta esos momentos no habían sido considerados importantes: no era suficiente que llegara la información a su destino, sino que debía llegar de manera confiable y segura, sin la posibilidad de ser interceptada o escuchada por otras personas, a pesar de la presencia inevitable del ruido en los canales de comunicaciones.

En estas fechas estamos presenciando una carrera tecnológica en la cual es frecuente ver nuevos sistemas y servicios que hasta hace unos años eran inimaginables. Ello ha sido generado por un sinnúmero de descubrimientos científicos y tecnológicos sobresalientes dentro de las comunicaciones, que han dado forma a lo que hoy son las telecomunicaciones modernas.

Los servicios y sistemas basados en tecnologías modernas que actualmente tiene a su disposición la humanidad cubren una amplia gama que va desde la telefonía hasta la transmisión de datos por medio de redes donde las computadoras establecen "diálogos" entre sí, pasando por todos los sistemas de comunicación con que gran parte del mundo se enfrenta todos los días, tales como todas las modalidades de radiodifusión, entre las que se encuentran la radio y la televisión, así como todas las variantes de la telefonía.

I.2 Los dispositivos inalámbricos

I.2.1 Introducción

La tecnología inalámbrica está influyendo ya en nuestras vidas y lo seguirá haciendo hasta el punto en que no podremos imaginar cómo hemos podido vivir sin ella.

Las capacidades que ofrece la tecnología inalámbrica proporcionan mayor comodidad y movilidad con total funcionalidad en cualquier lugar. Pero para que tenga aceptación entre los usuarios, esta funcionalidad debe garantizarse cualquiera que sea la plataforma o la marca que adquieran. Por lo tanto, los posibles competidores en este mercado se están poniendo de acuerdo para establecer estándares que aseguren a los usuarios finales la compatibilidad y/o el funcionamiento conjunto de sus distintos productos.

Aunque no nos hayamos dado cuenta, la tecnología inalámbrica ha estado a nuestro alrededor durante mucho tiempo. Las ondas de radio, infrarrojos, microondas y ondas de sonido influyen en nuestro mundo de muchas maneras distintas – y ninguna necesita hilos ni cables. Ahora, la tecnología inalámbrica ha dado un paso más, proporcionando conexiones de datos entre dispositivos informáticos y redes, y entre diversos dispositivos informáticos.

I.2.2 Internet Inalámbrico

Estos últimos años se han presentado como una suerte de Revolución Tecnológica y el mercado, siempre vanguardista, no está dispuesto a dejar pasar una oportunidad tan prometedora como puede ser abarcar el tópico de las comunicaciones.

Hoy hace sus apuestas a Internet inalámbrico, un nuevo sistema que intenta facilitar el quehacer de los hombres, tanto el aspecto laboral como social, adecuando los servicios de Internet con los que prestan los celulares. Es decir, que a través de la tecnología WAP (Wireless Application Protocol), se amplía la posibilidad de transferencia de datos.

El lanzamiento de Internet inalámbrico habilita también una serie de negocios colaterales, por lo que el aspecto a tener en cuenta se intensifica, al punto que puede tornarse ilimitado en posibilidades, puesto que las ventajas que otorga entre otras son:

- Permite consultar portales bajo este tipo de estándar y con formato de auto texto.
- Está habilitado para el uso de Intranet y es posible personalizar el servicio a las necesidades de cada usuario.
- Principalmente, el negocio se va a centrar en la posibilidad de realizar transacciones a través del celular.

La mirada del mercado está dirigida en dos ámbitos dispares, los jóvenes y los profesionales, en principio. Ya que Internet, omnipresente por naturaleza y la telefonía celular, forman parte de la cultura moderna.

En lo que hace a la demanda creciente en la movilidad de los profesionales y la necesidad de herramientas prácticas que mejoren la productividad conduce a la utilización de notebooks, palmtops, etc. conectadas al teléfono móvil, liberando a las personas de enchufes y edificios. Además, el tamaño y peso de los ordenadores portátiles los hacen idóneos para comunicar mensajes o información desde cualquier lugar con otros usuarios, con dispositivos móviles o no. Es por eso que ya llegan los teléfonos desde los cuáles se puede navegar por la Web.

No obstante, deben diferenciarse tres conceptos al considerar la movilidad del usuario:

- Conexión sin atadura: se refiere a la conectividad sin necesidad de cable físico.
- Acceso nómada: hace la flexibilidad geográfica en acceder a un servicio de comunicaciones.
- Acceso móvil: que alude a la conectividad cuando el usuario está en movimiento.

Los tres conceptos están interrelacionados. Los usuarios móviles son, por definición, nómadas y sin atadura física, pero los usuarios de acceso nómada, no son necesariamente sin cordón. Ejemplos de este sistema sin cordón son los teléfonos inalámbricos.

La movilidad es una característica deseable, altamente valorada por los usuarios porque estar limitado a una localización fija no concuerda con la esencia dinámica de muchas actividades humanas, aún dentro del lugar de trabajo o en hogar.

La experiencia demuestra que los usuarios prefieren elegir los servicios y aplicaciones móviles, cuando pueden optar.

La telemática móvil tiene un gran futuro, aunque también debe afrontar retos importantes. Si bien el sistema de Internet inalámbrica es técnicamente progresista, hay un peligro inminente, y es la natural falta de cultura en la utilización de Internet por parte de los usuarios de celulares. Hace falta educar al cliente para que esta clase de servicios tenga éxito. Cualquier mal paso puede llegar a generar frustraciones y retrocesos en la implementación del servicio.

Por otro lado, las aplicaciones de Internet en un celular son muy diferentes, por ahora, a las que se pueden realizar desde una computadora fija y aunque ambas tecnologías se complementen, se deben tener en cuenta las necesidades del cliente.

Es muy difícil que alguien se ponga a navegar a través del celular como lo haría desde su casa o en la oficina. Una de las razones que lo obstaculizan es el costo de la maniobra.

I.2.3 Perspectiva de la tecnología inalámbrica

Las empresas modernas cuentan cada vez más con personal móvil, que ya no está encadenado a la mesa durante ocho o más horas diarias. Los empleados están equipados con ordenadores notebook y pasan más tiempo trabajando fuera de los lugares tradicionales de trabajo. Están en la calle, metidos en el tráfico, entre vuelo y vuelo, en un taxi, trabajando en una habitación de hotel o en la piscina.

El uso de Internet como un potente medio de información y comunicación ha originado una gran demanda de acceso 24 horas al día, 7 días a la semana, sin importar la ubicación. Y como la mayor parte de la productividad se da en reuniones y fuera de la mesa de trabajo, es preciso disponer de acceso flexible a una red en cualquier situación posible.

Hasta ahora no había sido fácil. Pero cuando una persona ha experimentado y se ha acostumbrado al uso del ordenador personal y los servicios de comunicaciones en la oficina o en casa, el siguiente paso es esperar y demandar capacidades similares cuando está en movimiento.

Los consumidores están demandando una transición similar para las capacidades multimedia - en sonido, datos, imágenes y vídeo - en dispositivos móviles.

Nuevas e innovadoras tecnologías permiten acceder a Internet, a una intranet corporativa o a su propia red doméstica desde cualquier lugar donde haya cobertura telefónica. Y en un futuro con dispositivos electrónicos más pequeños, baratos y potentes, la velocidad y la comodidad con la que se accederá a la información crecerán exponencialmente.

Las comunicaciones inalámbricas y de la Internet se encuentran actualmente en pleno apogeo. Existen más de 300 millones de suscriptores inalámbricos y cerca de 200

millones de usuarios de la Internet alrededor del mundo. Hacia el año 2005, se presume que cada una de estas tecnologías enlazará a mil millones de usuarios.
La confluencia de estos dos medios interactivos integrados en la Internet inalámbrica constituye el evento más trascendente en la historia de las comunicaciones. En cualquier lugar que se encuentren los usuarios, la Internet inalámbrica posibilitará el acceso a una inmensa variedad de información sin limitaciones de tiempo o ubicación. A los operadores de redes y proveedores de servicios, les ofrecerá una variedad inusitada de novedosas y atractivas oportunidades de negocios. A entidades comerciales y agencias gubernamentales, les brindará el grado de flexibilidad necesario para optimizar el manejo de las múltiples dimensiones de sus sistemas de comunicación.

I.2.4 Características salientes de acceder a Internet utilizando acceso wireless

El acceso a Internet por este medio presenta características alentadoras para los usuarios, como son:

- ***Conexión instantánea y asegurada:*** no hay que discar teléfono y no existe la posibilidad de que dé ocupado. La conexión se establece en segundos y una vez ingresados el nombre de usuario y la contraseña se puede navegar.
- ***Presencia dedicada en la Red:*** aunque el servicio tiende a desconectarse si pasan varias horas sin transmisión de datos, es posible tener presencia permanente en Internet, algo apreciado para los usuarios acostumbrados a tener su propio servidor de FTP o a intercambiar cualquier clase de material vía programas para intercambio de archivos (tipo Napster, Kazaa o Limeware)
- ***Mayor estabilidad de la conexión:*** los responsables del servicio aseguran que prácticamente no se sufren cortes de la conexión.
- ***Buena velocidad de navegación:*** aquí *wireless* se pone a mitad de camino entre un *Dial-Up* y un Cable módem. A pesar de ser claramente más rápido que un Dial-Up que es habitualmente de 56Kbps, su velocidad máxima de entre 1,5 y 3Mbps no es competencia para la del cable módem que puede llegar hasta 10Mbps.

Frente a tantos puntos en los que sale bien parado, no puede dejar de mencionarse la mayor desventaja del *wireless*: “su restringido radio de acción”.

Pegado al concepto de *wireless* está el de movilidad. Porque, claro, la libertad que otorga no tener que depender de cables hace pensar enseguida en opciones portátiles.

Justamente, la mayor de las gracias de las conexiones *wireless* es ésa. Por eso se caía de maduro que, tarde o temprano, los celulares (que nos acostumbraron a que hablar por teléfono no es una actividad que necesariamente se tenía que limitar a la sala o al dormitorio de casa) se iban a sumar.

I.2.5 Dispositivos móviles.

Los dispositivos móviles como teléfonos celulares y asistentes personales digitales (PDAs) se están conectando a la Web.

"Wireless" es el término anglosajón que, como suele suceder con este idioma, no es sólo una descripción (literalmente significa "sin alambres"), sino también una filosofía tecnológica que hace algunos años se viene imponiendo en el mundo y que empezó a pisar fuerte con el establecimiento de las primeras empresas que proveen acceso a Internet utilizando esta variante y con la popularización de las posibilidades de acceso a la Red mediante teléfonos celulares.

La base en la que se sustenta todo lo que sea *wireless* son las microondas. Utilizando la misma tecnología que hace posible la telefonía celular, es posible ampliar el rango de prestación de servicios de casi cualquier objeto doméstico.

I.2.6 Aplicaciones del acceso inalámbrico

Lo importante de cualquier tecnología son las aplicaciones que se pueden realizar basándose en ella, por lo que la industria está dedicando tiempo, esfuerzo y dinero en desarrollar varias aplicaciones.

Aplicaciones hasta ahora:

- Compras en contexto
- Mapas y navegación
- Búsqueda, acceso tradicional a Internet
- Comunicaciones, Chat de texto
- Aplicaciones corporativas

Aplicaciones corporativas:

- Ventas en terreno
- Time tracking
- Inventario
- Fichas médicas

El problema parece ser que se necesita algo más que acceso a "información" para estar dispuesto a pagar.

I.2.7 Las particularidades de los dispositivos móviles

Los dispositivos inalámbricos presentan particularidades:

- ***Dificultades en la comunicación con el humano:*** pantallas muy chicas, dispositivos de entrada lentos o agobiadores.
- ***Dificultades tecnológicas:*** fuente de energía limitada y menor velocidad de procesamiento.
- ***Dificultades en la corrección de red:*** tiempos de latencia muy prolongados, y ancho de banda altamente variable, por infinidad de factores como el cambio de celda, condiciones de tráfico, competencia con la señal de voz, etc.

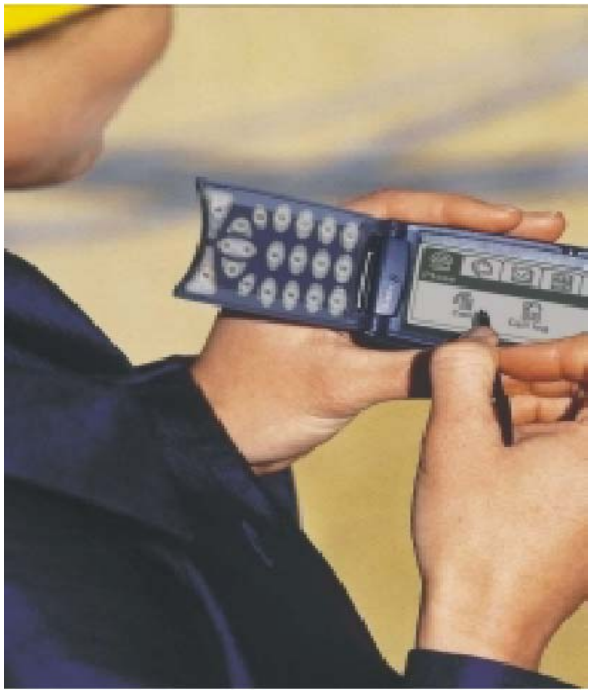


Figura I.4 *Dispositivo Móvil*

Respecto al uso, el problema mayor es que los usuarios de estos sistemas en teoría están además haciendo otras cosas, por lo que la aplicación no es el foco de su atención como en un PC de escritorio.

Otro punto respecto al mercado es que un celular, por ejemplo, es más barato que una computadora y por lo tanto el público es más amplio; por lo tanto serán muy sensibles al precio que deban pagar por el uso de sus aplicaciones.

Debido a estas particularidades, se estimó que una serie de protocolos debían ser creados. Esto para maximizar la eficiencia de los pocos recursos disponibles.

I.3 Transmisión de Información en dispositivos inalámbricos

I.3.1 Teléfonos celulares

La mayor revolución en las comunicaciones sin cables empezó con los teléfonos móviles. Los teléfonos móviles han sido el producto electrónico con más éxito de todos los tiempos.

Cuando se dieron a conocer por primera vez, los teléfonos móviles tenían una función principal (ofrecer comunicación por voz) pero esto ha cambiado. Los sistemas de telefonía actuales cada vez utilizan más recursos de las tecnologías informáticas. Y con baterías de mayor duración, interfaces inteligentes, reconocimiento de voz y mayor velocidad, el uso del teléfono móvil está destinado a dispararse aún más en un futuro próximo, aunque utilizando cada vez más sus nuevos servicios inalámbricos y cada vez menos sus tradicionales servicios de teléfono estándar.

Cronología de la telefonía móvil

1947	Se definen los conceptos básicos de los teléfonos celulares.
1947	AT&T propone a la FCC[1] la asignación de un gran número de radiofrecuencias, proposición que fue rechazada.
1968	La FCC reconsidera su posición respecto a la asignación de frecuencias para liberar espacio en el espectro radioeléctrico para los teléfonos móviles.
1977	AT&T desarrolla y pone en funcionamiento un prototipo de sistema celular.
1979	Empieza a funcionar en Tokio el primer sistema comercial de telefonía celular.
1982	La FCC autoriza el servicio celular comercial en E.U.A.
1987	En EUA, el número de abonados a la telefonía celular supera el millón.
1987	Se crea el estándar GSM (Global System for Mobile Communications) para Europa, basado en un híbrido de las tecnologías FDMA (analógica) y TDMA (digital).
1993	El CDMA (Code Division Multiple Access) se acepta como estándar para la separación de señales de voz utilizando tecnología de espectro ensanchado.
1994	El TDMA (Time Division Multiple Access) se publica como el estándar para las redes sin cables.
1994-1997	La FCC empieza a subastar espacio para la nueva banda PCS, una banda de frecuencia más elevada.
2000/2001	Llega el GPRS (General Packet Radio Services), que ofrece un enlace más rápido para datos sobre un enlace estándar GSM que está siempre conectado.

Para dar cobertura de transmisiones inalámbricas a una área geográfica determinada formando una red de área extensa (WAN), ésta se divide en zonas más pequeñas, denominadas células. Cada célula es la zona cubierta por una estación base radio de baja potencia con sus correspondientes antenas, y operando a frecuencias de radio individuales, que se repiten una y otra vez en otras células no adyacentes. Las llamadas realizadas en estas células se gestionan en las estaciones base o en conmutadores móviles. Estos últimos están conectados a bases de datos que hacen de interfaz entre la red inalámbrica y la red de telefonía cableada.

Cuando un teléfono móvil cruza los límites entre dos células, detecta que la señal de la célula que abandona es cada vez más débil, transfiriéndose automáticamente la llamada a la antena de la célula a la que se dirige cuando su señal sea la más potente.
Y como el sistema funciona con un nivel de potencia muy bajo, las frecuencias utilizadas en una célula determinada no producen ningún tipo de interferencia en células adyacentes.

[1] Federal Communications Commission

I.3.2 Métodos de Acceso Celular

Los usuarios que ocupan un área geográfica dada deben disputarse un número limitado de canales y existen varios modos de dividir el espectro para proporcionar acceso de forma organizada

El FDMA (Frequency Division Multiple Access) divide un espectro disponible en franjas no solapadas en la dimensión o dominio de la *frecuencia*. El FDMA es el modo más familiar de dividir un espectro y tradicionalmente ha sido utilizado por los sistemas analógicos.

El TDMA (Time Division Multiple Access) divide un espectro disponible en franjas no solapadas en la dimensión o dominio del *tiempo*. Los sistemas digitales son típicamente una combinación de FDMA y TDMA, donde la capacidad disponible se divide tanto en dimensiones de frecuencia como de tiempo, asignando a los usuarios canales de distintas frecuencias que utilizan en distintas franjas de tiempo.

El GSM (Global System for Mobile Communications) es un tipo de red digital inalámbrica TDMA con características de cifrado y usada ampliamente por toda Europa a 900 MHz.

El CDAM (Code Division Multiple Access) está basado en el concepto de espectro ensanchado, lo que significa que múltiples conversaciones comparten simultáneamente un espectro disponible y se distinguen entre sí mediante codificación en vez de usar canales de frecuencia o de tiempo.

Una compañía de telefonía inalámbrica que opera en una área geográfica definida ofrece este acceso WAN al usuario móvil en la forma de diversos planes y opciones de llamadas mensuales. Cuando sus abonados viajan fuera del área geográfica cubierta por la compañía, se les considera “en itinerancia” (*roaming*). Su compañía local transfiere el servicio a la compañía que opera esa área, con un coste de llamada mayor.

Existen dos tipos básicos de señales: *analógica* y *digital*. Una señal analógica puede tomar cualquier valor intermedio entre un máximo y un mínimo. Un ejemplo de señal analógica es la voz humana. Una señal digital no puede tomar cualquier valor, sino sólo un conjunto limitado de valores llamados símbolos, que pueden representar números o caracteres alfabéticos.

Ejemplos de señal digital son un impulso de corriente en un cable o un impulso de luz en un cable de fibra óptica. Los sistemas inalámbricos tienden cada vez más hacia los sistemas digitales y el uso de formas avanzadas de modulación digital. Esto es debido a que la señal digital es más inmune al ruido y su manipulación o procesamiento es más sencillo que el de una señal analógica.

¿Y cuál es la diferencia entre teléfonos celulares y teléfonos PCS[2]? Un teléfono celular utiliza transmisión analógica de onda corta o transmisión digital a través de una conexión sin cables. Un teléfono PCS es similar a un teléfono celular, pero utiliza exclusivamente transmisión digital, ofrece mayor movilidad y funciona a frecuencias más elevadas para conseguir una conexión de mayor calidad.

[2] Personal Communication System

Referencias

[Bunch, 1993] B. Bunch y A. Hellemans. *The Timetables of Technology*. Simon & Schuster, 1993

[Oberliesen, 1982] R. Oberliesen. *Information, Daten und Signale*. Rowohlt Taschenbuch Verlag Gmbh, 1982

[Kuhlmann, 1996] Kuhlmann Federico y Alonso Antonio. *Información y Telecomunicaciones*. Fondo de Cultura Económica, 1996

[Shannon, 1949] C. E. Shannon y W. Weaver. *A Mathematical Theory of Communication*. University of Illinois Press, USA, 1949

[WMLClub] http://www.wmlclub.com

[WAP] http://www.wapforum.org

CAPÍTULO II.
Protocolo de Aplicaciones Inalámbricas (WAP)

II.1 Concepto del WAP[1]

El *Protocolo de Aplicaciones Inalámbricas* surge como la combinación de dos tecnologías de amplio crecimiento y difusión durante los últimos años: *Las Comunicaciones Inalámbricas* e *Internet*.

Mas allá de la posibilidad de acceder a los servicios de información contenidos en Internet, el protocolo pretende proveer de servicios avanzados adicionales como, por ejemplo, el desvío de llamadas inteligente, en donde se proporcione una interfaz al usuario en el cual se le pregunte la acción que desea realizar: aceptar la llamada, desviarla a otra persona, desviarla a un buzón vocal, etc.

Para ello, se requiere que sea parte de una arquitectura basada en la arquitectura definida para el *World Wide Web (WWW)*, pero adaptada a los nuevos requisitos del sistema. En la Figura II.1 se muestra el esquema de la arquitectura WAP.

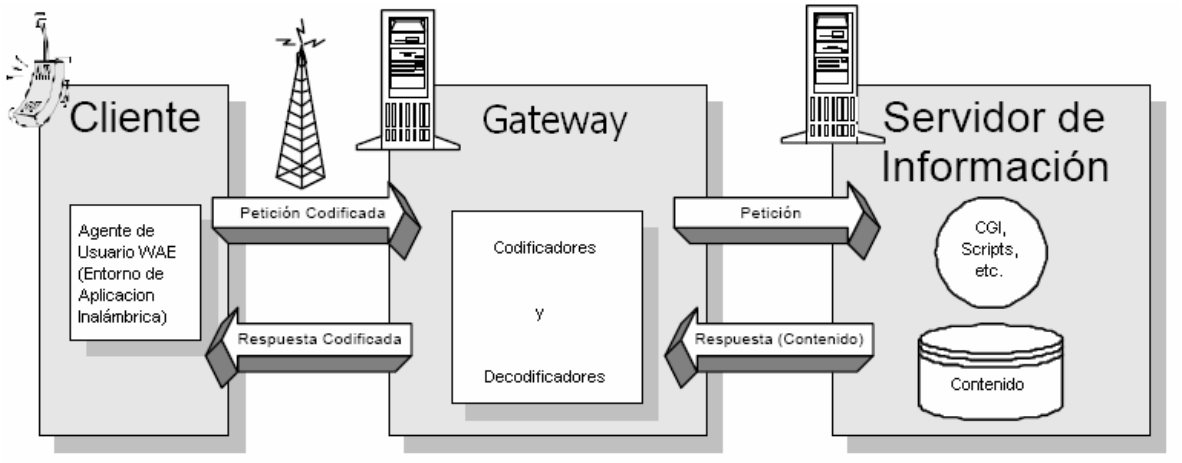


Figura II.1 *Modelo de funcionamiento del WAP*

De esta forma, en el terminal inalámbrico existiría un *"micro navegador"* encargado de la coordinación con el *gateway*, al cual le realiza peticiones de información que son tratadas y redirigidas al servidor de información adecuado. Se pretende que este micro navegador actúe como interfaz con el usuario de la misma forma que lo hacen los navegadores estándar. Una vez procesada la petición de información en el servidor, se envía esta información al *gateway* que de nuevo la procesa para enviarlo al terminal inalámbrico.

[1] Wireless Aplication Protocol - Protocolo de Aplicaciones Inalámbricas

Para conseguir consistencia en la comunicación entre el terminal móvil y los servidores de red que proporcionan la información, WAP define un conjunto de componentes estándar:

- ***Un modelo de nombres estándar.*** Se utilizan las URIs (Universal/Uniform Resource Identifier -Identificador Uniforme/Universal de Recurso-) definidas en WWW para identificar los recursos locales del dispositivo (tales como funciones de control de llamada) y las URLs (Universal/Uniform Resource Location - Localización Universal/Uniforme de Recurso-) también definidas en el WWW, para identificar el contenido WAP en los servidores de información.
- ***Un formato de contenido estándar***, basado en la tecnología WWW.
- ***Protocolos de comunicación estándar***, que permitan la comunicación del *micro navegador* del terminal móvil con el servidor Web en red.

En la figura II.2 se ilustra un modelo global del funcionamiento de este sistema.

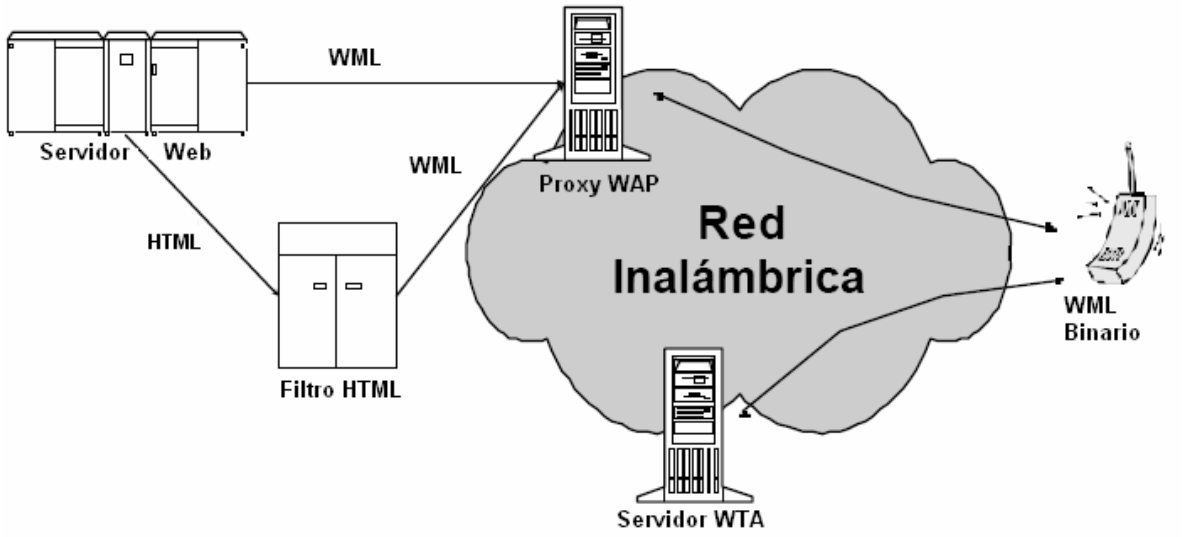


Figura II.2: *Ejemplo de una red WAP*

En el ejemplo de la figura II.2, nuestra terminal móvil tiene dos posibilidades de conexión: a un *proxy* WAP, o a un servidor WTA. El primero de ellos, el *proxy* WAP traduce las peticiones WAP a peticiones Web, de forma que el cliente WAP (el terminal inalámbrico) pueda realizar peticiones de información al servidor Web. Adicionalmente, este *proxy* codifica las respuestas del servidor Web en un formato binario compacto, que es interpretable por el cliente. Por otra parte, el segundo de ellos, el Servidor WTA (Wireless Telephony Application - Aplicación de Telefonía Inalámbrica-) está pensado para proporcionar acceso WAP a las facilidades proporcionadas por la infraestructura de telecomunicaciones del proveedor de conexiones de red.

II.2 Componentes de la Arquitectura WAP

Una vez introducido el sistema, vamos a ver la arquitectura que le da consistencia. La arquitectura WAP está pensada para proporcionar un *"entorno escalable y extensible para el desarrollo de aplicaciones para dispositivos de comunicación móvil"*. Para ello, se define una estructura en capas, en la cual cada capa es accesible por la capa superior así como por otros servicios y aplicaciones a través de un conjunto de interfaces muy bien definidos y especificados. Este esquema de capas de la arquitectura WAP la podemos ver en la Figura II.3.

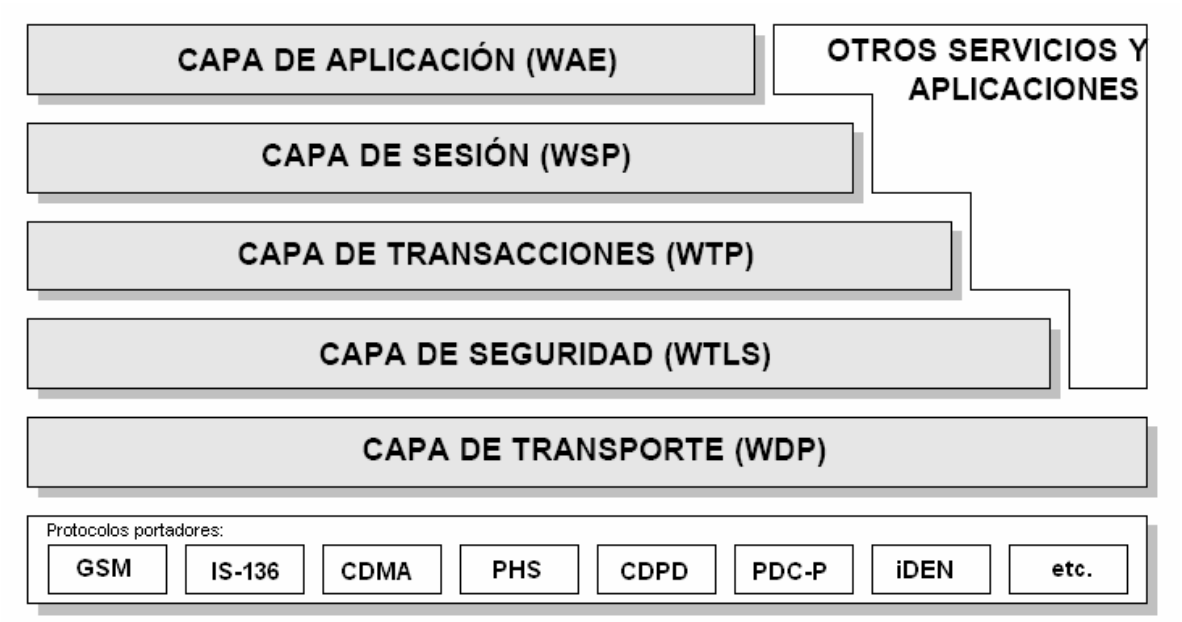


Figura II.3 *Arquitectura de WAP*

A continuación se hace una descripción de estas capas.

II.2.1 Capa de Aplicación WAE (Wireless Application Environment)

El *Entorno Inalámbrico de Aplicación (WAE)* es un entorno de aplicación de propósito general basado en la combinación del *World Wide Web* y tecnologías de Comunicaciones Móviles.

Este entorno incluye un *micro navegador*, del cual ya hemos hablado anteriormente, que posee las siguientes funcionalidades:

- Un lenguaje denominado WML(*Wireless Markup Language*) similar al HTML, pero optimizado para su uso en terminales móviles.
- Un lenguaje denominado *WMLScript*, similar al *JavaScript* (esto es, un lenguaje para su uso en forma de *Script*)

- Un conjunto de formatos de contenido, que son un conjunto de formatos de datos bien definidos entre los que se encuentran imágenes, entradas en la agenda de teléfonos e información de calendario.

II.2.2 CAPA DE SESIÓN WSP (Wireless Session Protocol)

El *Protocolo Inalámbrico de Sesión (WSP)* proporciona a la Capa de Aplicación de WAP interfaz con dos servicios de sesión: Un servicio orientado a conexión que funciona por encima de la Capa de Transacciones y un servicio no orientado a conexión que funciona por encima de la Capa de Transporte (y que proporciona servicio de datagramas seguro o servicio de datagramas no seguro)

Actualmente, esta capa consiste en servicios adaptados a aplicaciones basadas en la navegación Web, proporcionando las siguientes funcionalidades:

- Semántica y funcionalidades del HTTP/1.1 en una codificación compacta.
- Negociación de las características del Protocolo.
- Suspensión de la Sesión y reanudación de la misma con cambio de sesión.

II.2.3 Capa de Transacciones WTP (Wireless Transaction Protocol)

El *Protocolo Inalámbrico de Transacción (WTP)* funciona por encima de un servicio de datagramas, tanto seguros como no seguros, proporcionando las siguientes funcionalidades:

- Tres clases de servicio de transacciones:
    - Peticiones inseguras de un solo camino.
    - Peticiones seguras de un solo camino.
    - Transacciones seguras de dos caminos (petición-respuesta)
- Seguridad usuario-a-usuario opcional.
- Transacciones asíncronas.

II.2.4 Capa de Seguridad WTLS (Wireless Transport Layer Security)

La *Capa Inalámbrica de Seguridad de Transporte (WTLS)* es un protocolo basado en el estándar SSL, utilizado en el entorno Web para la proporción de seguridad en la realización de transferencias de datos. Este protocolo ha sido especialmente diseñado para los protocolos de transporte de WAP y optimizado para ser utilizado en canales de comunicación de banda estrecha. Para este protocolo se han definido las siguientes características:

- ***Integridad de los datos.*** Este protocolo asegura que los datos intercambiados entre el terminal y un servidor de aplicaciones no han sido modificados y no es información corrupta.

- ***Privacidad de los datos.*** Este protocolo asegura que la información intercambiada entre el terminal y un servidor de aplicaciones no puede ser entendida por terceras partes que puedan interceptar el flujo de datos.
- ***Autentificación.*** Este protocolo contiene servicios para establecer la autenticidad del terminal y del servidor de aplicaciones.

Adicionalmente, el WTLS puede ser utilizado para la realización de comunicación segura entre terminales, por ejemplo en el caso de operaciones de comercio electrónico entre terminales móviles.

II.2.5 Capa de Transporte WDP (Wireless Datagram Protocol)

El *Protocolo Inalámbrico de Datagramas (WDP)* proporciona un servicio fiable a los protocolos de las capas superiores de WAP y permite la comunicación de forma transparente sobre los protocolos portadores válidos.

Debido a que este protocolo proporciona una interfaz común a los protocolos de las capas superiores, las capas de Seguridad, Sesión y Aplicación pueden trabajar independientemente de la red inalámbrica que dé soporte al sistema.

A continuación se muestran tres ejemplos de interconexión de estas capas:

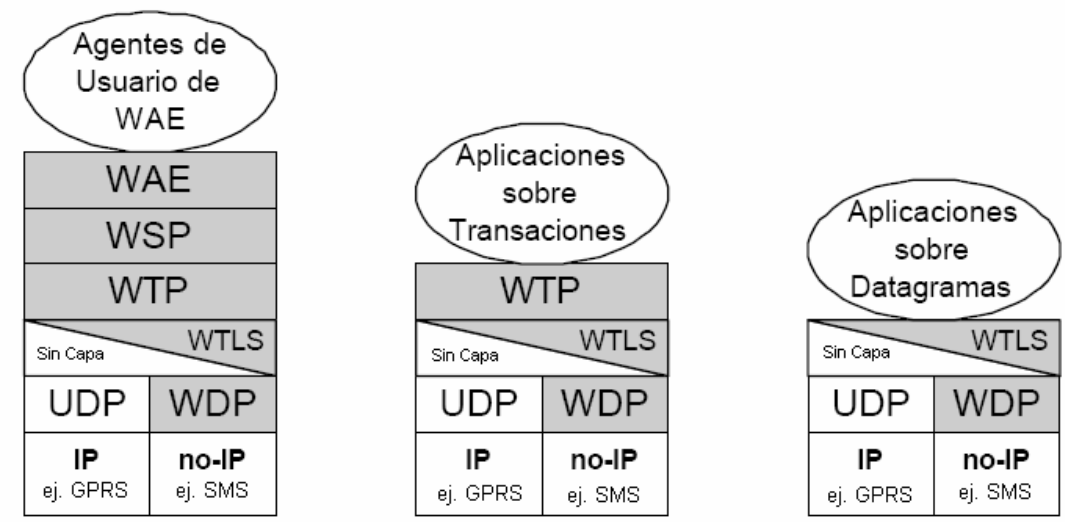


Figura II.4 *Ejemplo de capas en WAP*

Así pues, dependiendo de la aplicación en cuestión, la comunicación se realizará con una determinada capa de la estructura de WAP.

II.3 El entorno Inalámbrico de Aplicaciones

El objetivo del *Entorno Inalámbrico de Aplicaciones* es construir un entorno de aplicación de propósito general, basado fundamentalmente en la filosofía y tecnología del *World*

Wide Web (WWW). Principalmente, se pretende establecer un entorno que permita a los operadores y proveedores de servicios construir aplicaciones y servicios que puedan utilizarse en una amplia variedad de plataformas inalámbricas de forma útil y eficiente.

De esta forma, la arquitectura del Entorno Inalámbrico de Aplicaciones (en adelante WAE) está enfocado principalmente sobre los aspectos del cliente de la arquitectura del sistema de WAP, esto es, de los puntos relacionados con los agentes de usuario. Esto es debido a que la parte que más interesa de la arquitectura es aquella que afecta principalmente a los terminales móviles, esto es, a aquellos puntos en los cuales van a estar ejecutándose los diversos agentes de usuario. Un agente de usuario es todo aquel software o dispositivo que interpreta un contenido, p. e. WML. Esto incluye navegadores de texto, navegadores de voz, sistemas de búsqueda, etc.

Si volvemos sobre la Figura II.1, vemos que entre los agentes de usuario localizados en el cliente (en el terminal móvil) y los servidores de información se define un nuevo elemento: *Los Gateways*. Su función es codificar y decodificar la información intercambiada con el cliente, para así minimizar la cantidad de datos radiados, así como minimizar el proceso de la información por parte del cliente.

Basándonos en esta arquitectura, podemos profundizar un poco más en los componentes de este Entorno Inalámbrico de Aplicación. Tal y como podemos observar en la figura II.5, se divide en dos partes, dos capas lógicas:

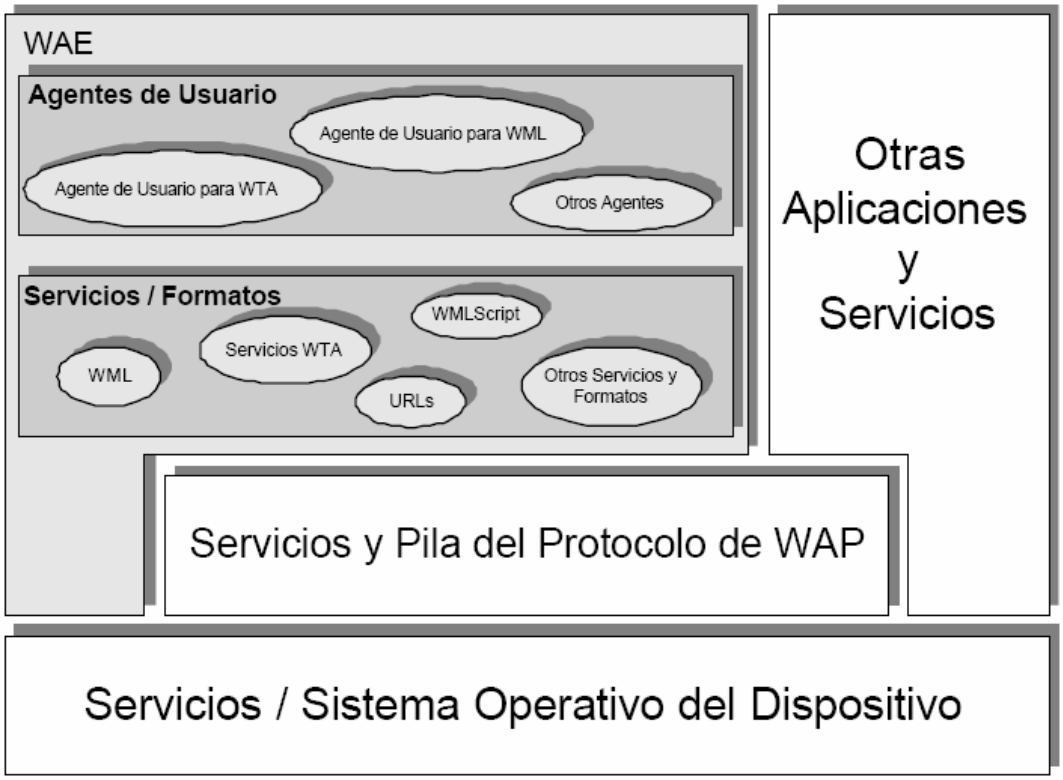


Figura II.5 *Componentes del Cliente de WAE*

- ***Los Agentes de Usuario***, que incluye aquellos elementos como navegadores, agendas telefónicas, editores de mensajes, etc.
- ***Los Servicios y Formatos***, que incluyen todos aquellos elementos y formatos comunes, accesibles a los Agentes de Usuario, tales como WML, WMLScript, formatos de imagen,etc.

Como se puede ver en la Figura II.5, dentro de WAE se separan Servicios de Agentes de Usuario, lo que proporciona flexibilidad para combinar varios Servicios dentro de un único Agente de Usuario, o para distribuir los Servicios entre varios Agentes de Usuario.

Los dos Agentes de Usuario más importantes son el Agente de Usuario para WML y el Agente de Usuario para WTA. El Agente de Usuario para WML es el Agente de Usuario fundamental en la arquitectura del Entorno Inalámbrico de Aplicación. A pesar de su importancia, este Agente de Usuario no está definido formalmente dentro de esta arquitectura, ya que sus características y capacidades se dejan en manos de los encargados de su implementación. El único requisito de funcionalidad que debe cumplir este Agente de Usuario, es el proporcionar un sistema intérprete a los lenguajes WML y WMLScript, de forma que se permita la navegación desde el terminal móvil.

Por otra parte, el Agente de Usuario para WTA permite a los autores acceder e interactuar con las características de los teléfonos móviles (p. e. Control de Llamada), así como otras aplicaciones supuestas en los teléfonos, tales como agendas de teléfono y aplicaciones de calendario.

II.4 El Protocolo Inalámbrico de Sesión

El *Protocolo Inalámbrico de Sesión* constituye la capa que se sitúa por debajo de la capa de Aplicación, proporcionando la capacidad necesaria para:

- Establecer una conexión fiable entre el cliente y el servidor, y liberar esta conexión de una forma ordenada.
- Ponerse de acuerdo en un nivel común de funcionalidades del protocolo, a través de la negociación de las posibilidades.
- Intercambiar contenido entre el cliente y el servidor utilizando codificación compacta.
- Suspender y recuperar la sesión.

Hoy por hoy, este protocolo ha sido definido únicamente para el caso de la navegación, definiéndose como WSP/B (Wireless Session Protocol -- Browsing). Esta implementación está realizada para el establecimiento de una conexión sobre la base de un protocolo compatible con HTTP1.1.

De esta forma, se han definido un conjunto de primitivas de servicio (una primitiva de servicio representa el intercambio lógico de información entre la capa de Sesión y capas adyacentes) para permitir la comunicación entre la capa de sesión integrada dentro del equipo cliente y la capa de sesión integrada en el equipo servidor. Estas primitivas, junto con una pequeña descripción de las mismas, puede verse en la tabla II.1:

Nombre de la primitiva	Descripción
S-Connect	Esta primitiva se utiliza para iniciar el establecimiento de la conexión, y para la notificación de su éxito
S-Disconnect	Esta primitiva se utiliza para desconectar una sesión, y para notificar al usuario de una sesión que esa sesión no se puede establecer, que ha sido desconectada
S-Suspend	Esta primitiva se utiliza para solicitar la suspensión de la sesión
S-Resume	Esta primitiva se utiliza para solicitar que se recupere la sesión utilizando para las direcciones el nuevo identificador de punto de acceso de servicio.
S-Exception	Esta primitiva se utiliza para notificar aquellos eventos que no están asignados a una transacción en particular, ni provocan la desconexión o suspensión de la sesión.
S-MethodInvoke	Esta primitiva se utiliza para solicitar una operación que deba ser ejecutada en el servidor.
S-MethodResult	Esta primitiva se utiliza para devolver una respuesta a una petición de operación.
S-MethodAbort	Esta primitiva se utiliza para abortar una solicitud de ejecución de operación, que no haya sido aún completada.
S-Push	Esta primitiva se utiliza para enviar información no solicitada desde el servidor, dentro del contexto de una sesión de forma y sin confirmación.
S-ConfirmedPush	Esta primitiva realiza las mismas funciones que la anterior, pero con confirmación.
S-PushAbort	Esta primitiva se utiliza para anular una primitiva anterior del tipo *S-Push* o *SConfirmedPush.*

Tabla II.1 *Primitivas de Servicio de Sesión*

Adicionalmente, existen cuatro tipos de cada una de estas primitivas, tal y como puede verse en la tabla II.2:

Tipo	Abreviación	Descripcion
Request	req	Se utiliza cuando una capa superior solicita un servicio de la capa inmediatamente inferior
Indication	ind	Una capa que solicita un servicio utiliza este tipo de primitiva para notificar a la capa inmediatamente superior de las actividades relacionadas con su par, o con el proveedor del servicio
Response	res	Este tipo de primitiva se utiliza para reconocer la recepción de la primitiva de tipo *Indication* de la capa inmediatamente inferior

Tipo	Abreviación	Descripcion
Confirm	cnf	La capa que proporciona el servicio requerido utiliza este tipo de primitiva para notificar que la actividad ha sido completada satisfactoriamente.

Tabla II.2 *Tipos de Primitivas de Servicio.*

Por último, cabe señalar que cada una de estas primitivas está perfectamente definida dentro de la especificación, tanto desde el punto de vista del diagrama de tiempos en el que se tienen que invocar las primitivas, como desde el punto de vista de los parámetros intercambiados.

II.5 El Protocolo Inalámbrico de Transacción

El *Protocolo Inalámbrico de Transacción* se establece para proporcionar los servicios necesarios que soporten aplicaciones de "*navegación"* (del tipo petición/respuesta). Es a este dúo petición/respuesta, lo que vamos a denominar como transacción. Este protocolo se sitúa por encima del *Protocolo Inalámbrico de Datagramas* y, de forma opcional, de la *Capa Inalámbrica de Seguridad de Transporte*, que serán estudiados posteriormente.

Las características de este protocolo son:

- Proporciona tres clases de servicios de transacción:
  - Clase 0: mensaje de solicitud no seguro, sin mensaje de resultado.
  - Clase 1: mensaje de solicitud seguro, sin mensaje de resultado.
  - Clase 2: mensaje de solicitud seguro, con, exactamente, un mensaje de resultado seguro.

- La seguridad se consigue a través del uso de identificadores únicos de transacción, asentimientos, eliminación de duplicados y retransmisiones.

- Seguridad opcional usuario a usuario.

- De forma opcional, el último asentimiento de la transacción puede contener algún tipo de información adicional relacionada con la transacción, como medidas de prestaciones, entre otras.

- Se proporcionan mecanismos para minimizar el número de transacciones que se reenvían como resultado de paquetes duplicados.

- Se permiten las transacciones asíncronas.

Al igual que en el protocolo anterior (el protocolo inalámbrico de sesión), en la tabla II.3 vamos a ver las primitivas de servicio (estas primitivas pueden ser de cuatro tipos, tal y como se puede ver en la tabla II.2) que sustentan la comunicación entre dos capas de transacciones situadas en dos equipos distintos:

Nombre de la primitiva	Descripción
TR-Invoke	Esta primitiva se utiliza para iniciar una nueva transacción.
TR-Result	Esta primitiva se utiliza para devolver el resultado de transacción iniciada anteriormente
TR-Abort	Esta primitiva se utiliza para abortar una transacción existente

Tabla II.3 *Primitivas de Servicio de Transacción*

A modo de ejemplo, vamos a ver en la figura II.6 la concatenación de Primitivas de Servicio de Sesión y de Transacción para el caso de una petición-respuesta:

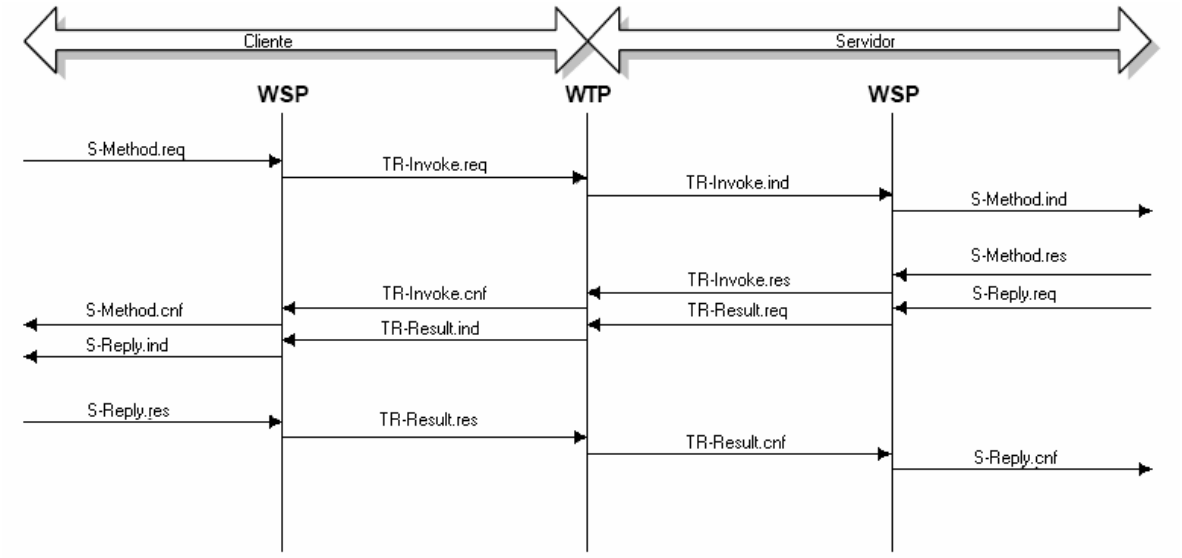


Figura II 6 *Ejemplo intercambio de primitivas entre capa Sesión y Transacción*

Para finalizar, vamos a detallar un poco más las principales características de este protocolo:

1. Transferencia de Mensajes.
Dentro de este protocolo se distinguen dos tipos de mensajes: mensajes de datos y mensajes de control. Los mensajes de datos transportan únicamente datos de usuario, mientras que los mensajes de control se utilizan para los asentimientos, informes de error, etc., pero sin transportar datos de usuario.

2. Retransmisión hasta el asentimiento.
Esta característica se utiliza para la transferencia fiable de datos desde un proveedor WTP a otro, en caso que haya pérdida de paquetes. A modo de comentario, dejar claro que para reducir lo máximo posible el número de paquetes que se transmiten, este protocolo utiliza asentimiento explícito siempre que sea posible.

3. Asentimiento de usuario.
El Asentimiento de Usuario permite al usuario de este protocolo, confirmar cada mensaje recibido por el proveedor WTP.

4. Información en el Último Asentimiento.
Se permite, así pues, enviar información en el último, y únicamente en el último, asentimiento de una transacción. De esta forma, se puede enviar, por ejemplo, información del rendimiento proporcionado por el sistema durante la transacción realizada, etc.

5. Concatenación y Separación.
Podemos definir concatenación como el proceso de trasmitir múltiples Unidades de Datos del Protocolo (PDU - Protocol Data Unit-) de WTP en una Unidad de Datos del Servicio (SDU - Service Data Unit-) de la red portadora.

Por el contrario, separación es el proceso de separar múltiples PDUs de un único SDU (esto es, el proceso inverso al anterior).
El objetivo de estos sistemas es proveer eficiencia en la transmisión inalámbrica, al requerirse un menor número de transmisiones.

6. Transacciones Asíncronas.
Para un correcto funcionamiento del protocolo, múltiples transacciones deben ser procesadas de forma asíncrona, debe ser capaz de iniciar múltiples transacciones antes que reciba la respuesta a la primera transacción.

7. Identificador de la Transacción.
Cada transacción está identificada de forma única por los pares de direcciones de los sockets (Dirección fuente, puerto fuente, dirección destino y puerto destino) y por el Identificador de Transacción (TID - Transaction Identifier -), el cual se incrementa para cada una de las transacciones iniciadas. Este número es de 16 bits, utilizándose el bit de mayor orden para indicar la dirección.

8. Segmentación y re-ensamblado. (opcional)
Si la longitud del mensaje supera la Unidad Máxima de Transferencia (MTU - Maximum Transfer Unit -), el mensaje puede ser segmentado por el WTP y enviado en múltiples paquetes. Cuando esta operación se realiza, estos paquetes pueden ser enviados y asentidos en grupos. De esta forma, el emisor puede realizar control de flujo cambiando el tamaño de los grupos de mensajes dependiendo de las características de la red.

II.6 La Capa Inalámbrica de Seguridad de Transporte

La *Capa Inalámbrica de Seguridad de Transporte* (en adelante WTLS), constituye una capa modular, que depende del nivel de seguridad requerido por una determinada aplicación. Esta capa proporciona a las capas de nivel superior de WAP de una interfaz de servicio de transporte seguro, que lo resguarde de una interfaz de transporte inferior.

El principal objetivo de esta capa es proporcionar privacidad, integridad de datos y autentificación entre dos aplicaciones que se comuniquen.

Adicionalmente, la WTLS proporciona una interfaz para el manejo de conexiones seguras.

En la tabla II.4 podemos ver las primitivas de servicio (Estas primitivas pueden ser de cuatro tipos, tal y como se mostró en la tabla II.2) que sustentan la comunicación entre dos capas situadas en dos equipos distintos:

Nombre de la primitiva	Descripción
SEC-Unitdata	Esta primitiva se utiliza para intercambiar datos de usuario entre los dos participantes. Sólo puede ser invocada cuando existe previamente una conexión segura entre las direcciones de transporte de los dos participantes.
SEC-Create	Esta primitiva se utiliza para iniciar el establecimiento de una conexión segura.
SEC-Exchange	Esta primitiva se utiliza en la creación de una conexión segura si el servidor desea utilizar autentificación de clave pública o intercambio de claves con el cliente.
SEC-Commit	Esta primitiva se inicia cuando el *handshake* (intercambio de primitivas entre cliente y servidor con el objetivo de establecer una sesión segura) se completa y cualquiera de los equipos participantes solicita cambiar a un nuevo estado de conexión negociado.
SEC-Terminate	Esta primitiva se utiliza para finalizar la conexión.
SEC-Exception	Esta primitiva se utiliza para informar al otro extremo sobre las alertas de nivel de aviso.
SEC-Create-Request	Esta primitiva se utiliza por el servidor para solicitar al cliente que inicie un nuevo *handshake*.

Tabla II.4 *Primitivas de Servicio de Capa de Seguridad*

Se mencionó en la tabla anterior del proceso de establecimiento de una sesión segura o *handshake*. En la figura II.7 podemos ver este intercambio de primitivas:

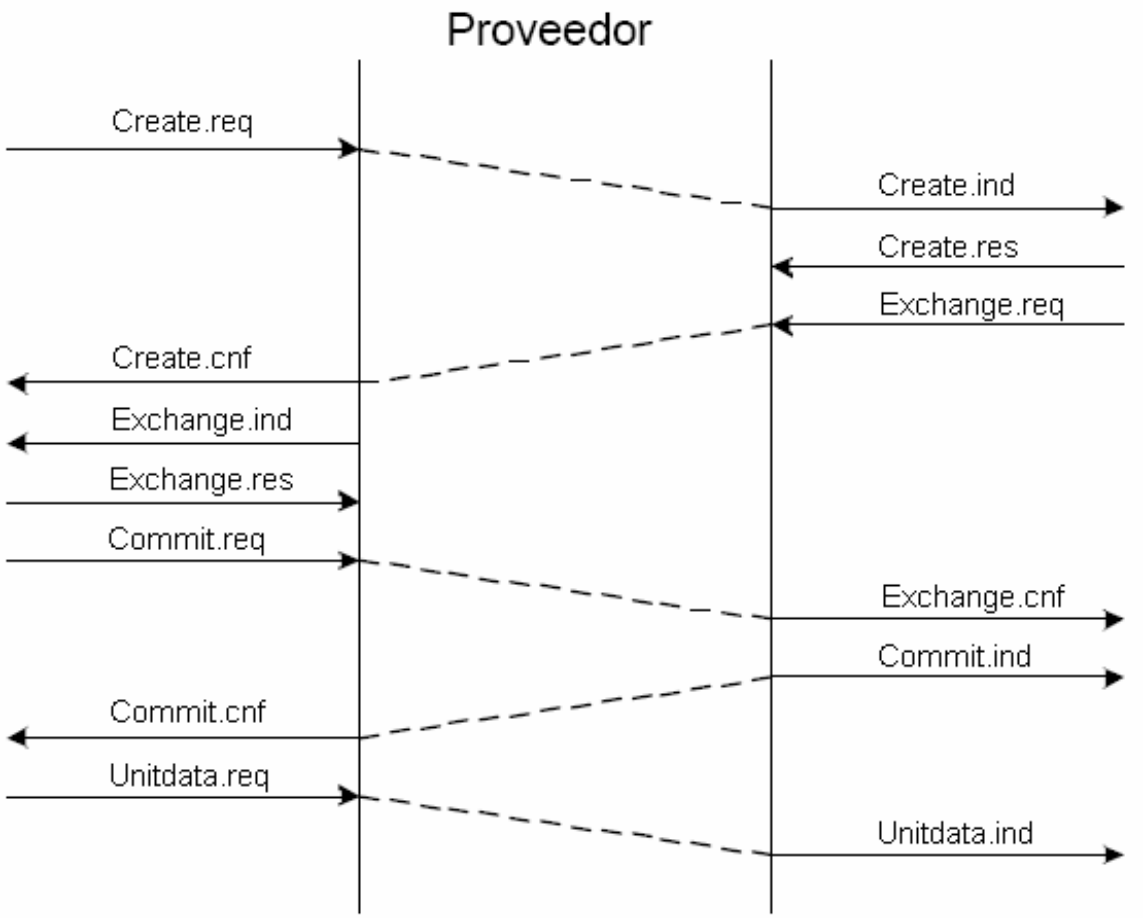


Figura II.7 *Secuencia de Primitivas para el establecimiento de una sesión segura*

II.7 El Protocolo Inalámbrico de Datagramas

El *Protocolo Inalámbrico de Datagramas* (en adelante WDP -Wireless Datagram Protocol-) ofrece un servicio consistente al protocolo (Seguridad, Transacción y Sesión) de la capa superior de WAP, comunicándose de forma transparente sobre uno de los servicios portadores disponibles.

Este protocolo ofrece servicios a los protocolos superiores del estilo a direccionamiento por número de puerto, segmentación y re-ensamblado (opcional) y detección de errores (opcional), de forma que se permite a las aplicaciones de usuario funcionar de forma transparente sobre distintos servicios portadores disponibles. Para ello, se plantea una arquitectura de protocolo como el que se muestra en la figura II.8:

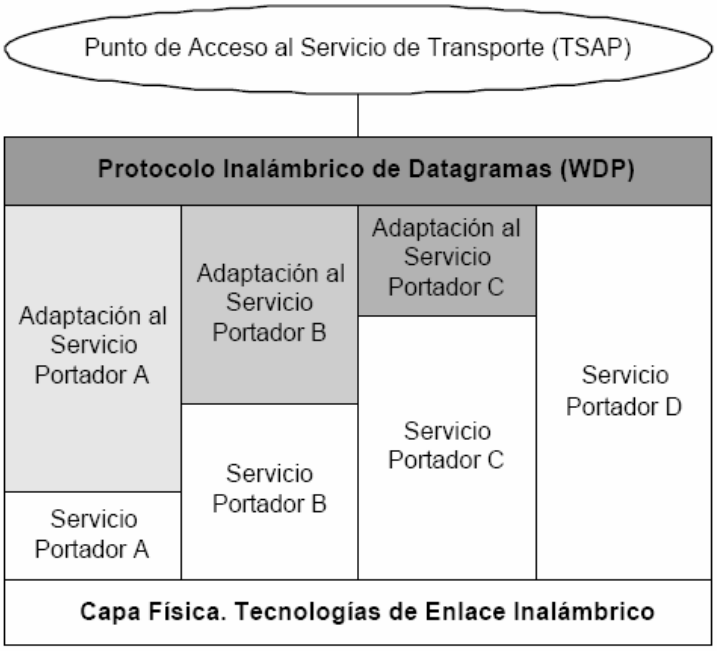


Figura II.8 *Arquitectura del Protocolo Inalámbrico de Datagramas*

En la tabla II.5 se definen las primitivas de servicio que se utilizan en este protocolo:

Nombre de la primitiva	Descripción
T-DUnitdata	Esta primitiva es la utilizada para transmitir datos como datagramas. No requiere que exista una conexión para establecerse.
T-DError	Esta primitiva se utiliza para proporcionar información a la capa superior cuando ocurre un error que pueda influenciar en el servicio requerido.

Tabla II.5 *Primitivas de Servicio de la Capa de Datagramas*

Por último, vamos a ver la arquitectura de este protocolo dentro de la arquitectura global de WAP, para el caso de utilizarse GSM (*Global System for Mobile Communications* -Sistema Global para Comunicaciones Móviles-) como servicio portador, que es el protocolo que más nos puede interesar por su amplia implantación en los sistemas de comunicaciones móviles telefónicas existentes hoy en día.

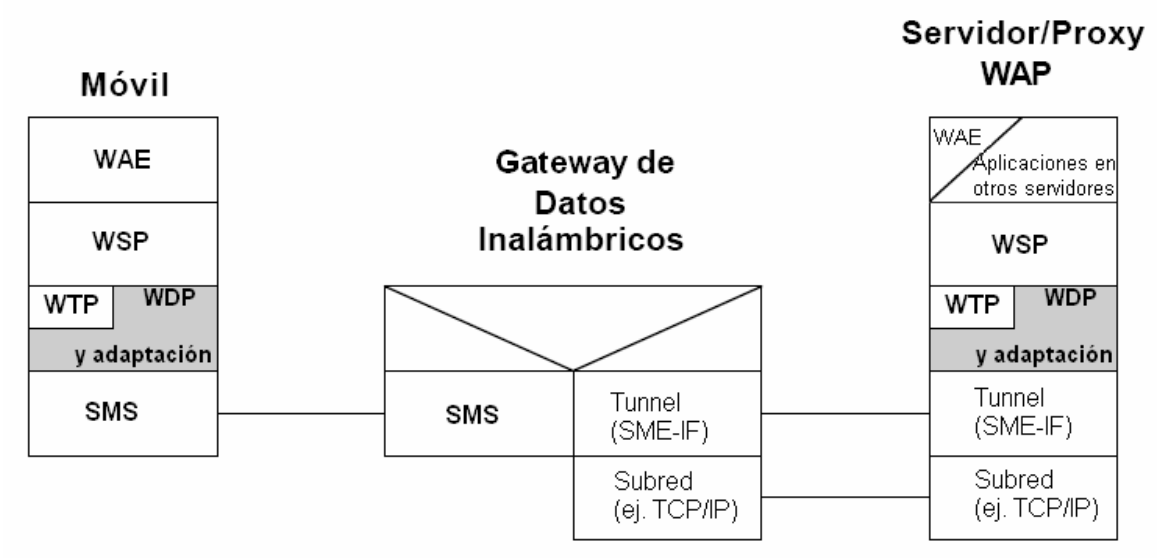


Figura II.9 *WDP sobre GSM SMS*

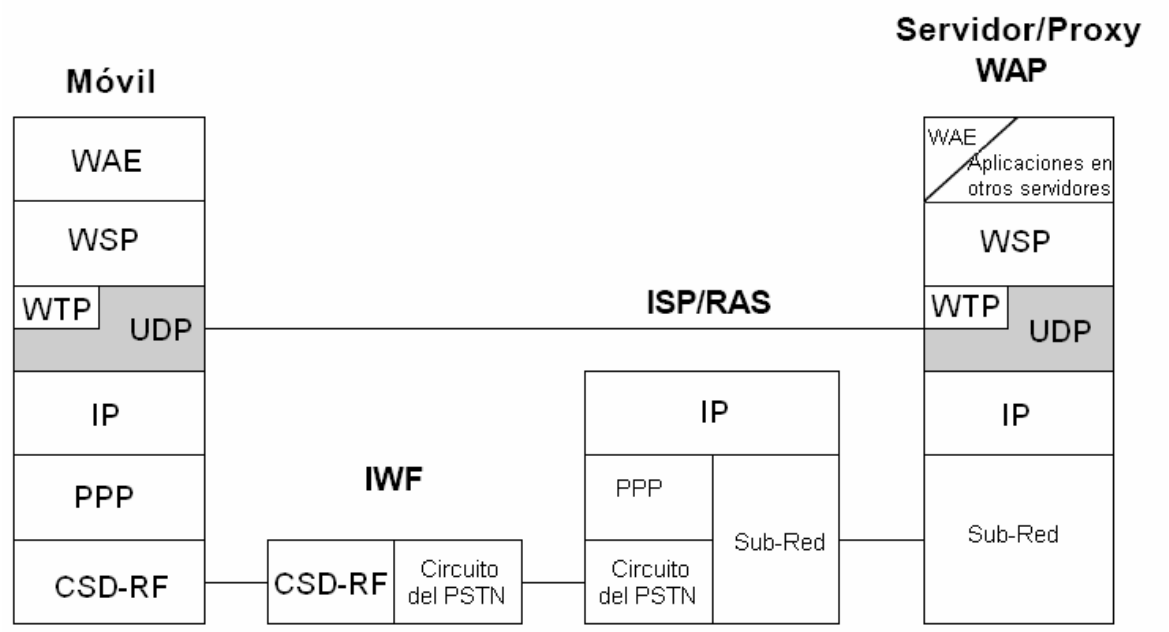


Figura II.10 *WDP sobre GSM Canal de Datos de Circuitos Conmutados*

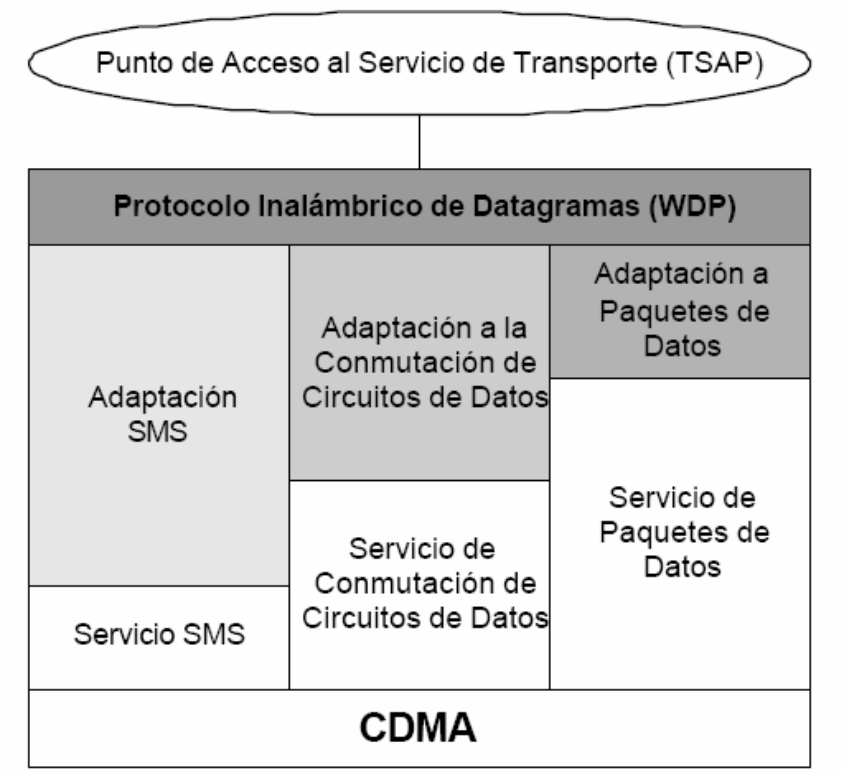


Figura II.11 *WDP sobre Servicios Portadores CDMA*

Referencias

[WAPARCH] "Wireless Application Protocol Architecture Specification"
[WAPWAEO] "Wireless Application Environment Overview"
[WAPWAES] "Wireless Application Environment Specification"
[WAPWDP] "Wireless Datagram Protocol Specification"
[WAPWML] "Wireless Markup Language Specification"
[WAPWSP] "Wireless Session Protocol Specification"
[WAPWTA] "Wireless Telephony Application Specification"
[WAPWTAI] "Wireless Telephony Application Interface Specification"
[WAPWTLS] "Wireless Transport Layer Security Specification"
[WAPWTP] "Wireless Transaction Protocol Specification"
[WAP] http://www.wapforum.org

CAPÍTULO III. Lenguaje de Marcación Inalámbrica (WML)

III.1 Introducción

Es un lenguaje tipo documento basado en rótulos (tags) o también conocido como lenguaje de marcación inalámbrica. WML es pariente cercano del HTML de la Web y del lenguaje para dispositivos portátiles de marcar (HDML), y está especificado como un tipo de documento XML.

Está optimizado para presentaciones específicas e interactuar con dispositivos con capacidades limitadas, lo que no es poco decir dada la amplia gama de ellos.

WML está diseñado para trabajar con dispositivos inalámbricos pequeños que poseen cuatro características:

- Pantalla pequeña de baja resolución. Por ejemplo, la mayoría de los teléfonos móviles pueden sólo mostrar unas pocas líneas de texto, y cada una de ellas puede contener solamente de 8 a 12 caracteres.

- Los aparatos de entrada tienen una capacidad limitada, o están diseñados para un propósito determinado. Un teléfono móvil tiene comúnmente teclas numéricas y un reducido número de teclas adicionales con funciones específicas. Dispositivos más sofisticados pueden poseer teclas programables de software, pero no un ratón ni otros dispositivos de selección.

- Los recursos computacionales están limitados por una CPU de baja potencia, una memoria reducida y una potencia restringida.

- La red ofrece un reducido ancho de banda y una alta latencia. No son infrecuentes los aparatos con conexiones de red de 300 bit/s a 10 Kbit/s y latencia "*round-trip*" de entre 5 y 10 segundos.

WML está basado en un subconjunto de HDML segunda versión (HDML2), introduciendo nuevos elementos, algunos de los cuales han sido modelados sobre elementos similares en HTML. El resultado es un lenguaje implementado como metáfora de cartas (card) y barajas (deck).

La interacción con el usuario está descrita en un conjunto de cartas, las cuales pueden ser agrupadas juntas en un único documento conocido como baraja. Lógicamente, cuando un usuario navega, lo hace a través de un conjunto de cartas, revisa el contenido de una, puede ingresar información requerida, puede hacer elecciones, y entonces se mueve a otra.

Es posible que instrucciones incrustadas en las cartas puedan invocar servicios localizados sobre servidores origen para realizar una particular interacción. Cuando esto

sucede, las barajas son traídas cuando son necesitadas desde archivos estáticos en los servidores, o pueden ser generadas dinámicamente por un generador de contenido corriendo sobre el servidor.

Cada carta, en una baraja, contiene particulares especificaciones de interacción con el usuario.

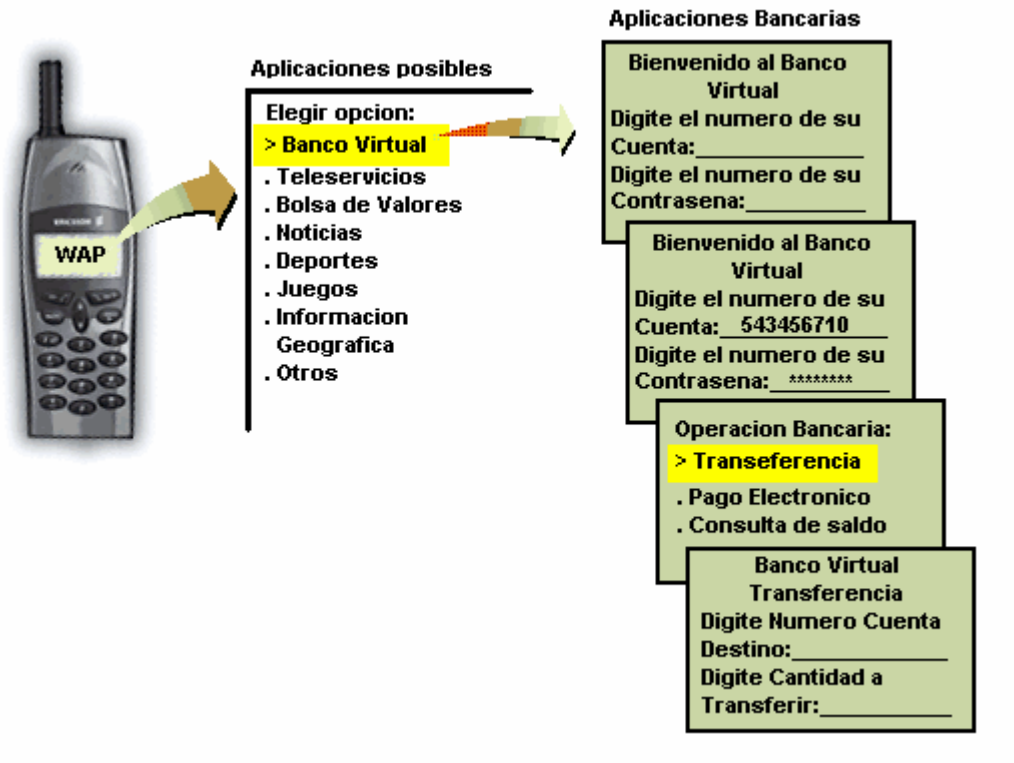


Figura III.1 *Barajas de cartas en WAP*

WML está especificada para permitir presentaciones en una amplia variedad de dispositivos, aún permitiendo a vendedores incorporar su propia interfaz. Lo que hace posible esta versatilidad es que WML no especifica cómo implementar los requerimientos de entrada desde el usuario, en vez de eso, especifica el intento en una manera abstracta.

Por ejemplo las distintas implementaciones pueden escoger entradas visuales, tal como muchos agentes de usuario de la Web, o pueden escoger usar interfaces activadas por la voz.

Estas decisiones las toma el agente de usuario teniendo en cuenta las capacidades del dispositivo y como él considere que sea la mejor forma de presentar todos los elementos presentes en la carta.

Por ejemplo, ciertos agentes de usuario podrían escoger presentar la información en un dispositivo de una pantalla grande en una sola carta, otros en cambio, con dispositivos de pantalla pequeña escogen fraccionar la información y desplegarla en muchas exposiciones.

Además WML posee una gran variedad de características entre las que se incluyen:

- ***Soporte para texto e imágenes.*** WML provee a los autores maneras de especificar texto e imágenes que serán presentadas al usuario. Esto puede incluir diseño y sugerencias de presentación. Tal y como otros lenguajes de marcación, WML requiere que el autor especifique la presentación en términos muy generales, y permita al agente de usuario determinar exactamente cómo debe ser presentada al usuario final. Para este fin, WML posee un conjunto de elementos para marcar texto entre los que se incluyen varios elementos de énfasis (remarcado, itálico, grande, etc.); modelos de línea discontinua y tabuladores.

- ***Soporte para entradas del usuario.*** WML soporta muchos elementos que solicita el usuario, los cuales pueden ser combinados en una o más cartas. Todos los requerimientos de entrada al usuario son hechos en términos abstractos, concediéndole al agente de usuario la libertad de optimizar las características de un dispositivo particular. Para ello, WML posee un pequeño conjunto de controles de entrada, como por ejemplo el control de entrada de texto, el cual, soporta la entrada de texto y passwords. En este caso, los campos de entrada de texto pueden ser enmascarados para prevenir que el usuario final ingrese tipos de carácter incorrecto. WML también soporta validaciones en el lado del cliente que le permiten al autor invocar scripts en el momento adecuado para comprobar la entrada de usuarios. WML también incluye un control de selección de opciones que permiten a los autores presentar al usuario una lista de opciones (puede escoger una sola opción o varias) que pueden ser: configurar datos, navegar entre cartas o invocar scripts, es decir, incluye selecciones de opción múltiple y sencilla. WML también incluye controles de invocación de tareas, que cuando se activan controlan el inicio del navegador, o la historia de la navegación a través de enlaces a otras cartas o scripts, o la salida de la carta actual de la pila historial. El agente de usuario es libre de escoger cómo presentar estos controles, pudiendo por ejemplo ligarlos a teclas físicas en el dispositivo, dibujar botones de control en una región particular de la pantalla (o entre el texto), ligarlos a comandos de voz, etc.

- ***Navegación y pila historial.*** WML permite muchos mecanismos de navegación usando URLs. También expone un mecanismo historial de primera clase tal como una pila, que incluye historial de navegación con *hiperlinks* HTML y elementos de navegación entre cartas.

- ***Soporte internacional.*** El conjunto de caracteres para documentos WML es el Conjunto Universal de Caracteres de ISO/IEC-10646. Actualmente este conjunto de caracteres es idéntico al Unicode 2.0.

- ***Independencia de interfaces de máquina.*** Entre sus especificaciones de diseño y presentación, WML permite que los proveedores controlen el diseño de las MMIs (Man Machine Interface) para sus particulares productos.

- ***Optimización de banda estrecha.*** WML posee una variedad de tecnologías que optimizan la comunicación entre dispositivos de banda estrecha, lo que incluye la habilidad para precisar interacciones de múltiples usuarios (como un conjunto de cartas) sobre una sola red (como una baraja). También incluye una variedad de facilidades de manejo de estado que minimizan la necesidad de requerimientos a

los servidores origen. WML también incluye otros mecanismos que ayudan a mejorar la respuesta en el tiempo y minimizan la cantidad de intercambio de datos sobre el aire. Por ejemplo, WML permite al autor parametrizar (o pasar variables) al contexto siguiente sin alterar las URLs, mejorando así los aciertos del cliente en la memoria caché (*hits*).

- ***Manejo del contexto*.** Cada control de entrada de WML puede introducir variables, y el estado de éstas puede ser usado para modificar el contenido de las tarjetas parametrizadas sin tener que establecer comunicación con el servidor. Adicionalmente, el tiempo de vida de las variables puede ser tan largo como una sencilla baraja, y puede ser compartido por múltiples barajas sin tener que usar el servidor para salvar el estado intermedio de las barajas cuando fueron invocadas

III.2 Tipos de datos en el núcleo de WML

Este apartado describe los tipos de datos que acepta el núcleo de WML. Estos tipos de datos se utilizan en las descripciones de elementos de WML a lo largo de todo el capítulo.

III.2.1 Tipos de caracteres

Todos los tipos de caracteres de WML se definen en términos de tipos de datos de XML. Los tipos de datos de caracteres se resumen en la tabla III.1

TIPO DE DATOS	EXPLICACIÓN
CDATA	Texto que puede contener caracteres alfanuméricos. *CDATA* sólo puede desempeñar la función de valor de un atributo. Ejemplos: "$ (*value*)" name="*value*"
PCDATA	CDATA analítico. Texto que puede contener caracteres alfanuméricos. Este texto también puede contener etiquetas. *PCDATA* se usa sólo entre elementos. Ejemplo: Texto escrito <*big*> EN MAYÚSCULAS </*big*>. (Texto escrito en mayúsculas)
NMTOKEN	Un testigo de nombre, que contiene cualquier combinación de números, letras y algunos caracteres de puntuación ".", "-", "_", y ":". Así, datos de este tipo pueden comenzar con signos de puntuación. Un NMTOKEN no puede usarse en referencias a variables ni en nombres de elementos. Ejemplos: "texto" _card1 a.name.token
Id	Identificador del elemento, que además es único. Ejemplo: <card id= "card1"/>

Tabla III.1 *Tipos de Caracteres*

III.2.2 Longitud

El tipo *%length* puede ser especificado bien como un entero que represente el número de píxeles de la pantalla, o bien como un porcentaje de espacio horizontal o vertical disponible. De este modo, el valor “50” significa 50 píxeles. Para la anchura, el valor “50%” significa la mitad del espacio horizontal disponible. Y para la altura, el valor “50%” significa la mitad de espacio vertical disponible.

El valor entero consiste en uno o más dígitos decimales ([0-9]) seguido por un carácter porcentual opcional (%). El tipo *length* sólo se emplea como valor de un atributo.

Nombre	Tipo	Utilización
%length	CDATA	[0-9] para píxeles o [0-9] + % para porcentaje de tamaño

Tabla III.2 *Tipo Longitud*

Ejemplo
Los valores de los atributos *hspace* y *vspace* son *%length* :

```
<img src="bitmaps/moon.wbmp" hspace="1" vspace="1"/>
<img src="bitmaps/moon.wbmp" hspace="2%" vspace="2%"/>
```

III.2.3 Vdata

El tipo *%vdata* representa una cadena de caracteres (*string*) que puede contener referencias a variables. Este tipo se usa sólo en valores de atributos.

Nombre	Tipo	Utilización
%vdata	CDATA	Valor de atributo que puede contener referencias a variables.

Tabla III.3 *Tipo Vdata*

Ejemplo

```
<card id="card1" title="$(showme)">
```

III.2.4 Flow, inline y layout

El tipo *%flow* (flujo) representa la información a "nivel de carta" y el tipo *%inline* (en línea) representa a "nivel de texto". En general, el %flow se utiliza en cualquier lugar en el que se puede incluir una marca general. El tipo %inline indica las áreas donde se maneja texto o referencias a variables.

Nombre	Tipo	Utilización
%layout	br	Disposición del texto, tales como interrupciones de renglón.
%inline	%text %layout	Indica las áreas donde se maneja sólo texto o %layout referencias a variables.
%flow	%inline anchor a table img	Cubre elementos al nivel de carta, tales como texto o imágenes.

Tabla III.4 *Tipos Layout, Inline, Flow*

Ejemplo
Los tipos de datos %flow y %inline se usan para texto general que puede tener atributos de formato tales como itálica, subrayado y negrita.

```
<em>
Una línea enfatizada
<big>
Una línea grande y enfatizada.
</big>
</em>
Una línea sin formato de texto.
```

III.2.5 Text

El tipo *%text* (texto) puede incluir las siguientes entidades:

Nombre	Tipo	Ubicación
%text	#PCDATA %emph	Indica texto que contiene formato

Tabla III.5 *Tipo Text*

Ejemplo

```
Una línea con texto plano.
<em>
<strong>
Una línea con fuerte enfatizado.
</strong>
</em>
```

III.2.6 Href

El tipo %href se refiere tanto a un identificador de recurso uniforme (URI, Uniform Resource Identifier) absoluto como a uno relativo.

Nombre	Tipo	Utilización
%href	%vdata	URI, URL (localizador de recursos uniforme), o de diseño de nudos de hipertexto. Puede contener referencias a variables.

Tabla III.6 *Tipo Href*

Ejemplo

```
<go href="http://wapforum.org/"/>
<go href="file:///d:\dir\file.wml"/>
<go href="app.wml"/>
```

Los valores de los eventos intrínsecos son URL:

```
<card onenterforward="#card2"/>
```

El atributo *src* de un elemento *img* es un URL:

```
<img src="earth.wbmp"/>
```

III.2.7 Boolean

El tipo %boolean se refiere a valores lógicos de verdadero o falso.

Nombre	Tipo	Utilización
%boolean	true | false	Valores lógicos de verdadero o falso.

Tabla III.7 *Tipo Boolean*

Ejemplo

```
<card newcontext="true"/>
<do optional="true" type="accept"/>
```

III.2.8 Number

El tipo %number (número) representa un valor entero mayor o igual a cero.

Nombre	Tipo	Utilización
%number	NMTOKEN	Un numero entre [0-9].

Tabla III.8 *Tipo Number*

Ejemplo

```
<select tabindex="2"/>
<input name="setvar" size="4" maxlength="20" tabindex="3"/>
```

III.2.9 Emphasis

El tipo %emph representa etiquetas de formato de texto descritas en la tabla siguiente:

Nombre	Tipo	Utilización
%emph	em | strong | b | i | u | big | small	Formateo de texto, por ejemplo itálica o subrayado.

Tabla III.9 *Tipo Emphasis*

Ejemplo

```
<em>
        Una línea enfatizada
        <big>
                Una línea grande y enfatizada.
        </big>
</em>
```

III.2.10 Entidades de caracteres y caracteres especiales

Un codificador de caracteres dado puede no ser capaz de expresar todos los caracteres del juego de caracteres del documento. Para tal codificador, o cuando las características del aparato no permitan a los usuarios entrar directamente algunos caracteres, se puede usar entidades de carácter, que son un mecanismo codificador independiente para introducir cualquier carácter desde el juego de caracteres de documento.

WML posee los siguientes formatos de caracteres:

- Caracteres de nombre, tales como & y <
- Caracteres numérico decimales, tales como &#123
- Caracteres numérico hexadecimales, tales como #x12;

La siguiente tabla ilustra las siete entidades de caracteres de nombre que son especialmente importantes en el procesador de WML

Entidad	Notación	Explicación
quot	"	Comillas
amp	& #38;	&
Apos	&#39	Apóstrofe
lt	&#60;	<
gt	&#62;	>
nbsp		Espacio sin separación

Entidad	Notación	Explicación
shy		guión opcional

Tabla III.10 *Entidades de caracteres*

Nota: El punto y coma (;) es parte de la secuencia de escape para un carácter especial.

Ejemplo
Para incluir un carácter especial, simplemente se usa la notación descrita en la tabla anterior. Por ejemplo, el siguiente código incluye un carácter "menor que" (<) en la forma de escape < .

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="Card_1">
<p>
        Numerically 5 &#60; 10
</p>
</card>
</wml>
```

Esta baraja genera la siguiente interfaz de usuario:

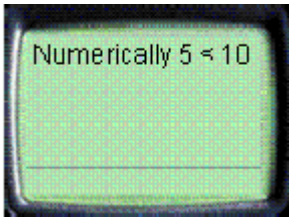


Figura III.2 *Uso de caracteres especiales.*

III.2.11 Sintaxis WML

WML hereda la mayoría de sus construcciones sintácticas de XML.

III.2.11.1 Entidades

El texto WML puede contener entidades de caracteres numéricas o de nombre las cuales especifican caracteres específicos en el juego de caracteres de documento. Estas entidades especiales se utilizan para especificar caracteres que deben ser solventados en WML o que pueden ser difíciles de introducir en un editor de texto. Por ejemplo, el ampersand (&) se representa por la entidad de nombre &. Todas las entidades empiezan con un ampersand y terminan con un punto y coma.

WML es un lenguaje XML. Esto implica que los caracteres ampersand y "menor que" deben ser simulados cuando se utilicen en datos de texto, es decir, esos caracteres

pueden aparecer en su forma literal sólo cuando se utilicen como delimitadores de marca, dentro de un comentario, etc.

Ejemplo

```
<card id="card1">
<p>
    Ampersand = &amp; <br/>
    Quote = &quot; <br/>
    Less than = &lt; <br/>
</p>
</card>
```

La baraja genera la siguiente interfaz de usuario en la terminal de destino:

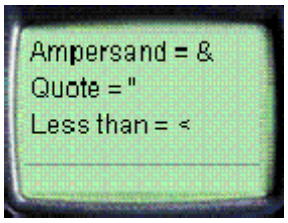


Figura III.3 *Ejemplo de entidades.*

III.2.11.2 Etiquetas

Una etiqueta (tag) es un descriptor de elementos del lenguaje. Una etiqueta describe un elemento y contiene un nombre de tipo de elemento y un identificador único. Una etiqueta podría también incluir atributos describiendo otras propiedades.

El código WML consiste en un conjunto de etiquetas, cada una de ellas encerrada en un par de corchetes, < y >.

<tag> Comienzo de elemento. La etiqueta de comienzo puede contener atributos.

</tag> Etiqueta que indica el fin de un elemento.

<tag/> Indica un elemento vacío, por ejemplo <br/>, que indica una salto de línea.

III.2.11.3 Elementos

Los elementos especifican todas las marcas e información estructural de una baraja de WML. Los elementos pueden contener una etiqueta de comienzo, contenido, otros elementos y una etiqueta de final. Los elementos tienen una o dos estructuras:

<tag> contenido </tag> ó

<tag/>

Los elementos que incluyen contenido u otros elementos están identificados por una etiqueta de comienzo <tag> y una etiqueta de finalización </tag>. Una etiqueta de elemento vacío <tag/> identifica elementos sin contenido.

III.2.11.4 Atributos

Los atributos WML especifican información adicional para un elemento. Los atributos se especifican siempre en la etiqueta de comienzo de un elemento. Por ejemplo,

```
<tag attr="value"/>
```

Nota: Los nombres de los atributos deber estar en minúsculas.

WML requiere que todos los valores del atributo sean entrecomillados usando tanto dobles comillas (") como comilla simple ('). Las comillas simples pueden incluirse dentro del valor de atributo cuando el valor se delimite por comillas dobles y viceversa. Las entidades de caracteres pueden incluirse en un valor de un atributo.

Algunos atributos son obligatorios en las descripciones de elemento mostrados posteriormente en este capítulo, dichos elementos están resaltados en negrita. Por ejemplo, el elemento *go* requiere el atributo *href* :

```
<go href="http://www.acme.com"/>
```

III.2.11.5 Comentarios

Los comentarios en WML siguen el estilo de comentario de XML y tienen la siguiente sintaxis:

```
<!-- comentario  -->
```

Los comentarios son una ayuda para el programador del documento WML y no se muestran al usuario en la terminal de destino. Hay que señalar que los comentarios WML no pueden ser almacenados.

III.2.11.6 Variables

Los parámetros pueden ser incluidos en las cartas y barajas de WML usando variables. Para sustituir una variable (por su valor) dentro de una carta o baraja, se utilizan las siguientes sintaxis:

```
$identificador
$(identificador)
$(identificador:conversión)
```

Se requieren paréntesis si el espacio en blanco no indica el final de una variable, es decir si el nombre de la variable lo conforman varias palabras separadas por espacios en

blanco. La sintaxis de variable tiene la mayor prioridad en WML, esto es, en cualquier lugar donde la sintaxis de variable sea legítima, un carácter “sin escapar” '*$*' conlleva a una sustitución de una variable. Las referencias de variables son legítimas en cualquier *PCDATA* y en cualquier valor de atributo que sea de tipo entero *vdata*.

Ejemplo
Una secuencia de dos signos de dólar ($$) representa un carácter de signo de dólar sencillo.

El código WML podría tomar la siguiente forma:

```
Este es un $$ carácter.
El valor es $(cantidad)$$.
```

En la terminal del usuario, esto se mostraría como:

```
Este es un $ carácter.
El valor es 5000$.
```

III.2.11.7 Sensibilidad de formato de letra

El lenguaje XML es un lenguaje sensible al formato de letra (mayúsculas-minúsculas) y WML ha heredado dicha característica. Cuando se analiza una baraja de WML no se realiza aceptando el doble formato como uno, es decir reconoce como diferentes lo que está en mayúsculas y lo que está en minúsculas. Esto implica que todas las etiquetas, atributos y contenidos de WML son sensibles al formato de letra. Además cualquier valor de atributo enumerado es sensible al formato.

Ejemplo
Los siguientes valores de atributo son todos diferentes:

```
id="Card1"
id="card1"
id="CARD1"
```

III.3 PRIMEROS PASOS EN WML

Como se vio WML es un lenguaje de etiquetas o tags, es decir, se compone de unas marcas para definir la estructura. Estas etiquetas poseen una serie de atributos, permitiendo manejar las etiquetas y modificar la forma en que se muestra la información que contiene, por ejemplo el tag <card> posee el atributo id y title, los atributos se definen dentro de la etiqueta, <card id="identificador" title= "titulo">.

Existen dos tipos de definición de tags, por un lado aquellos que tienen un principio y un final, por ejemplo:

```
<p>
..... párrafo
</p>
```

Por otro lado están aquellos que en sí misma es una única etiqueta como <br/>, donde se escribe la "/" al final del tag.

Para comenzar a programar código WML necesitaremos un editor de textos como notepad. Existen editores que ayudan en la programación añadiendo código o ayudando a hacerlo.

Para visualizar el resultado se utiliza un emulador, cada marca suele tener su propio emulador.

Cuando hayas cargado una página en el emulador o la terminal, se queda en memoria caché, por lo que si realizas alguna modificación necesitarás *borrar la caché* para poder visualizar el cambio.

Los archivos con código WML deben tener extensión wml, por ejemplo index.wml, primero.wml, altas.wml, etc.

Publicación en Internet
Para publicarla en Internet, prácticamente cualquier servidor de espacio servirá. Sólo tenemos que configurar en el servidor (o pedir al administrador del sistema que nos lo configure) los MIME types:

	MIME type:	EXTENSIÓN
Para el código WML	text/vnd.wap.wml	.wml
Para las imágenes wml (extensión wbmp):	image/vnd.wap.wbmp	.wbmp
Para el WML Script:	text/vnd.wap.wmlscript	.wmls
Para el WML compilado:	application/vnd.wap.wmlc	.wmlc
Para el WML Script compilado:	application/vnd.wap.wmlscriptc	.wmlsc

Pero si sólo usamos los .wml y .wbmp sólo tendremos que configurar el servidor para los mismos.

Y si queremos que el index.wml sea el index por defecto (para acceder directamente a: http://www.wmlclub.com/wap/ y no tener que escribir: http://www.wmlclub.com/wap/index.wml) tendremos que configurar el index.wml como página por defecto (cada servidor se configura de una forma distinta).

Cartas y Barajas
Todo el código WML se organiza dentro de una colección de cartas y barajas. Las cartas especifican una o más unidades de interacción con el usuario, por ejemplo un menú de opciones, una pantalla de texto o un campo de entrada de texto. Lógicamente, un usuario navega a través de una serie de cartas de WML, revisando el contenido de cada una de ellas, introduciendo la información requerida, realizando elecciones y moviéndose a otra carta.

Las cartas están agrupadas dentro de barajas. Una baraja es la unidad más pequeña de WML que un servidor puede enviar al terminal del usuario.

La siguiente figura ilustra el significado de carta y baraja:

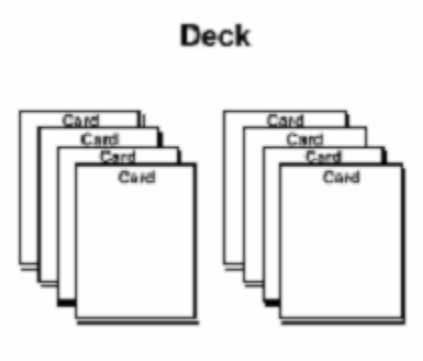


Figura III.4 *Cartas (card) y barajas (deck) en el lenguaje WML.*

Primer ejemplo de programación en WML

Nuestro primer ejemplo de WML introduce una baraja WML simple conteniendo dos cartas. Cuando el usuario presiona la tecla ACCEPT (Aceptar), rotulada "Next"(Siguiente), el usuario navega a la segunda carta de la baraja y muestra su contenido.

```
<?xml version="1.0"?>                                          <!-- 1 -->
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"            <!-- 2 -->
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>                                                          <!-- 3 -->
<card id="First_Card">                                         <!-- 4 -->
<do type="accept" label="Next">                                <!-- 5 -->
<go href="#Second_Card"/>                                      <!-- 6 -->
</do>                                                          <!-- 7 -->
<p>                                                            <!-- 8 -->
Select <b>Next</b> to display the next card                    <!-- 9 -->
</p>                                                           <!-- 10 -->
</card>                                                        <!-- 11 -->
<card id="Second_Card">                                        <!-- 12 -->
<p>                                                            <!-- 13 -->
This card contains the following                               <!-- 14 -->
</p>                                                           <!-- 15 -->
</card>                                                        <!-- 16 -->
</wml>                                                         <!-- 17 -->
```

Esta baraja genera una interfaz de usuario como la siguiente:

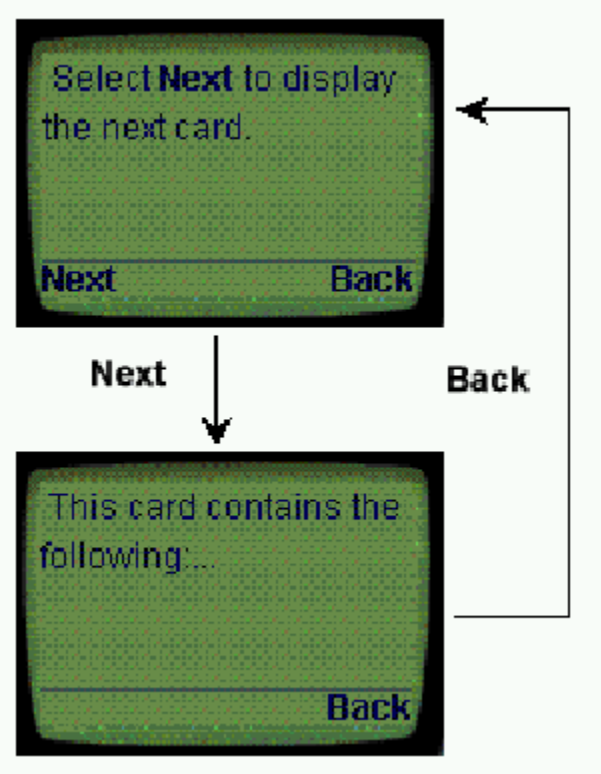


Figura III.5 *Baraja que contiene dos cartas.*

A continuación explicamos de manera detallada este ejemplo para comprender los conceptos vistos al principio:

1. Las dos primeras líneas definen la cabecera del documento que identifica el subjuego XML. Esta cabecera debe incluirse al comienzo de cada baraja de WML, es decir, antes de cada etiqueta <wml>.
2. La tercera línea define el encabezado de la baraja WML. Todas las barajas WML deben empezar con una etiqueta <wml> y terminar con una etiqueta </wml>.
3. La cuarta línea de la baraja especifica el encabezado e identificador de la primera carta. Análogamente a las barajas, las cartas también requieren etiquetas de comienzo y de finalización, <card> y </card>. La mayoría de los elementos de WML permiten especificar atributos. Los atributos se introducen en la forma atribute=value, donde *atribute* es el nombre de atributo y *value* es un valor alfabético o numérico que toma el atributo.
4. La quinta línea define una acción, la cual especifica lo que la terminal del usuario debiera hacer al presionar una determinada tecla de función. El atributo *type* identifica la tecla (*accept*) y el atributo *label* asigna un rótulo (*Siguiente*) a la tecla especificada.
5. La sexta línea indica una acción de relación entre cartas, páginas webs, etc. El atributo *href* indica el destino URI donde dicha acción nos va a llevar (en este ejemplo éste será la carta “Second-Card”).
6. La séptima línea indica el cierre de la definición de la acción a realizar cuando se presione una tecla.

7. La octava línea especifica que se va a insertar el contenido de la carta. El objetivo principal de cualquier carta es presentar información en la terminal.
8. En la novena línea se encuentra el texto que aparecerá en la terminal diferenciado entre texto normal y el texto con formato que se encuentra entre los *tags* de formato negrita.
9. En la décima línea se define el final del contenido visible de la carta.
10. En la undécima línea indica el final de la primera carta.
11. La duodécima línea define el inicio de una nueva carta para la baraja en cuestión con su misma estructura que la carta anterior con su contenido particular, el cual se observará al navegar entre cartas.
12. En la línea catorce se incluye el texto que aparecerá en la segunda carta, cabe señalar que a diferencia de la novena línea aparece texto plano sin formato.
13. Para culminar, en la línea diecisiete se encuentra el final de nuestra baraja como se mencionó al principio del ejemplo.

III.3.1 Paso a Paso

Para comprender mejor el manejo de WML, en esta parte aprenderemos cómo realizar páginas de WML [WMLClub].

Éste es el código que siempre debe aparecer al principio de una página WML; mediante él se informa que pertenece a código wml.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
```

Tras estas etiquetas se escribe el código que será en realidad lo que se visualice en nuestra terminal.

La siguiente etiqueta es <wml>, donde recoge el código de la página.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
" http://www.wapforum.org/DTD/wml_1.1.xml">

<wml>
..... código
</wml>
```

Empezaremos por ver cómo se pueden definir varias cartas dentro de una página:

```
< ?xml version= « 1.0 » ?>
< !DOCTYPE <WML> PUBLIC « -//WAPFORUM//DTD <WML> 1.1//EN
« http ://www.wapforum.org/DTD/wml_1.1.xml »>
<wml>
<card id=”carta1” title=”titulo decarta1”>
</card>
<card id=”carta2” title=”titulo decarta2”>
</card>
```

```
<card id=”carta3” title=”titulo decarta3”>
</card>
</wml>
```

La idea de crear las cartas es intentar dividir la versión en varias subpáginas para presentarla de forma ordenada en la pantalla de la terminal.

Para introducir texto en la página se realiza mediante la etiqueta , el texto irá contenido entre el *tag* de comienzo ( ) y el de fin ( ).

```
<?xml version=”1.0”?>
<!DOCTYPE wml PUBLIC “-//WAPFORUM//DTD WML
1.1//EN” “ http://www.wapforum.org/DTD/wml_1.1.xml”>
<wml>
<card id=”cartauno” title=”titulo de carta”>
<p>Este es el texto de mi primera carta</p>
</card>
</wml>
```

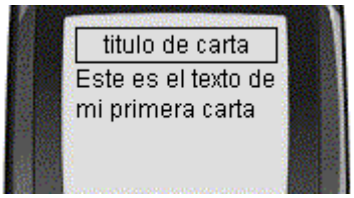


Figura III.6 *Ejemplo etiqueta *

Cuando se crean varias cartas, únicamente se pueden visualizar una de ellas, por lo que es necesario establecer un mecanismo para pasar de una carta a otra.

En el siguiente ejemplo se ha utilizado la etiqueta <a ...>:
texto visible para el usuario
La tecla de la terminal que permite ejecutar esta opción es la de OK.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML
1.1//EN" " http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="cartauno" title="titulo de carta 1">
<p>Este es el texto de mi primera carta <br/>
<a href="#cartados"> ir a carta dos </a></p>
</card>
<card id="cartados" title="titulo de carta 2">
<p> Este es el texto de mi segunda carta <br/>
<a href="#cartauno">ir a carta uno</a></p>
</card>
</wml>
```

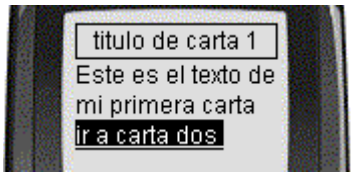


Figura III.7 *Primera carta de una baraja*

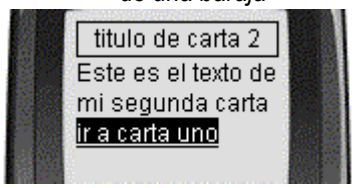


Figura III.8 *Segunda carta de una baraja*

Una vez cargada la página, mediante la tecla OK (Yes) se permite enlazar a la carta dos para poder ver el contenido de ésta. Como se puede observar todo el contenido tanto de la carta cartauno como de cartados se encuentra contenido dentro de las etiquetas y . La etiqueta <br/> permite realizar un salto de línea en pantalla.

En esta ocasión, en lugar de hacer un enlace a una carta de la misma página se realizará una llamada a una página diferente. El cambio se encuentra en que en lugar de colocar <a href="#carta"> se cambia por <a href= "pagina.wml">.

Página del programa lec5.wml

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-
//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="deck1" title="Deck 1">
<p>Este es el texto de mi primera carta
Pagina 1
<br/>
<a href="lec5_2.wml"> ir a pagina 2 </a>
</p>
</card>
</wml>
```

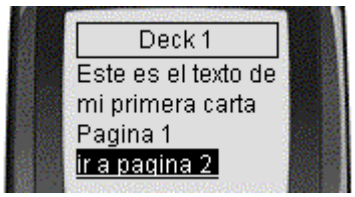


Figura III.9 *Primera carta de una baraja*

En el momento de pulsar la tecla OK (Yes) se activaría el enlace a la página siguiente, que en este caso tal y como está declarada será lec5_2.wml

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-
//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="deck2" title="Deck 2">
<p>texto de la carta 1 Pagina 2
<br/>
<a href="lec5.wml"> ir a pagina anterior </a>
</p>
</card>
</wml>
```

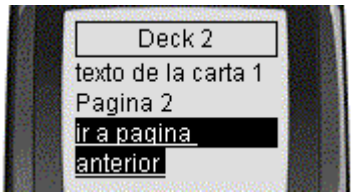


Figura III.10 *Segunda carta en una página diferente*

Nuevamente, si se pulsa el botón OK se ejecutará el enlace a la página lec5.wml.

Existe, además de la etiqueta <a href="…"> otro método para poder pasar de una carta a otra o bien a una página diferente. Se puede realizar mediante el *tag* <go ...../>.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-
//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="cartauno" title="Envio con go">
<do type="accept" label="Ir a Carta 2">
<go href="#cartados"/>
</do>
<p>Pulse 3 seg. la tecla OK
<br/>
</p>
</card>
<card id="cartados" title="titulo de carta 2">
<p> Bienvenido a la carta 2
<br/>
</p>
</card>
</wml>
```

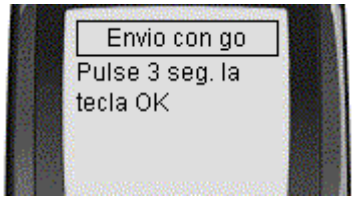


Figura III.11 *Primera carta en una baraja*

Figura III.12 *Tarea go*

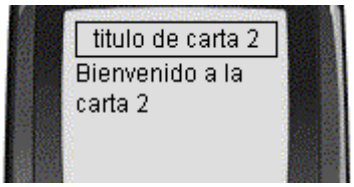


Figura III.13 *Segunda carta en una baraja*

A diferencia de <a href...> donde únicamente con pulsar tecla OK se realiza el cambio, cuando se utiliza go, se debe pulsar al menos 3 segundos, tras ello aparece una ventana con las opciones donde te puedes dirigir, en este caso existen las opciones de la carta 2, la página previa o volver al menú.

Mediante <do type="accept" label="Ir a Carta 2"> se permite el visualizar el menú de la figura III.12 y <go href="#cartados"/> define la página o carta que se visualizará a continuación.

Además de usar enlaces para dirigirnos a otras cartas de una baraja, podemos redirigir a una dirección URL.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML
1.1//EN" "http://www.wapforum.org/DTD/wml_1.1.xml ">
<wml>
<card id="" title="Enlace cocotero.com">
<p align="center">Enlace a una direccion URL<br/>
<a href="http://www.cocotero.com/wap">
cocotero</a>
</p>
</card>
</wml>
```

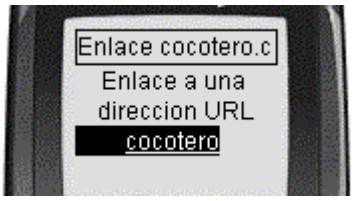


Figura III.14 *Enlace a una URL*

Así mismo se puede hacer un enlace a una carta que pertenece a una determinada página.

En el siguiente ejemplo Enlace a carta 2 de la baraja lec8_2.wml nos envía a la carta de la baraja deseada, para esto en primer lugar se define la baraja a la cual se desea enlazar y luego a la carta en cuestión.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML
1.1//EN" " http://www.wapforum.org/DTD/wml_1.1.xml ">
<wml>
<card id="" title="Enlace">
<p align="center"><a href="lec8_2.wml#c2">
Enlace a carta 2 de la<br/>
pagina lec8_2.wml</a><br/>
</p>
</card>
</wml>
```

Figura III.15 *Enlace a una carta de otra baraja*

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML
1.1//EN" " http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="c1" title="Enlace a carta">
<p align="center">Enlace carta<br/>
<a href="lec8_2.wml#c2">Carta 2</a>
</p>
</card>
<card id="c2" title="Carta 2">
<p align="center">
Carta perteneciente a<br/>
lec8_2.wml
</p>
</card>
</wml>
```

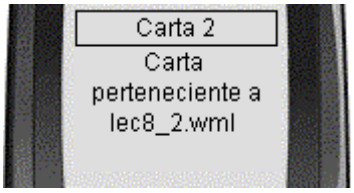


Figura III.16 *Carta enlazada*

III.3.2 Imágenes

En los dispositivos móviles se permite la visualización de imágenes, pero éstas deben estar en formato WBMP, en lugar de las utilizadas en HTML que pueden ser gif, jpg o bmp.

También hay que indicar que entre las terminales móviles existen diferencias en cuanto al tipo de imágenes que soportan, por lo que se recomienda conocer el tipo de imágenes soportadas por el móvil con el que se va a trabajar. En el caso de simuladores no existen graves problemas con imágenes.

Hay que tener en cuenta el tamaño de la página tras insertar una imagen, ya que en una terminal real puede dar error, porque se ha sobrepasado el tamaño, y en el simulador puede observarse perfectamente.

Hay distintos medios para producir una imagen WBMP, una de ellas es crear imágenes propias, para ello se puede utilizar distintos programas que se encuentran en la web como

el programa Generador WBMP (http://www.amazingvb.latinaddress.com/wbmp) realizado por un miembro de la comunidad de WmlClub. Además este programa permite convertir de un formato no soportado como gif, jpg, etc., a tipo wbmp.

Tras la generación de la imagen nos queda introducirla en la página asignada para ello.

En el siguiente ejemplo podemos ver el código necesario para hacerlo.

El atributo alt es obligatorio, es un texto alternativo a la imagen, ya que si ésta no se puede visualizar correctamente o el navegador no permite su visualización, se imprimirá éste.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" "
http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card title="IMAGEN" id="ejemplo">
<p><img src="logo.wbmp" alt="logo" /></p>
</card>
</wml>
```

III.3.3 Variables

Una de las grandes diferencias entre el wml y el html es que con el wml se pueden definir variables en las cartas, asignarles valores y presentarlos en la pantalla, incluso utilizar las variables en expresiones (programa.pl?f=$(mivar), etc.)

La mayor ventaja de todo esto es que se puede conservar información en el paso de una carta a otra y así poder dividir el contenido en varios pasos (que en pantallas tan enanas, se agradece).

Las variables son cadenas de texto (*case sensitive* -que distingue mayúsculas y minúsculas-, como siempre en el wml) a la que se le asigna un valor (secuencia de caracteres) o ningún valor.

El nombre de la variable puede empezar por el guión bajo: "_" o una letra US-ASCII, seguida de una o más letras US-ASCII, números o el guión bajo.

III.3.4 Crear Variables y Asignarles Valores

Varias formas posibles:

setvar

```
<setvar name="mivar1" value="Juan"/>
```

Juan es el valor de la variable mivar1. Con *setvar* se crea la variable y a la vez se le asigna un valor.

input
Con *input* se puede crear (declarar) la variable, asignarle un valor y también devolverle el valor original.

Input se utiliza para la entrada de datos y esos datos (valores) se asignarán a una variable.

select
Con *select* se puede crear (declarar) la variable, asignarle un valor y también devolverle el valor original.

Select permite seleccionar al usuario una o más valores entre una lista de opciones que se asignarán a una variable.

postfield
Con *postfield* se puede crear (declarar) la variable, asignarle un valor y también devolverle el valor original.

III.3.5 Referenciar variables

Podemos incluir el valor de una variable dentro de un documento wml, tanto para que aparezca en la pantalla como para que se envíe a un programa.

Hay tres formas de referenciar la variable:
$nombrevariable
se utiliza cuando no hay ambigüedad con el nombre de la variable dentro del contexto.

$(nombrevariable)
Cuando puede existir ambigüedad con el nombre de la variable dentro del contexto.

$(nombrevariable:conversión) Se explica más adelante.

Como el wml se reserva el uso del signo del dólar, si queremos que aparezca en la pantalla este signo, hay que escribirlo dos veces ($$). Por ejemplo tenemos esta variable: $moneda y queremos presentar su valor en la pantalla, así: $15, escribiremos:
Saldo actual: $$$moneda

CONVERSIÓN AL FORMATO ESCAPE $(nombrevariable:conversión)
En su día se acordó una sustitución de algunos caracteres propios de los URLs para que el servidor no los confundiera (reglas del formato escape (RFC2396)).
Estas reglas nos facilitan un mecanismo para poder incluir en una línea URL, esos caracteres. Aunque estas reglas fueron creadas para referenciar URLs, se pueden aplicar las mismas reglas para referenciar variables.
Ejemplo:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="carta1" title="WMLCLUB">
<p>Bienvenido
```

```
<br/>
Teclee un texto con caracteres especiales:
<br/></p>
<p><input type="text" name="texto"/></p>
<do type="accept">
<go href="#carta2"/>
</do>
</card>
<card id="carta2" title="WMLCLUB">
<p>
Este es el texto introducido, en formato escape:
<br/>
$(texto:e)
<br/>
Este es el texto introducido, normal:
<br/>
$(texto)
</p>
</card>
</wml>
```

Cuando se introduce el valor de una variable dentro de una baraja, se puede definir el formato (escape, unescape o no escape) así:
$(var:e) $(var:E) $(var:escape) (cualquiera de los tres traduce al formato escape) $(var:u) $(var:U) $(var:unesc) (cualquiera de los tres traduce del formato escape a texto US-ASCII)
$(var:n) $(var:N) $(var:noesc) (cualquiera de los tres hace que el valor de la variable no se traduzca al formato escape)

El WML siempre aplica el formato escape cuando se trata de atributos que trabajan con URLs. Por eso muchas veces se puede asumir que el WML convertirá al formato escape cuando lo tiene que hacer. De todas formas, conviene incluirlo si tenemos mínimas dudas de lo que ocurrirá.

Aquí se presenta una lista de los caracteres de escape (todos empiezan por %)

;	%3b
/	%2f
?	%3f
:	%3a
@	%40
&	%26
=	%3d
+	%2b
$	%24
,	%2c
espacio	%20
{	%7b
}	%7d
|	%7c
\	%55c

^	%5e
[	%5b
]	%5d
`	%27
<	%3c
>	%3e
#	%23

III.3.6 Las variables y las tareas <noop>, <prev>, <refresh> y <go>

<noop/>
No hace nada. Se utiliza para desactivar eventos en el nivel de la baraja.

<prev/>
prev vuelve al anterior URL.
Si dentro de prev hay un elemento setvar, como en el ejemplo, se procesa antes de que se ejecute el prev.

```
<prev>
<setvar name="mivar" value="contenido"/>
</prev>
```

<refresh/>
refresh refresca los contenidos visibles del navegador.
Si refresh contiene un setvar, como en el ejemplo, se procesa el setvar y después se procede al refresh.

```
<refresh>
<setvar name="mivar" value="contenido"/>
</refresh>
```

<go>
go lleva a otra URL u otra carta. Si dentro de go hay un elemento setvar, se procesa primero éste y después se ejecuta la tarea "go".

```
<go href="http://www.wmlclub.com/cgi-bin/programa.pl?x=$(mivar1)&y=$(mivar2)"
method="post">
<setvar name="mivar1" value="50"/>
<setvar name="mivar2" value="80"/>
</go>
```

En el siguiente ejemplo, veremos cómo se inicializa una variable y las diferentes formas de que se puede referenciar.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-
//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card title="Variables 1" id="ejemplo">
<!-- Inicialización de variable -->
<onevent type="onenterforward">
<refresh>
<setvar name="var1" value="valor var1"/>
</refresh>
</onevent>
<p>Variables:<br/>
lt;!-- Visualización -->
$var1<br/>
$(var1)<br/>
$(var1:e)</p>
</card>
</wml>
```

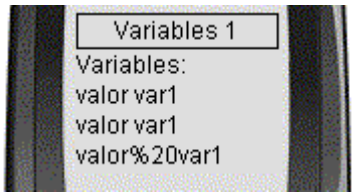
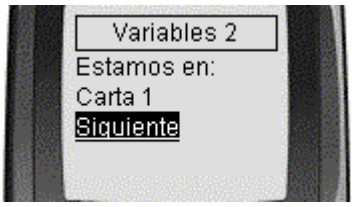
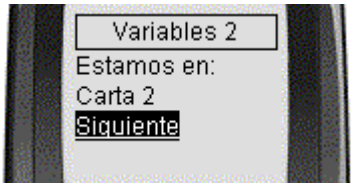


Figura III.17 *Variables*

Se puede utilizar la misma variable en diferentes cartas, variando su valor.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML
1.1//EN" "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card title="Variables 2" id="c1">
<onevent type="onenterforward">
<refresh>
<setvar name="var1" value="Carta 1"/>
</refresh>
</onevent>
<p>Estamos en:<br/>
$var1<br/>
<a title="Siguiente" href="#c2">Siguiente</a>
</p>
</card>
<card title="Variables 2" id="c2">
<onevent type="onenterforward">
<refresh>
<setvar name="var1" value="Carta 2"/>
</refresh>
</onevent>
<p>Estamos en:<br/>
$var1<br/>
<a title="Siguiente" href="#c1">Siguiente</a>
</p>
</card>
</wml>
```





Se pueden utilizar como variables los valores de los campos que se introducen mediante un *input*.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML
1.1//EN" "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card title="Variables 3" id="c1">
  <p align="center">
  <b>Introducir Nombre</b><br/>
  </p>
  <p align="left">
  <fieldset title="formInfo">
<input type="text" name="txtNombre" value="" format="*X"
size="8" maxlength="15" emptyok="false"/>
  </fieldset>
  <do type="accept" label="Enviar">
     <go href="#c2" method="get">
      <postfield name="nombre" value="$(txtNombre)"/>
    </go>
  </do>
  </p>
 </card>
<card title="Variables 3" id="c2">
  <p align="center">
  <b>Nombre introducido:</b><br/>
  $txtNombre
  </p>
   </card>

</wml>
```

III.3.7 Creación de Tablas

Aunque el *tag* de <table> se encuentra en las especificaciones del WML 1.1, existen algunos móviles que no lo implementan, como es el caso del Nokia 7110, en su lugar los datos son mostrados en una tabla de una columna.
Ejemplo de código para creación de tablas:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="Tabla" title="Tabla 1">
<p><table columns="3">
<tr><td><p>Elemento 1</p></td>
<td><p>Elemento 2</p></td>
<td><p>Elemento 3</p></td></tr>
</table></p>
</card>
</wml>
```

Referencias

[WMLClub] http://www.wmlclub.com
[WMLReference] http://www.forum.nokia.com
[WMLSpec] Wireless Markup Languaje Specification
http://www.wapforum.org

CAPÍTULO IV
Problemática en la Edición de Documentos WML.

IV.1 Tipo de equipo requerido.

Los sistemas WAP definen toda una arquitectura de funcionamiento que hace posible la transmisión de contenidos de texto hacia las terminales móviles para su posterior representación sobre el display del mismo. En este sentido, la adaptación de los contenidos accesibles por Internet, teniendo presente las limitaciones de representación sobre una terminal móvil, harían posible el acceso a Internet desde los mismos teléfonos móviles.

IV.1.2 Escenario de trabajo

El WWW[1] de Internet provee un modelo lógico flexible y muy potente el cual se representa en la figura IV.1. Un agente usuario envía peticiones a un servidor de origen, el servidor de origen responde a las peticiones de datos en uno de los formatos estándares conocido por el agente usuario (ej. HTML).

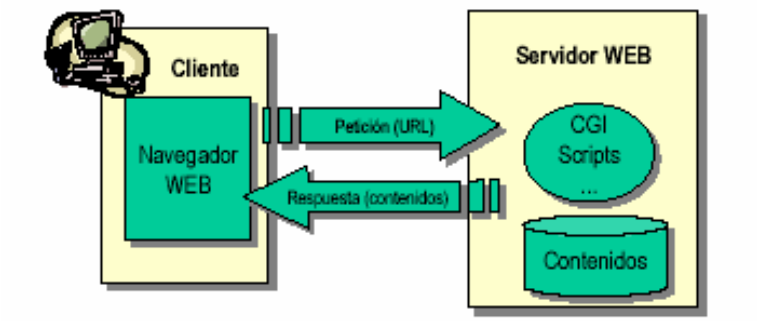


Figura IV.1 *Escenario de comunicaciones típico en un Entorno Internet-WEB*

Los estándares de WWW incluyen todos los mecanismos necesarios para construir un entorno de propósito general. El estándar de Internet conocido como *Uniform Resource Locators (URLs)* se usa para darle nombre a los recursos dentro del WWW. De manera adicional WWW define una gran variedad de formatos de contenido que son soportados por la mayoría de los browsers[2] (ej. HTML). Además un conjunto de protocolos estandarizados permiten a cualquier browser comunicarse con los servidores de origen.

En los sistemas WAP se utiliza un modelo basado en el modelo lógico de WWW, todo el contenido se especifica en formatos que son similares a los formatos estándar de Internet,

[1] WWW. World Wide Web
[2] Browser. Se conoce como navegador

la transportación del contenido se maneja por medio de los protocolos estándar del dominio WWW y un protocolo parecido al protocolo http en el dominio inalámbrico.

El escenario de comunicación entre el dispositivo WAP e Internet se comporta de la siguiente manera: primero, el dueño de un teléfono que cuenta con la tecnología WAP usa el micro-browser (un pariente menor del software de aplicación utilizado en las PCs para navegar en Internet, buscando páginas de la WWW) para solicitar una página Web específica. El pedido viaja como ondas radiofónicas hacia una torre de transmisión celular y es ruteado al servidor por la empresa de servicios inalámbricos.

Entre el cliente y el servidor está la "WAP *gateway*" (una combinación de programas y hardware que comunica dos tipos diferentes de redes), es decir, en pocas palabras el software que sirve como un filtro entre la red inalámbrica e Internet. La "WAP *gateway*" codifica el WML por lo que se puede transferir inalámbricamente, enviando el documento al usuario de teléfono celular, apareciendo éste en el pequeño display del aparato. Como se muestra en la figura IV.2

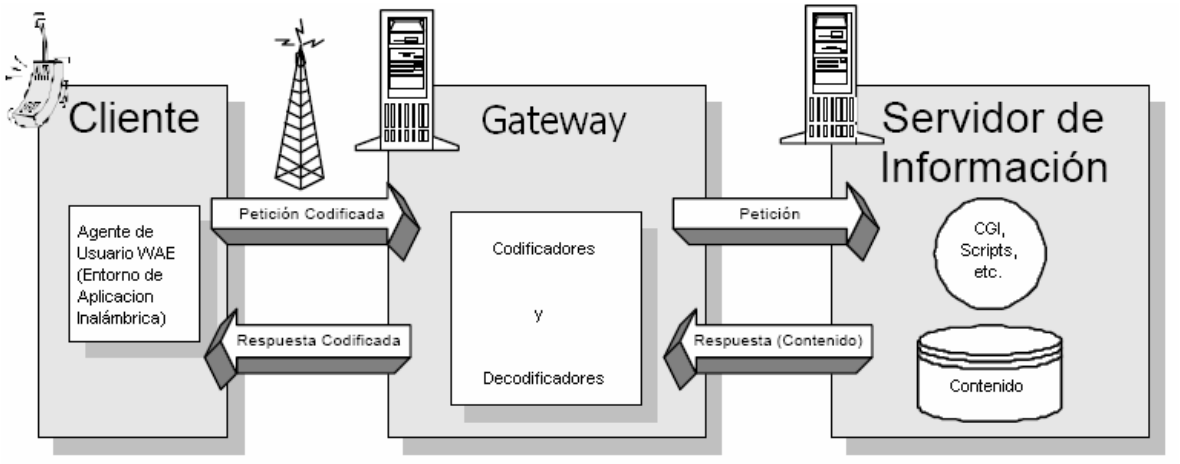


Figura IV.2 *Modelo Lógico de comunicación WAP*

El servidor de contenidos WML realiza las mismas funciones que un servidor de Internet, las páginas son creadas y almacenadas en los directorios de acceso desde el exterior a través de direcciones "http" lo mismo que en los servidores Web. El servidor WML se puede implementar en una gran variedad de plataformas para que sea compatible con los servidores WWW. Como ejemplos de estas plataformas tenemos a Unix (Linux) o Windows'NT, y para el servidor de contenidos WML: Apache, IIS, Xitami, etc. La utilización de una u otra plataforma, así como del servidor Web, tiene sus ventajas e inconvenientes que van más allá de la explicación que se pretende dar en este apartado.

Ahora bien, ¿cómo se comunica el equipo móvil con un servidor de origen? ¿Qué necesita tener el dispositivo para poder interpretar la información que envía el servidor? Para poder dar respuestas a estas interrogantes, debemos entender primero la capa del modelo WAP que se encarga de la presentación de las aplicaciones propias de los equipos móviles y nos referimos a la capa WAE.

IV.2 Visualización de la página desarrollada (Introducción a la capa WAE)

IV.2.1 Antecedentes

El protocolo de aplicaciones inalámbricas (Wireless Application Protocol WAP) es el resultado del trabajo continuo para definir una especificación amplia para el desarrollo de aplicaciones que operan en redes de comunicaciones inalámbricas.

El entorno de aplicación inalámbrica (WAE Wireless Application Environment) es parte del esfuerzo del WAP Forum para especificar un marco de aplicación para terminales inalámbricas como teléfonos móviles y PDA's, dicho marco se extiende y tiene influencia sobre otras tecnologías WAP, incluyendo WTP y WSP así como otras tecnologías de Internet como XML, URLs, Scripting y varios *media types*.

WAE es un entorno de aplicación de propósito general donde los operadores y los proveedores de servicio pueden construir sus aplicaciones y servicios para una gran variedad de plataformas inalámbricas.

WAE ha sido construido basándose fundamentalmente en la tecnología WWW con una optimización adicional para dispositivos y redes inalámbricas, tomando en cuenta sus limitantes como son pantallas pequeñas, limitadas facilidades para ingresar información al dispositivo, un pequeño ancho de banda, memoria limitada , CPU limitado, etc.

Como el rango de dispositivos conectándose al Internet ha crecido, con su incremento de capacidades en términos de interfaz de usuario, capacidades de presentación, etc, el núcleo de la tecnología Internet/WWW seguirá evolucionando para satisfacer los requerimientos de estos dispositivos y los subsecuentes.

IV.2.2 Metas y Requerimientos del WAE

IV.2.2.1 Metas

- Permitir el acceso optimizado de aplicaciones y servicios para las distintas generaciones de dispositivos WAP (actuales y futuras). Estos dispositivos generalmente son pequeños, operan con baterías, tienen memoria y CPU relativamente limitadas. Su tamaño nos limita a espacios de pantalla pequeños y limitadas facilidades de entrada de información por parte del usuario.
- Crear sistemas y servicios de aplicación sensibles al limitado ancho de banda que tienen los dispositivos.
- Extender y realzar el modelo de aplicaciones centralizadas para soportar una amplia variedad de dispositivos inalámbricos y redes que es posible con el estándar de la tecnología web, y proveer un entorno más rico en interpretación de documentos.
- Permitir la creación de Interfaces Hombre-Máquina (MMI[3]) con la máxima flexibilidad y capacidad para los fabricantes de dispositivos con el fin de realzar la

[3] Man Machine Interfaces

experiencia del usuario. A este respecto, los fabricantes pueden proveer distintas interfaces de usuario.
- La utilización de códigos de caracteres comunes para su uso internacional. Esto incluye símbolos y pictogramas internacionales para los usuarios finales y la codificación de caracteres de uso local para los desarrolladores.
- Permitir la personalización del dispositivo, del contenido y de la presentación del mismo. Esto lo pueden realizar tanto los usuarios finales, los operadores, los proveedores de servicio y los fabricantes.

IV.2.2.2 Requerimientos

- WAE debe permitir de manera simple y eficiente el desarrollo y ejecución de aplicaciones poderosas y significativas
- WAE no puede asumir que un browser es el agente controlador del dispositivo, tampoco puede asumir que el navegador permanece activo todo el tiempo. Pueden existir otras aplicaciones en el dispositivo, en cuyo caso WAE no debe impedir que dichas aplicaciones coexistan o incluso que se integren al navegador. Adicionalmente pueden acceder e influenciar a los servicios comunes de WAE en el dispositivo cuando sea apropiado.
- WAE no debe asumir ningún modelo de Interfaz Hombre-Máquina (MMI). Las implementaciones WAE deben ser capaces de introducir nuevos modelos MMI o utilizar los existentes. Los implementadores deben ser capaces de presentar sus documentos consistente y significativamente en los dispositivos.
- La convergencia con las especificaciones W3C[4]
    - XHTML definido por W3C como una reformulación de HTML en XML. Al incluir XHTML en las especificaciones de WAP, el WAP Forum incorporará fácilmente nuevas tecnologías del W3C
    - CSS es un lenguaje de hojas de estilo para controlar la presentación de documentos HTML/XML. Los autores pueden controlar la presentación del contenido WAP especificando un subconjunto apto de CSS para las terminales.
- La definición de los *media types* específicos de WAP, ej. WBMP

IV.2.3 Arquitectura WAE

La arquitectura WAE incluye todos los elementos de la arquitectura WAP relacionados con la especificación y ejecución de aplicaciones. La arquitectura WAE se enfoca predominantemente en los aspectos del lado del cliente de la arquitectura del sistema WAP, a saber de los elementos relacionados con los agentes usuario. Específicamente la arquitectura está definida de manera primaria en términos de esquemas de red, formatos de contenido, lenguajes de programación y servicios compartidos. Las interfaces no están estandarizadas y son específicas a una implementación en particular. Esta propuesta le permite a WAE ser implementada en una gran variedad de formas sin comprometer interoperabilidad o portabilidad, además de que ha trabajado particularmente bien con un navegador (una clase de agente usuario) tal y como se usa en WWW. Internet y WWW son la inspiración y motivación detrás de las partes significativas de la especificación WAE, y consecuentemente se ha usado una propuesta similar dentro de WAE.

[4] World Wide Web Consortium

IV.2.3.1 El modelo WWW.

WWW provee de un modelo lógico muy flexible y poderoso. Las aplicaciones presentan el contenido al cliente en un conjunto de formatos de datos que son *navegados* por un agente usuario conocido como Web browser (o simplemente browser) en el cliente. De forma típica el agente usuario envía peticiones a un servidor origen, el cual responde con los datos solicitados expresados en uno de los formatos estándar conocidos por el agente usuario (ej. HTML)

Los estándares de WWW incluyen todos los mecanismos necesarios para construir un ambiente de propósito general como son:

- Todos los recursos en WWW son nombrados con el estándar URL (Uniform Resource Locators)
- Todas las clases de datos en WWW tienen un tipo específico permitiendo al agente usuario el distinguirlos y presentarlos correctamente. Es más, WWW define una gran variedad de formatos estándar que son soportados por la mayoría de los navegadores. Entre ellos encontramos HTML, JavaScript y una larga lista de formatos (ej. Formatos de imágenes)
- WWW también define un conjunto de protocolos de comunicación que permite al navegador comunicarse con cualquier servidor, el más conocido es http

La infraestructura WWW ha permitido a los usuarios alcanzar fácilmente un gran número de aplicaciones y contenido.

IV.2.3.2 El modelo WAE

WAE adopta un modelo muy parecido al modelo WWW. Todo el contenido se especifica en formatos que son similares a los formatos estándar de Internet. El contenido se transporta usando los protocolos estándares en el dominio WWW y en un protocolo parecido a HTTP en el dominio inalámbrico. La arquitectura WAE le permite a todo el contenido y a los servicios ser hospedados en servidores estándar de Web, es decir, todo el contenido se localiza usando URLs estándar.

Se agregan las extensiones WAE para soportar los servicios de la red móvil como el control de llamadas y mensajes. En esta arquitectura se presta especial atención a las restricciones que se tienen en las terminales móviles como son la memoria y el procesamiento de CPU, también se tiene soporte para el reducido ancho de banda.

WAE asume la existencia de un *gateway* responsable de codificar y decodificar la transferencia de datos entre el cliente móvil y el servidor. El propósito de codificar el contenido entregado al cliente es el de minimizar el tamaño del envío de datos al cliente por aire, así como minimizar la energía computacional requerida por el cliente para procesar los datos. La funcionalidad del *gateway* puede ser absorbida por el servidor de origen o se puede separar en un *gateway* dedicado como se muestra en la figura IV.2, donde también se representa un escenario de comunicaciones típico en un entorno Internet-WAP con WMLS.

Los elementos más importantes que incluyen el modelo WAE son:

- **Agentes Usuario WAE**
  Software que se encuentra del lado del cliente que provee de una funcionalidad específica al usuario: ej. Desplegar el contenido de las páginas. Los agentes usuario (como los browsers) están integrados en la arquitectura WAP, ellos interpretan el contenido referenciado por una URL, WAE incluye agentes de usuario codificado para Wireless Markup Languaje (WML) y compilados para Wireless Markup Languaje Script (WMLScript)

- **Generadores de Contenido**
  Aplicaciones (o servicios) que se encuentran en el servidor origen (ej. Scripts CGI) que producen contenido con formato estándar como respuesta a las peticiones de los agentes usuario de la terminal móvil.

- **Codificadores de contenido estándar**
  Un buen conjunto de codificadores de contenido estándar permite al agente usuario WAE (ej browser) navegar de manera conveniente por el contenido web, entre estos codificadores encontramos al codificador de WML, el codificador de *bytecode* para WMLScript, los formatos de imágenes estándar entre otros.

El resultado de la arquitectura WAE encaja dentro de un modelo:

- Que está influenciado por Internet (el modelo aprovecha los estándares, la tecnología y la infraestructura desarrollada para Internet)
- Que está influenciada por la arquitectura *thin-client* (el servicio de despliegue tiene un menor costo por dispositivo debido a la naturaleza independiente de WAE y la administración centralizada de los servicios en los servidores origen)
- Que provee a los usuarios finales de servicios avanzados de la red móvil.
- Que provee los medios necesarios para que los fabricantes puedan construir diferentes servicios amigables para los usuarios que aprovechan los servicios de WWW y la red móvil y
- Que provee un marco extensible y abierto para la construcción de servicios inalámbricos.

Típicamente, un agente usuario en una terminal inicia con una petición, sin embargo, no todo el contenido entregado a la terminal provendrá de una petición de la terminal. Por ejemplo, WTA incluye mecanismos que permiten a los servidores de origen entregar contenido generado sin que la terminal lo haya solicitado como se muestra en la figura IV.3.

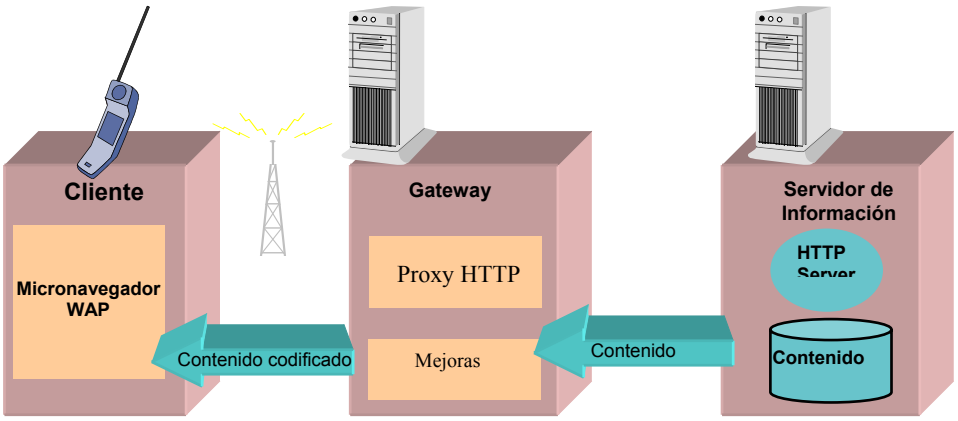


Figura IV.3 *Envío de mensajes sin la petición del cliente*

En algunos casos lo que entrega el servidor de origen al dispositivo podrá depender de las características del dispositivo. Las características del agente usuario se envían al servidor lo que permite a las aplicaciones en el servidor determinar las características del dispositivo móvil. WAE define un conjunto de capacidades que tiene el agente usuario que serán intercambiadas usando los mecanismos de WSP. Estas capacidades incluyen características tales como la versión de WML, WMLScript que soporta el dispositivo, los formatos de imagen que maneja, entre otras.

IV.3 Componentes necesarios para la Navegación en WAP

WAE se divide en dos capas lógicas como se muestra en la figura, dichas capas son:

- Agentes Usuario, los que incluyen browsers, editores de mensajes, directorios etc.
- Formatos y servicios, que incluyen elementos y formatos comunes accesibles para los agentes usuario como WML, WMLScript, formatos de imágenes, formatos vCard y vCalendar, etc.

WAE separa los agentes usuarios de los servicios y asume un entorno con múltiples agentes usuarios. Esta vista lógica no implica o sugiere una implementación en particular, por ejemplo, algunas implementaciones WAE pueden combinar todos los servicios en un solo agente usuario, otros podrán escoger un modelo en el que se distribuyan los servicios entre varios agentes usuario, por estas razones, la implementación será determinada por las decisiones de diseño tomadas por sus implementadores y deberán seguir las limitantes y objetivos de su entorno particular.

IV.3.1 Agente Usuario WAE

El agente Usuario por excelencia es el agente usuario de WML, el cual tiene la particularidad de ser un microbrowser que nos permite navegar en Internet por medio de páginas WML.

WAE no especifica de manera formal ningún agente usuario. Las características y capacidades del agente usuario se le deja a los implementadores. En su lugar, WAE define los servicios y formatos fundamentales que son necesarios para asegurar la interoperabilidad entre implementaciones.

IV.3.2 Agente Usuario WAE

WAE asume que existe un cliente en una red inalámbrica WAP y que un servidor existe en Internet, en algunas configuraciones puede existir un servidor WAP *proxy* entre el cliente y el servidor origen, el cual es el responsable de realizar funciones de *gateway* y *proxy* para la transferencia de datos desde y hacia el cliente. De manera adicional un *proxy* WAP puede proveer servicios adicionales en el comportamiento del cliente. Aún así el cliente y el servidor se pueden comunicar directamente usando el protocolo de Internet o por medio del *proxy* WAP el que tiene la responsabilidad de traducir el protocolo cuando el que es usado en la red inalámbrica WAP es diferente del de Internet.

IV.3.2.1 WAE Media Types.

La tabla IV.1 muestra la lista de los Media Types soportados por WAE, cabe aclarar que un agente usuario WAE puede soportar cualquier otro *media type*.

Data Type	MIME Media Type	Extensión del Archivo
WML1 forma texto	text/vnd.wap.wml	.wml
WML1 forma binaria	application/vnd.wap.wmlc	.wmlc
WML2	application/vnd.wap.wml+xml	.wml, .wml2[5]
XHTML Basico	application/xhtml+xml	.xhtml, .xht
WAP CSS	text/css	.css
WMLScript forma texto	text/vnd.wap.wmlscript	.wmls
WMLScript forma binaria	application/vnd.wap.wmlscriptc	.wmlsc
WBXML	application/vnd.wap.wbxml	
WBMP	image/vnd.wap.wbmp	.wbmp
VCard	text/x-vCard	.vcf
vCalendar	text/x-vCalendar	.vcs
WTA-WML forma texto	text/x-wap-wta-wml	
WTA-WML forma binaria	application/x-wap-sta-wmlc	
Data Type	**MIME Media Type**	**Extensión del Archivo**
Forma textual de Multipart Messages que son usadas cuando las partes del cuerpo son independientes y necesitan ser organizados en un orden en particular	multipar/ mixed	
Forma binaria de Multipart Messages que son usadas cuando las partes del	application/vnd.wap.multipart.mixed	

[5] La extensión "wlm2" se provee para la configuración de algunos servidores WWW que dependen de las extensiones de los archivos para asignar el Content-Type para cada recurso. Este problema puede ser resuelto por diversos métodos.

cuerpo son independientes y necesitan ser organizados en un orden en particular		
Forma textual de Multipart Messages que representan objetos en las partes de un cuerpo MIME	multipart/related	
Forma binaria de Multipart Messages que representan objetos en las partes de un cuerpo MIME	application/vnd.wap.multipart.related	
Forma textual de Multipart Messages que se usan cuando cada parte del cuerpo es una versión alternativa de la misma información	multipart/alternative	
Forma binaria de Multipart Messages que se usan cuando cada parte del cuerpo es una versión alternativa de la misma información	application/vnd.wap.multipart.alternative	
Forma textual de Multipart Message para regresar valores de una forma	multipart/form-data	
Forma binaria de Multipart Message para regresar valores de una forma	application/vnd.wap.multipart.form-data	
Canales en forma texto	text/vnd.wap.channel	
Canales en forma binaria	application/vnd.wap.channelc	
Indicación de servicio en forma texto	text/vnd.wap.si	
Indicación de servicio en forma binaria	application/vnd.wap.sic	
Carga de servicio en forma texto	text/vnd.wap.sl	
Carga de servicio en forma binaria	application/vnd.wap.slc	
Operación de Cache en forma texto	text/vnd.wap.co	
Operación de Cache en forma binaria	application/vnd/wap.coc	
Aprovisionar documento en forma texto	text/vnd.wap.connectivity-xml	
Aprovisionar documento en forma binaria	Application/vnd.wap.connectivity-wbxml	

Tabla IV.1 *Media Types*

Un agente usuario debe usar el MIME Media Type para determinar el tipo de datos en un documento. El procesamiento del documento no debe depender de la extensión cuando está disponible el MIME Media Type.

IV.3.2.2 Imágenes Gráficas

El formato Wireless BitMaP (WBMP) permite enviar información gráfica a una gran variedad de dispositivos inalámbricos. El formato WBMP es independiente de los dispositivos y describe sólo información gráfica.

Un *proxy* WAP puede transformar la imagen gráfica en otro formato cuando el Agente Usuario no soporta el formato de la imagen original, para esto el Agente Usuario se comunica con el *proxy* WAP para informarle de los formatos de imágenes que soporta.

IV.4 Metas del Diseño

Cualquier herramienta de software interactivo que maneja programas inevitablemente despliega código fuente para que cualquier persona pueda leerlo o posiblemente modificarlo.

La meta principal de un editor de Lenguajes es ayudar a los programadores a realizar su labor mediante sus propias habilidades y las herramientas con las que cuentan. Esto nos traslada a dos conjuntos de requerimientos, que en algunas ocasiones pueden resultar conflictivos entre sí:

- *La perspectiva del programador*: El editor deberá lograr que la lectura y la escritura del código fuente sea más fácil y más provechosa.
- *La perspectiva del diseñador del editor.* El editor deberá compartir información con otras herramientas, con las cuales podrá actuar como interfaz de usuario.

Los requerimientos antes mencionados reflejan las diferentes perspectivas: la de los programadores y la de los diseñadores de las herramientas. Las antiguas fallas son el resultado del descuido de alguno de los dos puntos de vista.

IV.4.1 Sin entrenamiento

Toda la evidencia disponible demuestra que los programadores leen los programas de manera textual; así mismo tienen una manera de pensar estructurada, pero es altamente variable y no se basa en el análisis del lenguaje [Holt *et al.*, 1987][Letovsky, 1986]. Los programadores tienen inculcados hábitos de trabajo, así como el aprendizaje motorizado que involucra la edición textual; únicamente aceptarán una herramienta que les resulte lo suficientemente familiar como para utilizarla de manera inmediata y cómoda sin la necesidad de entrenamiento especial.

Esta necesidad no prohíbe la funcionalidad avanzada. Consideremos cómo los usuarios experimentados en simples editores de texto realizan la transición hacia los procesadores de texto de manera sencilla por la entrada de texto y los comandos.

IV.4.2 Reforzamiento de la lectura y la escritura

Servicios de edición adicionales derivan de la especialización de tareas de los programadores. Un ejemplo familiar es la identación automática de las líneas del código fuente. Este servicio está basado en la estructura lingüística y ayuda en la lectura (retroalimentación visual en la anidación) y la escritura (ahorrando la pulsación innecesaria). Este servicio en particular puede ser utilizado en un simple editor de texto, pero puede y debe ser llevado más allá.

En muchos ambientes, la lectura es la tarea predominante para los programadores, aún más que la escritura de código. También es altamente importante que soporte los comentarios. Para cualquier lenguaje los comentarios son tediosos en el formato pero cruciales para los lectores.

Aunque las perfecciones son importantes, es esencial que no compliquen las cosas. Cualquier intrusión en la edición de texto debe respetar el "equilibrio del poder" entre el usuario y la herramienta. Esto puede ser delicado incluso en el más simple de los casos, por ejemplo los mecanismos de auto identación que los programadores encuentran útiles pero "no del todo correctas."

IV.4.3 Acceso a la estructura lingüística

Las herramientas de Ingeniería de Software (por ejemplo analizadores, compiladores, etc.) operan sobre representaciones de código fuente estructuradas como son los árboles abstractos de sintaxis. Una herramienta de edición es integrada más fácilmente con otras herramientas si puede compartir dichas representaciones, pero como se presentará más adelante, esto implica un gran desafío de diseño para una herramienta cuyo trabajo es desplegar y permitir la modificación del código fuente en términos de texto.

IV.4.4 Configuración e incrustación

Finalmente, como las herramientas de ingeniería de software evolucionan, existen cambios de los sistemas de edición solitarios a herramientas especializadas que deben trabajar con otras herramientas. Una herramienta para la edición de código fuente debe estar bien encapsulada, algo como un componente GUI (Interfaz Gráfica de Usuario), y que no demande apoyo complejo como un tipo especial de repositorio de código fuente. Reflejando la realidad de que la ingeniería de software involucra muchos lenguajes, debe ser fácilmente configurable por medio de las especificaciones del lenguaje.

IV.5 El espacio de diseño

En el corazón de una herramienta especializada en la edición, se encuentra la representación interna para el código fuente. Las opciones convencionales están divididas por un abismo entre diferentes enfoques: uno orientado hacia la utilidad y otro orientado hacia los altos niveles de servicio como se muestra en la figura IV.4.

Figura IV.4 *Elecciones de Diseño para programas editores.*

IV.5.1 Diseños Puros

En el lado derecho del diagrama se encuentran los "editores estructurados" [Notkin 1985], llamados de esta manera porque las representaciones internas están estrechamente relacionadas con las estructuras de árbol y gráfico usadas por los compiladores y otras herramientas. Esto simplifica grandemente algunos tipos de servicios orientados al lenguaje, pero requiere que el programador edite de manera estructural en lugar de hacer

uso de comandos textuales. Desde la perspectiva de una herramienta de integración las ventajas de un análisis lingüístico completo son desplazadas por su fragilidad (en la presencia de un usuario) y la dependencia del contexto (el significado del código en muchos lenguajes depende potencialmente de otro código con el cual deberá funcionar). Hoy en día pocos editores estructurados son utilizados.

En el lado izquierdo se encuentran los editores de texto simples que no cuentan con el soporte lingüístico. La edición resulta simple y familiar, pero no existe una especialización real respecto al código fuente. Integrando un editor simple de texto con las herramientas de ingeniería de software requiere de un complejo trazado entre la estructura y el texto, pero esto resulta típicamente en una funcionalidad restrictiva y confusa y en representaciones frágiles (por ejemplo, donde la identidad de los elementos estructurales no se preserva durante las operaciones de edición) o ambas [Van De Vanter *et al,* 1995].

IV.5.2 Diseños Modificados

Esfuerzos subsecuentes en la edición basada en lenguaje pueden ser vistos como intentos para sortear este abismo. Algunos editores estructurales permiten a los programadores "escapar" la estructura transformando regiones selectas del árbol en texto plano [Reps *et al*, 1989], pero los problemas de utilización persisten. La compleja relación entre el desplegado textual y la representación interna vuelve confusas y aparentemente impredecibles las operaciones de edición de escape estructural y textual por el "estado escondido" [Van De Vanter *et al,* 1995]. Los escapes textuales dan importancia a una distinción confusa entre las partes del programa donde se proporcionan servicios basados en el lenguaje y donde no. En algunas ocasiones los servicios de lenguaje y las herramientas se detienen hasta que todas las regiones textuales son sintácticamente correctas y promovidas a la estructura.

Del lado izquierdo de la figura IV.5 se encuentran los ampliamente usados editores de texto orientados a código. Éstos utilizan una representación puramente textual, asistidas por expresiones regulares *ad-hoc* que reconocen ciertas construcciones de lenguaje. La información estructurada computada por un simple editor de texto es, por definición, incompleta e imprecisa. Por consiguiente no soporta servicios que requieren de un análisis lingüístico. Los editores de texto simples proveen de identación, resaltado de sintaxis y servicios de navegación que pueden soportar errores estructurales. Un programa malformado puede, en el peor de los casos, estar incorrectamente resaltado.

Pocos editores de texto llevan a cabo un análisis léxico por línea con cada pulsación de tecla, pero la información no es explotada totalmente y la falta de una representación verdadera del programa produce confusión en la inevitable presencia de comillas o de los delimitadores de comentarios.

IV.5.3 Diseños Inclusivos

Un acercamiento más inclusivo se presenta en mantener las representaciones textuales y estructurales. Aunque este acercamiento promete un gran número de ventajas [Van De Vanter *et al,* 1992], resulta difícil el mantener las representaciones consistentes y no se ha demostrado que la complejidad y el costo estén justificados.

IV.6 Encontrando el punto medio

La sección 5 describió la tensión fundamental del diseño:

- Es deseable el mantener una representación de programa lingüísticamente exacta, actualizando en cada modificación, aunque sea pequeña.
- Mientras más grande sea la sofisticación estructural, más frágil es la representación en la presencia de la edición textual sin restricciones y hay más cabida para el comportamiento confuso e inconsistente entre lo que se ve y lo que se representa de manera interna.

En resumen, una representación ideal estaría estrechamente relacionada al desplegado del texto, pero también reflejaría la estructura lingüística en todo momento. Lo que se necesita es el compromiso en algún lugar en el centro de la Figura, donde la cantidad de análisis del lenguaje realizado es simple, posible y útil.

El compromiso puede ser encontrado al echar un vistazo más de cerca al análisis del lenguaje: tanto para la ingeniería interna de los compiladores como para la teoría formal del lenguaje detrás de ella. Un compilador analiza los programas textuales en fases, como se muestra a continuación:

Análisis léxico → parsing → análisis semántico

Cada etapa se maneja por un tipo diferente de gramática (correspondiendo aproximadamente a los tipos 3, 2 y 1 dentro de la jerarquía de gramáticas de Chomsky) y utiliza su correspondiente tipo de analizador. A menudo los lenguajes de programación están diseñados de acuerdo a esta descomposición gramatical y los compiladores orientados a bloques se benefician de la simplicidad y las fundaciones formales de las fases separadas.

Esta descomposición revela opciones adicionales, como se muestra en la figura IV.5, para analizar y representar programas editados. Las posibles representaciones incluyen las producciones estándar de cada fase: *lexical token stream*, *parse tree*, y el *attributed tree* respectivamente. Las opciones intermedias incluyen análisis parcial del siguiente nivel gramatical: la coincidencia de las expresiones regulares es un análisis léxico parcial, el *fuzzy parsing* es un análisis sintáctico parcial que reconoce únicamente ciertas características de la sintaxis libre de contexto (ej. Paréntesis anidados o la categorización dependiente del contexto de los identificadores dentro de una función o los nombres de las variables), y la atribución semántica parcial que puede ser usada para computar cantidades limitadas de contexto semántico. En algunas ocasiones los análisis parciales son más simples de implementar (el *fuzzy parsing* puede ser llevado a cabo a través de un simple patrón de *match* en el *token stream*) y más perdonables de inconsistencias en la representación.

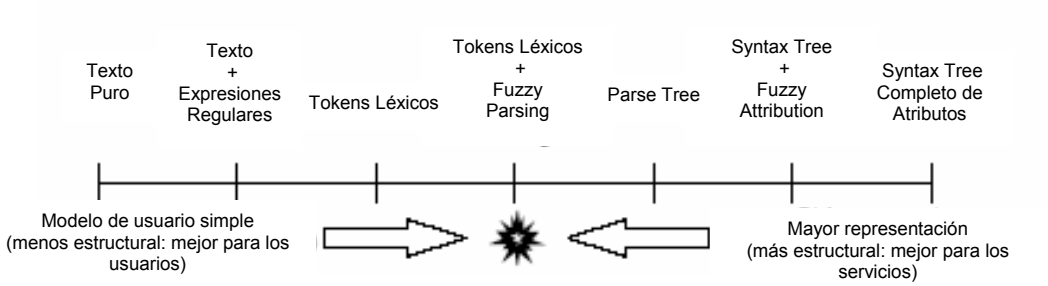


Figura IV.5 *Opciones adicionales para representación del programa y análisis*

Una importante distinción entre las tres fases del análisis concierne al ámbito de causa y efecto. El análisis semántico (relacionado estrechamente con la sintaxis sensitiva al contexto de Chomsky) en cada punto del programa depende potencialmente del programa entero. Parsing (sintaxis libre de contexto) depende únicamente de la frase adjunta, pero asume que el programa está bien formado. Análisis Léxico (sintaxis regular) depende únicamente de los tokens adyacentes, haciéndolos altamente convenientes para el ciclo interno de un editor.

Así, la representación léxica, emerge de un compromiso prometedor:

- El análisis necesita actualizar la representación después de cada edición, usualmente requiere únicamente del contexto local.
- Es conveniente para los fragmentos de programa.
- Tiene la suficiente información lingüística para proveer muchos servicios basados en el lenguaje, incluyendo más implementaciones robustas de servicios familiares como identación, la coincidencia de paréntesis y llaves, etc.
- Es una representación del lenguaje conveniente para la integración con otras herramientas, incluyendo analizadores completos del lenguaje.

Aunque este acercamiento es prometedor, quedan pendientes varias preguntas de diseño.

- ¿Pueden el comportamiento y el desplegado textual ser hechos de manera tan familiar que no se requiera de un entrenamiento?
- ¿A qué grado puede el display estar especializado para programas utilizando únicamente información léxica?
- La representación del lexical token puede ser hecha robusta en la presencia de tokens malformados y parcialmente escritos. Por ejemplo la falta de cierre de un paréntesis o de las comillas dobles
- ¿Qué tan especializado debe ser el soporte para los comentarios y otras anotaciones no textuales?
- ¿Cómo puede ser adaptado el analizador léxico para que actualice la representación después de cada pulsación de tecla?

Las respuestas a estas interrogantes se encuentran en las siguientes secciones donde se resumen dos de los aspectos mutuamente dependientes del diseño:
La arquitectura / implementación y el modelo de usuario.

IV.7 Arquitectura

La arquitectura se conforma de tres módulos separados en dos componentes: uno que implementa la funcionalidad del lenguaje y el otro (conocido como Módulo de Lenguaje) que provee de las características sensitivas del lenguaje. En lo que resta de esta sección se hablará de cada una de las partes de la arquitectura.

IV.7.1 El Controlador

Se basa en el uso de un Editor Kit de las Java Foundation Classes “Swing” el cual implementa el comportamiento complicado de la edición.

Gran parte de la funcionalidad del Editor Kit es independiente del lenguaje; su principal responsabilidad es implementar las acciones del usuario que requiere tomar el contexto de la acción en consideración. Algunas acciones, como los comandos de movimientos del cursor, no requieren cambios en el modelo del código fuente; su ejecución depende únicamente del contexto (tokens). Otras acciones, como las inserciones y borrado, pueden depender no únicamente de la modificación del contexto, sino también del estado después de la modificación, debido a que ciertos matices del modelo de usuario requieren “ver hacia el futuro”.

Para facilitar esta tarea, el Editor Kit, comienza un proceso de modificación de dos estados en cualquier cambio potencial. Primero, el código fuente es pedido para considerar los efectos del cambio sin modificar el contenido. Esto produce un objeto que describe el cambio en términos de un modelo de transformación que necesita tener lugar. Cuando el Editor Kit recobre el control, examina la transformación, descartándola si no tiene efecto o no es válida o aplicándola al modelo.

IV.7.2 El Modelo

Como se discutió en la sección anterior, el código fuente se representa como una secuencia de tokens léxicos, aunque esta representación se extiende en muchas maneras cruciales. Esta representación permite mucha flexibilidad requerida, debido a que soporta el modelo de usuario y encaja naturalmente con el análisis léxico, el cual puede ser implementado por JavaCC [JAVACC] desde una especificación léxica.

El modelo del código fuente es responsable de agregar y quitar “separadores”, tokens no lingüísticos como comentarios, saltos de línea y otras directivas.

IV.7.3 La Vista

La pantalla es facilitada por la asignación de propiedades estilísticas para cada token por los medios del componente *Styler*. Los *Stylers* pueden ser usados para exportar código fuente legible por el usuario interpretándolo en un flujo de caracteres, arrojando información estilística que no puede se representada. El formato apropiado puede ser alcanzado por *Stylers* optimizados para la salida de texto.

IV.8 Funcionalidad y Modelo de Usuario

Esta sección presenta un vistazo al comportamiento funcional del editor, así como la experiencia del programador en el modelo de usuario.

IV.8.1 Tipografía avanzada

El editor se distinguirá visualmente por sus “estilos” tipográficos avanzados por la arquitectura descrita en la sección IV.7.3.

Cada estilo de token especifica tipo de letra, tamaño, estilo (plano, negrita, itálica), los colores de primer plano y el fondo y las especificaciones de los límites izquierdo y derecho usados para el desplegado de los espacios entre tokens adyacentes.

IV.8.2 Comportamiento de la edición

El editor se comportará como un editor de texto orientado a código en la mayoría de los aspectos. Algunos comportamientos serán completamente convencionales. La identación será automática. Los saltos de línea serán explícitamente introducidos y borrados por el programador. La escritura de texto dentro de comentarios y tokens del lenguaje (especialmente literales de tipo cadena) será igualmente convencional.

El comportamiento no estándar aparecerá dentro y alrededor de los límites de los tokens. Los límites de los tokens son determinados por el analizador léxico.

Referencias

[Holt *et al.*, 1987] Robert W. Holt, Deborah A. Boehm-Davis and Alan C. Schultz, "Mental Representations of Programs for Students and Professional Programmers" en *Empirical Studies of Programmers: Second Workshop,* Ablex Publishing, Norwood, New Jersey,1987

[Letovsky, 1986] Stanley Letovsky, "Cognitive Processes in Program Comprehension," en *Empirical Studies of Programmers,* Ablex Publishing, Norwood, New Jersey,1986

[Notkin 1985] David Notkin, "The GANDALF Project", *Journal of Systems and Software,* 1985

[Van De Vanter *et al,* 1995] Michael L. Van De Vanter, "Practical Languaje-Based Editing for Software Engineers", en *Software Engineering and Human-Computer Interaction,* Spinger Verlag, Berlin,1995

[Reps *et al*, 1989] Thomas Reps and Tim Teitelbaum, *The Synthesizer Generator Reference Manual,* Springer Verlag, Berlin, 1989

[Van De Vanter *et al,* 1992] Michael L. Van De Vanter,Susan L. Graham and Robert A. Ballance, "Coherent User Interfaces for Language-Based Editing Systems," en *International Journal of Man-Machine Studies,*1992

[JAVACC] JavaCC – The Java Parser Generator: A product of Sun Microsystems" http://javacc.dev.java.net

CAPÍTULO V
Análisis del Sistema de Edición y Emulación de Documentos WML

V.1 Generalidades

Para la propuesta, el análisis y el diseño del Sistema de Edición y Emulación de Documentos WML (SEEDWML) nos basamos en una tecnología de nombre UML (*Unified Modeling Language*).

En este capítulo abarcamos los dos primeros pasos del flujo de trabajo definido en el "Proceso unificado de desarrollo de software" (que se explicará después)

V.1.1 ¿Qué es el UML?

El lenguaje unificado de modelado o UML (por sus siglas en inglés) es el sucesor de la oleada de métodos de análisis y diseño orientados a objetos (OOA&D) que surgió a finales de la década de 1980 y principios de la siguiente. El UML unifica, sobre todo, los métodos de Booch, Rumbaugh (OMT) y Jacobson, pero su alcance llega a ser más amplio debido a los modelos que maneja.

V.1.2 Historia del UML

Los métodos de desarrollo para los lenguajes de programación tradicionales, tales como C, Cobol y Fortran, emergieron en los años 70 y llegaron a ser ampliamente difundidos en los 80. Principalmente entre ellos estaba el Análisis estructurado y el diseño estructurado.

El primer lenguaje que es generalmente reconocido como orientado a objetos es Simula 67, desarrollado en 1967. Este lenguaje nunca tuvo un seguimiento significativo, aunque influyó notablemente en los desarrolladores de varios de los lenguajes orientados a objetos posteriores. El movimiento de orientación a objetos se convirtió en un esfuerzo constante gracias a la amplia difusión de la disponibilidad del Smalltalk a principios de los 80, seguido por otros lenguajes orientados a objetos como C++, Eiffel, etc.

El uso real de los lenguajes orientados a objetos fue limitado al principio, pero la orientación a objetos atrajo mucho la atención. Aproximadamente cinco años después de que Smalltalk llegará a ser conocido, fueron publicados los primeros métodos de desarrollo orientado a objetos, Shlaer/Mellor (1988), Coad/Yourdon (1991), Booch (1991) y Rumbaugh (1991), por mencionar algunos. El libro de Objectory de Jacobson (1992) en el cual se introduce el concepto de casos de uso y proceso de desarrollo, completa la primera fase del desarrollo de metodologías orientadas a objetos.

El primer intento exitoso de combinar los métodos existentes se da cuando Rumbaugh se une a Booch en Rational Software Corporation en 1994. Ellos combinaron conceptos de los métodos OMT (Rumbaugh) y BOOCH (Booch), obteniendo una primera propuesta en

1995. En ese año se unió Jacobson al equipo de desarrollo de Rational y comenzó a trabajar con Rumbaugh y Booch, el resultado final de ese proyecto en conjunto fue llamado Lenguaje Unificado de Modelado (*UML* –por sus siglas en inglés-).
En 1996, El Object Management Group (OMG) publicó una petición de propuestas para una estandarización sobre el modelado orientado a objetos. Los autores del UML se pusieron en contacto con metodólogos y desarrolladores de otras compañías para obtener una propuesta atractiva para los miembros de OMG. Todas las propuestas se unieron a la propuesta final de UML que fue sometida a consideración del OMG a finales de Septiembre de 1997. El UML fue adoptado únicamente por los miembros del OMG como estándar en noviembre de 1997, a partir de ese momento comenzó un proceso de aceptación por parte de la comunidad de analistas y desarrolladores que va ganado terreno, y que ha colocado al UML como la herramienta más difundida para el modelado.

V.1.3 Conceptos de UML

Hay tres conceptos básicos en UML, éstos son vocabulario, modelos y vistas.

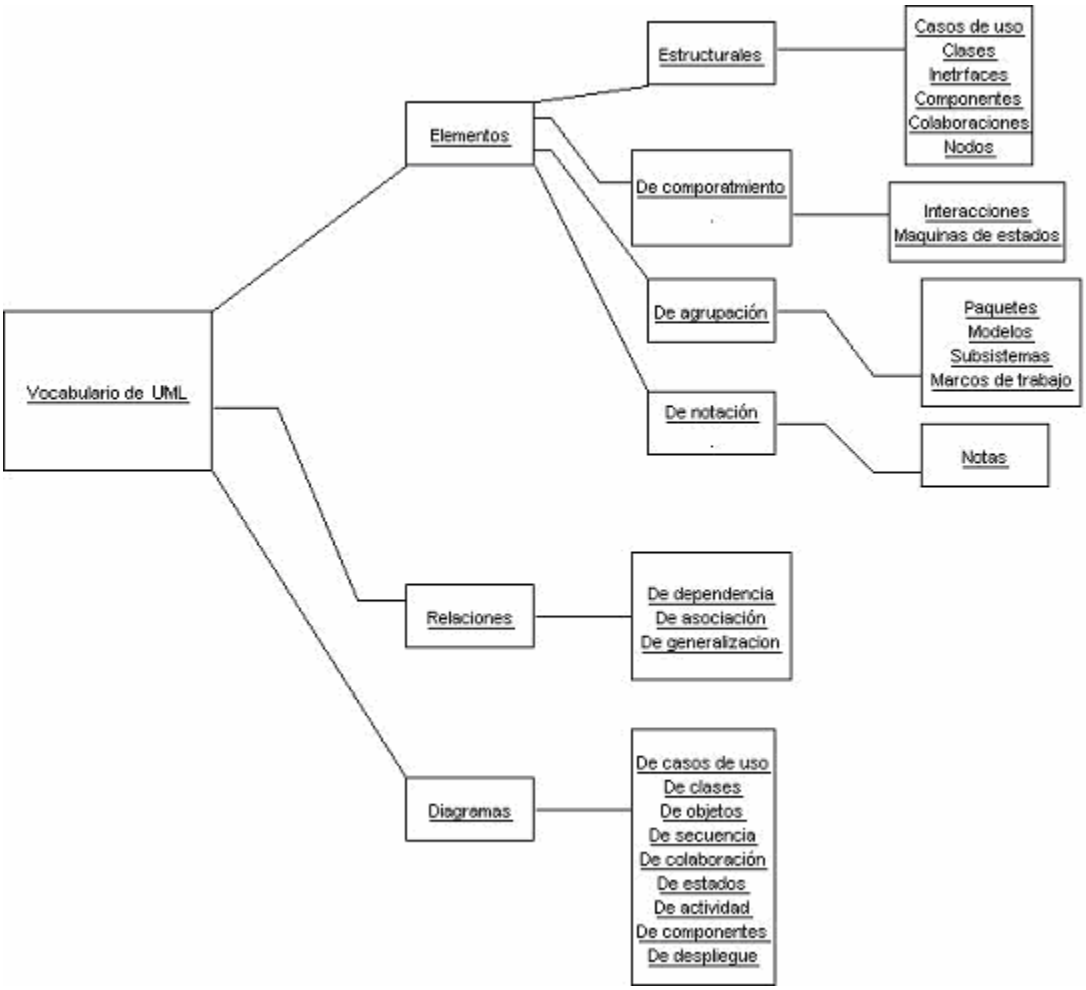
El vocabulario es toda la serie de términos que vamos a manejar en UML y debido a que UML es un lenguaje gráfico, el lenguaje que manejamos en UML tiene que ver con símbolos y diagramas. Los modelos son, de manera sencilla, una simplificación de la realidad. Las vistas son nuestra manera de organizar (agrupar) estos modelos, gracias al enfoque que dan.

V.1.3.1. Vocabulario de UML

En UML tenemos un vocabulario que podemos dividir en tres categorías

- Elementos
- Relaciones
- Diagramas

Como se muestra en el esquema V.1:



Esquema V.1. *Vocabulario de UML*

V.1.3.2. Modelos

Para poder hablar de modelos debemos entender y dejar claro un concepto que puede causar confusión, éste es el de abstracción.

El termino “abstracción” llega a ser confundido con “encapsulado” y “ocultamiento de información”, pero debe dejarse bien claro que son conceptos completamente diferentes.

Definiremos primero cada uno de los conceptos para después mostrar en conjunto sus diferencias.

La abstracción se puede tratar como proceso o entidad, para el caso en el que se trata como proceso, citaremos la definición de la IEEE [IEEE,1983]:

> *“La vista de un problema que extrae la información relevante a un propósito particular e ignora el resto de la información”*

Una definición que trata el concepto como entidad es la de Booch [Booch, 1991] , que cita:

> *"Una abstracción denota las características esenciales de un objeto que lo distingue de los otros tipos de objeto y de esta manera proporciona tajantemente límites conceptuales, relativos a la perspectiva del observador"*

Dentro de la abstracción encontramos diferentes grados de abstracción (también llamados niveles), en los cuales, mientras más alto sea el nivel de abstracción, estaremos más enfocados en piezas más importantes de información, en los niveles de abstracción más bajos nos encontramos con más detalles y con más elementos individuales, lo cual incrementa el volumen de información con el cual debemos tratar.

El ocultamiento de información lo define Budd [Budd,1986] como:

> *"El ocultamiento de información es el principio de que, los usuarios de un componente de software (como una clase, por ejemplo), deben conocer solamente los detalles esenciales de cómo inicializar y acceder al componente y no necesitan saber los detalles de la implementación"*

Aquí podemos marcar la primera diferencia entre los dos términos citados anteriormente, ya que, la abstracción es una técnica para definir qué información se esconde y cuál no.

El encapsulamiento es el término que más confusión provoca, inclusive se le trata como sinónimo de ocultamiento de información, pero veremos que son bien diferentes uno del otro, veamos la definición que da Wirfs [Wirfs-Brock *et al*,1990]:

> *"El concepto de encapsulamiento como es usado en un contexto orientado a objetos no es esencialmente diferente de su definición en un diccionario. Todavía se refiere a construir una cápsula, en ese caso una barrera conceptual, alrededor de alguna colección de cosas"*

Observemos la siguiente definición, hecha por Blair [Blair et al, 1991], que nos acerca más aún a la comprensión de los tres términos que estamos tratando:

> *"El encapsulamiento es usado como un término genérico para técnicas las cuales utilizan abstracción de datos. Encapsulamiento por lo tanto implica la obtención de mecanismos para apoyar tanto modularidad como ocultamiento de información. Hay por tanto una correspondencia uno a uno, en este caso, entre la técnica de encapsulamiento y el principio de abstracción de datos"*

Dadas las definiciones podemos decir entonces que encapsular es agrupar elementos comunes, entonces, podemos ver las diferencias y podemos concluir que aunque los términos son muy diferentes, también pueden parecer muy cercanos, y también podemos citar que:

> *"La abstracción es la técnica que nos ayuda a definir qué información deberá estar oculta y cuál no, por medio del enfoque (del usuario) de lo que es importante en ciertos niveles. Y el encapsulamiento es la técnica para empaquetar la información a manera de ocultar lo que debe estar oculto y hacer visible lo que pensamos hacer visible"*

Y para hacer más claro este razonamiento pensemos esto: si encapsulamiento fuera lo mismo que ocultamiento de información, entonces, todo lo que está encapsulado está oculto, lo cual es falso. [TOA]

Con estos puntos claros podemos ahora dar una definición más formal de modelo:

> *"Un modelo es una abstracción semánticamente completa de un sistema"*

Esto significa que un modelo es una abstracción más o menos completa de un sistema desde un punto determinado de vista. Por tanto distintos modelos proporcionan puntos de vista diferentes e independientes que podemos manejar por separado. Y aunque parece obvio, la siguiente pregunta no está de más ¿para que modelar?

Nosotros construimos modelos para visualizar y controlar la arquitectura del sistema así como para una mejor comprensión del sistema que estamos desarrollando.

Existen cuatro principios básicos del modelado [Booch, 1999]:

1º. La elección de qué modelos crear tiene una profunda influencia sobre cómo se acomete un problema y cómo se da forma a una solución.
2º. Todo modelo puede ser expresado a diferentes niveles de precisión.
3º. Los mejores modelos están ligados a la realidad.
4º. Un único modelo no es suficiente. Cualquier sistema no trivial se aborda mejor a través de un pequeño conjunto de modelos casi independientes.

Y aunque no está escrito, es importante mencionar el principio cero del modelado, dictado por Edsger Dijkstra, que se basa únicamente en el principio de divide y vencerás. Éste es:

> *Acometer un problema difícil dividiéndolo en una serie de subproblemas más pequeños que se puedan resolver.* [Booch, 1996]

Ahora una segunda y última pregunta obligada ¿cómo utilizar los modelos?

La respuesta la daremos con el enfoque de UML, el modelo de casos de uso nos ayuda a determinar el alcance. Para esbozar algunos conceptos se emplea el diagrama conceptual de clases, conceptos de casos de uso. Esto nos puede ayudar para saber cómo encaja el caso de uso en el software. Si el caso de uso contiene elementos importantes de flujo de trabajo, éstos (los elementos) se pueden ver mediante un diagrama de actividades.

Durante este proceso, las técnicas se pueden usar en conjunto con el experto del dominio. Después de esto, un diagrama de clases correspondiente a la perspectiva de especificaciones puede ser muy útil para proyectar con mayor detalle las clases. Los diagramas de interacción son valiosos para mostrar la manera en que interactuarán las clases para implementar el caso de uso.

Por último, cabe mencionar que quizá el aspecto más importante de un modelo, es que nos permite fallar en condiciones controladas, para luego corregir el error, sin arriesgar el producto final.

V.1.3.3 Las vistas en UML

No hay ninguna línea entre los diferentes conceptos y las construcciones en UML pero por conveniencia se dividen en diferentes vistas, como se muestra en la tabla V.1. Una vista es simplemente un subconjunto de UML que modela construcciones que representan un aspecto en una sistema. Una o dos clases de diagramas proporciona una notación visual para los conceptos de cada vista.

Área	Vista	Diagramas	Conceptos principales
Estructural	Vista estática	Diagrama de clase	Clase, asociación, generalización, dependencia, realización, interfaz.
	Vista de casos de uso	Diagrama de casos de uso	Caso de uso, actor, asociación, extensión, inclusión, generalización de casos de uso.
	Vista de implementación	Diagrama de componentes	Componente, interfaz, dependencia, realización.
	Vista de despliegue	Diagrama de despliegue	Nodo, componente, dependencia, localización.
Dinámica	Vista de máquina de estados	Diagrama de estados	Estado evento, transición, acción.
	Vista de actividad	Diagrama de actividad	Estado, actividad, transición de terminación, división, unión.
	Vista de interacción	Diagrama de secuencia	Interacción, objeto, mensaje, activación.
		Diagrama de colaboración	Colaboración, interacción, rol de colaboración, mensaje.
Gestión del modelo	Vista de gestión del modelo	Diagrama de clases	Paquete, subsistema, modelo.
Extensión de UML	Todas	Todos	Restricción, estereotipo, valores etiquetado.

Tabla V.1. *Vistas en UML* [Rumbaugh *et al*, 2000]

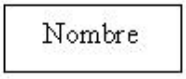
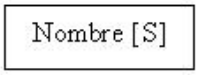
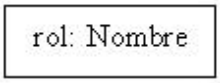
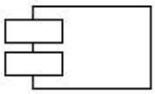
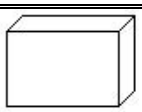
En la tabla V.1 se muestra una tabla dividida en tres áreas: clasificación estructural, comportamiento dinámico y gestión del modelo.

Para efecto de este trabajo sólo se considerarán cuatro vistas, ya que, como se explicará más adelante en el proceso unificado de desarrollo de software, no se emplean todas las vistas y diagramas existentes en el UML, sino que usamos los diagramas que nos den una visión más sencilla de los procesos, relaciones e interacciones dentro del sistema, y no todos; por tanto, las vistas que consideramos para nuestra aplicación (de tipo monolítica) son: La vista estática, la vista de casos de uso y la vista de interacción.

V.1.3.3.1 Vista estática

La vista estática es la base de UML. Los elementos de la vista estática de un modelo son los conceptos significativos de cualquier aplicación. La vista estática captura la estructura del objeto, e incluye todo lo referente a las estructuras tradicionales, así como la organización de las operaciones sobre los datos. Los datos y las operaciones son cuantificadas en clases. Los elementos clave en la vista estática son los clasificadores y sus relaciones.

Un clasificador es un elemento del modelado que describe cosas. Los aspectos del comportamiento son representados por otros clasificadores, como los casos de uso, por ejemplo. Los clasificadores tienen identidad, estado, comportamiento y relaciones. Entre los clasificadores tenemos la clase, la interfaz y los tipos de datos. La tabla V.2 enumera las distintas clases de clasificadores y sus funciones.

Clasificador	Función	Notación
Actor	Un usuario externo al sistema	
Clase	Un concepto del sistema modelado	
Clase en un estado	Una clase restringida a estar en el estado dado	
Rol	Clasificador restringido a un uso particular en una colaboración	
Componente	Una pieza física de un sistema	
Tipo de dato	Un descriptor de un conjunto de valores primitivos que carecen de identidad	**Nombre**
Interfaz	Un conjunto de operaciones con nombre que caracteriza un comportamiento	 **Inombre**
Nodo	Un recurso computacional	

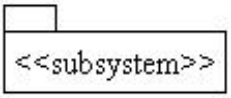
Clasificador	Función	Notación
Señal	Una comunicación asíncrona entre objetos	
Subsistema	Un paquete que es tratado como una unidad con una especificación, implementación e identidad	
Caso de uso	Una especificación del comportamiento de una identidad y su interacción con los agentes externos	

Tabla V.2 *Clasificadores y sus funciones* [Rumbaugh *et al*, 2000]

Las relaciones entre clasificadores son asociación, generalización, flujo y varias clases de dependencia, que incluyen la realización y el uso. En la tabla V.3 se presentan los tipos de relaciones.

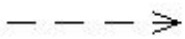
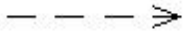
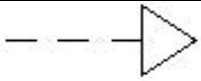
Relación	Función	Notación
Asociación	Una descripción de una conexión entre instancias de clases	
Dependencia	Una relación entre dos elementos del modelo	
Flujo	Una relación entre dos versiones de un objeto en sucesivas veces	
Generalización	Una relación entre una descripción más general y una variedad más específica de la general, usada para herencia	
Realización	Relación entre una especificación y su implementación	
Uso	Una situación en la que un elemento requiere otro para su correcto funcionamiento	

Tabla V.3 *Tipos de Relaciones* [Rumbaugh *et al*, 2000]

V.1.3.3.2 Vista de casos de uso

La vista de casos de uso captura el comportamiento de un sistema, de un subsistema o de una clase, tal como se muestra a un usuario exterior. Los elementos de la vista de casos de uso son los actores, los casos de uso y las relaciones. A continuación, en la tabla V.4, se presenta una tabla con los diferentes tipos de relaciones para la vista de casos de uso.

Relación	Función	Notación
Asociación	La línea de comunicación entre un actor y un caso en el que participa	

Relación	Función	Notación
Extensión	La inserción de comportamiento adicional en un caso de uso base que no tiene conocimiento sobre él	<<extend>>
Generalización de casos de uso	Una relación entre un caso de uso general y un caso de uso más específico, que hereda y añade propiedades a aquel	
Inclusión	Inserción de comportamiento adicional en un caso de uso base, que describe explícitamente la inserción	<<include>>

Tabla V.4 *Relaciones en los casos de uso* [Rumbaugh *et al*, 2000]

V.1.3.3.3 Vista de interacción

Los diagramas de secuencia y de colaboración son dos de los diagramas utilizados para representar los aspectos dinámicos del sistema. Para modelar este aspecto falta preguntarse cómo funcionaría el sistema en ejecución.

En estos diagramas vamos a encontrar tres partes principales:

- Objetos
- Enlaces
- Mensajes

Una colaboración es una descripción de cómo interactúan los objetos para implementar un cierto comportamiento del sistema, tiene un aspecto estructural y de comportamiento.

Una interacción es un conjunto de mensajes dentro de una colaboración. Estos mensajes se pueden ordenar en hilos secuenciales de control.

Un diagrama de secuencia representa una interacción como un gráfico bidimensional. La dimensión vertical es el eje del tiempo, que avanza hacia debajo de la página. La dimensión horizontal muestra los roles de clasificador que representan objetos individuales en la colaboración. Cada rol de clasificador se representa mediante una columna vertical-línea de vida. Durante el tiempo que dura una activación de un procedimiento en el objeto, la línea de vida se dibuja cono una línea doble. Se muestra un mensaje como una flecha desde la línea de vida de un objeto a la del otro. Las flechas se organizan en el diagrama en orden cronológico hacia abajo.

Una activación es la ejecución de un procedimiento (esto incluye el tiempo que esperan los procesos anidados para ejecutarse). Lo vemos representado por una línea doble que sustituye la parte de la línea de vida en un diagrama de secuencia.

V.2 Planificación del proceso de desarrollo

Consideraremos al proceso de desarrollo como el conjunto de actividades necesarias para obtener un producto software basados en los requisitos de un cliente. Estas actividades se distribuyen en un diagrama, en el cual en la columna vertical tenemos los flujos de trabajo fundamentales (ciclo de vida del software) y en la parte horizontal están

distribuidas las fases del proceso de desarrollo unificado. Esto significa que el ciclo de vida tiene lugar sobre las cuatro fases.

V.2.1 Flujo de trabajo del ciclo de vida del software

Éste se divide en cinco partes: Captura de requisitos, análisis, diseño, implementación y prueba. Los cuales se describen a continuación. [Jacobson *et al*, 2000]

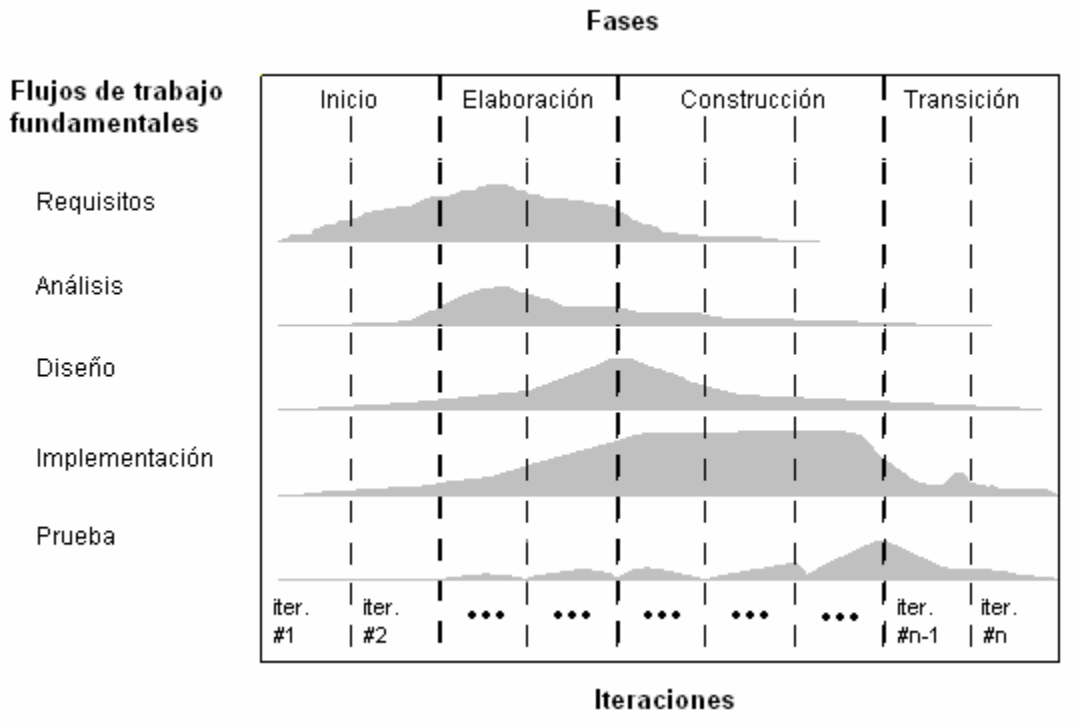


Figura V.1 *Flujo de trabajo del ciclo de vida del software*

V.2.1.1 Captura de requisitos

Ésta es una parte complicada, ya que en esta etapa se debe averiguar lo que se debe construir y debido a que los desarrolladores no serán los usuarios finales (generalmente), los desarrolladores deben tratar de entender lo que los usuarios desean, lo cual la mayoría de las veces no es algo trivial, ya que, aunque los usuarios pueden saber lo que quieren, no saben como expresarlo. Para ayudarse, los desarrolladores utilizan un modelo del dominio.

V.2.1.1.1 Modelo del dominio

Un modelo del dominio contiene los tipos más importantes de objetos en el contexto del sistema. Los objetos del dominio representan las “cosas” que existen o los eventos que suceden en el entorno en el que trabaja el sistema.

Muchos de los objetos del dominio o clases (hablando apropiadamente) pueden obtenerse de una especificación de requisitos o mediante la entrevista con los expertos del dominio.

El modelo del dominio se describe mediante diagramas de UML, especialmente diagramas de clase, aunque también se pueden utilizar casos de uso para entender el funcionamiento del negocio con el lenguaje del cliente.

V.2.1.2 Análisis

V.2.1.2.1 Introducción

El objetivo del análisis es tener una comprensión más precisa de los requisitos y una descripción de los mismos que sea fácil de mantener.

El análisis se describe utilizando el lenguaje de los desarrolladores, por tanto tiene un mayor formalismo y ayuda a razonar sobre el funcionamiento interno del sistema.

V.2.1.2.2 El papel del análisis en el ciclo de vida del software

Como se ve en la figura V.1, las iteraciones iniciales se centran en el análisis, por tanto un buen análisis es garantía de una buena arquitectura. [Jacobson *et al*, 2000]

V.2.1.3 Diseño

V.2.1.3.1 Introducción

En el diseño modelamos el sistema y encontramos su forma (incluida la arquitectura) para que soporte todos los requisitos. Los propósitos del análisis son comprender aspectos relacionados con los lenguajes de programación, componentes reutilizables, sistemas operativos y tecnologías de distribución, de interfaz de usuario entre otras. Parte de los propósitos es también descomponer el trabajo de implementación en partes que sean más pequeñas y manejables.

V.2.1.3.2 El papel del diseño en el ciclo de vida del software

Nos enfocamos en el diseño como final de la fase de elaboración y el comienzo de las iteraciones de implementación.

El diseño es una parte importantísima en el desarrollo, ya que sirve de enlace entre el análisis y la implementación, esto es, define elementos propios del lenguaje que utilizaremos, así como estructuras que intervendrán en la implementación, basándonos en el análisis previamente hecho. [Jacobson *et al*, 2000]

V.2.1.4 Implementación

V.2.1.4.1 Introducción

En la implementación empezamos con el resultado del diseño e implementamos el sistema en términos de componentes, o sea, código fuente, scripts, ejecutables, etc.
En esta etapa se codifican las clases y subsistemas encontrados durante el diseño.

V.2.1.4.2 El papel de la implementación en el ciclo de vida del software

La implementación es el centro de atención durante las iteraciones de construcción, aunque también se lleva a cabo trabajo de implementación durante la fase de elaboración y durante la fase de transición, en este caso para corregir los detalles que surgen con las versiones beta. [Jacobson *et al*, 2000]

V.2.1.5 Prueba

V.2.1.5.1 Introducción

En el flujo de trabajo de la prueba verificamos el resultado de la implementación probando cada construcción, incluyendo tanto construcciones internas como intermedias, así como las versiones finales del sistema que serán entregadas a los “beta testers”.

V.2.1.5.2 El papel de la prueba en el ciclo de vida del software

Las pruebas se llevan a cabo sobre todo cuando una construcción (resultado de la implementación) es sometida a pruebas de integración y/o de sistema. Esto quiere decir que las pruebas se centran en la fase de elaboración cuando se cuenta con una base de la arquitectura, y en la fase de construcción cuando una buena parte del sistema está implementado. En el caso de la fase de transición, las pruebas se centran en la corrección de defectos. [Jacobson *et al*, 2000]

V.2.1.6 Desarrollo iterativo e incremental

V.2.1.6.1 Introducción

Un sistema de software pasa por varios ciclos de desarrollo a lo largo de su vida. Cada uno de estos ciclos da como resultado una nueva entrega del producto a clientes y usuarios, pudiendo ser la primera de estas entregas, con toda seguridad la más difícil.

La base del proceso de desarrollo unificado, es la iteración con distintos niveles de enfoque, con el fin de obtener mejores productos al final de cada iteración.

V.2.1.6.2 El flujo de trabajo de iteración genérica

El flujo de trabajo de iteración genérica incluye los cinco flujos de trabajo: requisitos, análisis, diseño, implementación y prueba, en cada fase.

El primer paso hacia la división del proceso de desarrollo de software consiste en separar las partes en cuatro fases: inicio, elaboración, construcción y transición. Cada una de las fases se divide entonces en una o más iteraciones.

V.2.1.6.2.1 Fase de Inicio

El objetivo principal de esta fase es decidir si se sigue adelante con el proyecto. Esto significa que establece la viabilidad. [Jacobson *et al*, 2000]

V.2.1.6.2.2 Fase de elaboración

Esta fase lleva al estudio del sistema propuesto al punto de planificar la fase de construcción con gran precisión (la arquitectura) y de hacer una estimación de costes con precisión. Aquí (en esta fase) hacemos un estudio de factibilidad. [Jacobson *et al*, 2000]

V.2.1.6.2.3 Fase de construcción

El objetivo de esta fase es obtener un producto listo en versión beta para ser sometido a pruebas. En esta fase es donde se ocupa más personal que en cualquier otra. Esta fase requiere un mayor número de iteraciones que las fases anteriores. [Jacobson *et al*, 2000]

V.2.1.6.2.4 Fase de transición

Esta fase inicia con la entrega de un software (versión beta) con un funcionamiento inicial para que los usuarios lo evalúen. En esta etapa intervienen procesos como: aconsejar al cliente respecto al producto entregado (hardware, sistemas operativos, actualizaciones, protocolos de comunicaciones, etc), preparar manuales y documentos para la entrega del producto, ajustar el software (puesta a punto), corregir defectos y modificar el software al encontrar problemas no previstos. [Jacobson *et al*, 2000]

V.2.1.6.2.5 La iteración genérica

Una de las grandes ventajas de construir software utilizando un enfoque que use múltiples fases y un desarrollo iterativo e incremental es que nos permite equilibrar la asignación de recursos y de tiempo a lo largo del ciclo de vida.

V.3 Definición del problema

Este apartado corresponde a los requisitos, entre los que podemos mencionar:

Se quiere desarrollar un sistema para que el usuario pueda crear sus propios documentos WML (para visualizar en su teléfono celular) de manera sencilla, por tanto, el sistema a desarrollar debe:

- Ser fácil de manejar
- Contar con un entorno que no necesite de antecedentes en programación
- Ser gráfico
- Permitir ver el código del documento
- Contar con botones de desarrollo explícitos

- Tener un menú que especifique claramente su función
- Mostrar el resultado del desarrollo para ver que coincide con lo que se espera

V.4 El editor WML

V.4.1 Fase de inicio

- El editor permite que se interactúe con el menú y los botones
- Se contará con un área de texto, que es donde se verá el código
- Tendrá una área para mostrar los errores

V.4.2 Fase de elaboración

- El editor deberá tener la facilidad de escribir código WML de manera rápida y correcta.
- Deberá tener la particularidad de proporcionar código WML al crear un documento nuevo, de manera tal que tenga un patrón para generar nuevos documentos con mayor rapidez.
- Tendrá también las principales opciones de edición (cortar, copiar, pegar) para una mayor rapidez de escritura de código
- El editor deberá tener un acceso rápido al emulador, de manera tal que se pueda tener una vista previa de cómo se verá el documento en un dispositivo WAP.
- El editor deberá contar con ayuda sobre el WML

V.4.3 Fase de construcción

- El editor deberá enviar posibles mensajes de error en la sintaxis del código.
- Deberá proporcionar cambio de color de acuerdo a las diferentes sentencias que se coloquen, de manera que se tenga una mayor facilidad de poder leer y escribir el código.
- Tendrá la posibilidad de utilizar diferentes documentos WML dentro de una misma ventana.
- Se mantendrá la separación de documentos por medio de pestañas
- Se tendrá una separación por pestañas en el área del debugger
- El usuario deberá tener acceso a los diferentes elementos ya sea por clasificación dentro de un menú o de manera directa
- Se definirán colores para distinción de palabras específicas propias de WML.
- El movimiento entre pestañas del editor se realizará de la misma manera en el área del debugger. Existiendo una correspondencia directa entre los índices seleccionados de los *tabbedpane* superior (área de escritura) y el inferior(debugger) así como sus eventos.

V.5 El emulador WML

V.5.1 Fase de inicio

- Deberá tener una interfaz gráfica que nos permita ver el resultado de lo desarrollado.
- Deberá comprobar que lo que se escribió es correcto, si no lo es, informar cuál es el error.

V.5.2 Fase de elaboración

- El emulador deberá tener la posibilidad de actuar como un dispositivo WAP.

V.5.3 Fase de construcción

- Deberá tener, al menos, los elementos más básicos de WML, como son el input, barajas(decks), option, select y acciones.

V.6 Interacción del editor y emulador WML

- Deberá mostrar el código en el área de texto definida para eso
- Mostrará el trabajo por medio de las opciones del menú
- Debe permitir guardar el trabajo recientemente realizado
- Debe permitirme abrir un trabajo realizado anteriormente

V.7 El entorno de programación del editor y emulador WML

- El editor tendrá la opción de llamar al emulador siempre y cuando, el código sea correcto.
- Deberá utilizarse un parser para revisar y evaluar el código escrito en el editor.
- En caso de existir, se deberán mostrar los errores del código en la zona del editor creada para esa función (el debugger).
- Si no hay errores, el emulador mostrará la apariencia del documento editado en ese momento.

Un aspecto fundamental de este sistema es el desarrollo del analizador léxico y del analizador sintáctico. Dichos programas se realizarán con JavaCC, un poderoso meta-compilador que combina las funciones y operabilidad de Java con una gran facilidad en el manejo de tokens y reglas de producción.

V.7.1 JavaCC

V.7.1.1 Fundamentos de la Construcción de Compiladores

Los lenguajes de Programación son usualmente divididos en:

- Lenguajes Compilados

- Lenguajes interpretados

Los conceptos que comentaremos aplican de igual manera para los lenguajes compilados como para los lenguajes interpretados. Utilizaremos de manera más frecuente la palabra compilador, pero de igual manera los conceptos aplican para el intérprete.

Los compiladores deben realizar tres tareas principales con el texto que se presenta en un programa (código fuente).

1. Análisis Léxico
2. Análisis Sintáctico
3. Generación de código o ejecución.

La mayor parte de los compiladores centran su trabajo alrededor de las tareas 1 y 2, los cuales involucran el entendimiento del código fuente del programa y se aseguran de que se encuentre sintácticamente correcto.

V.7.1.2 Análisis Léxico

El análisis léxico realiza un vistazo rápido al código fuente del programa y lo divide en *tokens.* Un token es una secuencia de caracteres de entrada que tienen un significado basado en la sintaxis deseada. El primer paso en la construcción de un parser es el extraer los tokens del flujo de entrada. Esto generalmente involucra la especificación de los tokens en alguna forma de expresiones regulares. La extracción de tokens también es conocida como escaneo o *lexing* (por análisis léxico).

Como ejemplos de tokens tenemos las palabras claves (palabras reservadas del lenguaje), puntuación, literales como números y cadenas. Los no tokens incluyen espacios en blanco, que son usualmente ignorados pero usados para separar comentarios y tokens.

V.7.1.3 Análisis sintáctico (parsing)

Durante el análisis sintáctico, el parser extrae el significado del código fuente del programa asegurándose de que el programa sea sintácticamente correcto y construyendo una representación interna del programa.

La teoría de los lenguajes de computadora habla de programas, gramática y lenguajes. En este sentido, un programa es una secuencia de tokens. Una literal es un elemento básico de lenguaje de computadora que no puede ser reducido. Una gramática define las reglas para construir programas sintácticamente correctos. Solamente los programas que manejan las reglas definidas en una gramática son correctos. El lenguaje es simplemente el conjunto de todos los programas que satisfacen todas las reglas gramaticales.

Durante el análisis sintáctico un compilador examina el código fuente del programa respecto a las reglas definidas en la gramática del lenguaje. Si cualquier regla del programa es violada, el compilador despliega un mensaje de error. Mientras examina el programa, el compilador crea una representación interna fácilmente procesada del programa.

Las reglas gramaticales de un lenguaje pueden ser especificadas de manera no ambigua y enteramente con la notación BNF (Backus-Naur-Form). Esta notación define la gramática en términos de reglas de producción. Una producción BNF es un conjunto de secuencias de tokens que tienen un significado basado en la sintaxis deseada, así mismo, una regla de producción indica que un elemento de la gramática (ya sean literales o elementos compuestos) puede ser compuesto por otros elementos de la gramática. Las literales que son irreducibles, crean palabras clave o fragmentos estáticos de texto del programa. Los elementos compuestos se derivan de la aplicación de las reglas de producción. Las reglas de producción tienen el siguiente formato general:

ELEMENTO_GRAMATICA := lista de los elementos de la gramática
| lista alterna de los elementos de la gramática

Como un ejemplo, veamos las reglas gramaticales para un pequeño lenguaje que describe las expresiones aritméticas básicas.

```
expr := numero
        | expr '+' expr
        | expr '-' expr
        | expr '*' expr
        | expr '/' expr
        | '(' expr ')'
        | - expr
numero := digito+ ('.' Digito+)?
digito := '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
```

Tres reglas de producción definen los elementos de la gramática:

- expr
- numero
- digito

El lenguaje definido por la gramática nos permite especificar las expresiones gramaticales. Una *expr* es tanto un número como uno de los cuatro operadores aplicados a dos *expr*'s, una *expr* dentro de un paréntesis, o una *expr* negativa. Un *numero* es un número de coma flotante con una fracción decimal opcional. Definimos un *digito* como uno de los dígitos decimales que nos son familiares.

Los tokens y las producciones, son sólo la mitad de la historia. Determinar la definición correcta de los tokens y las producciones para la gramática es una tarea larga, pero no es la única tarea que deseamos realizar. Probablemente, no se está desarrollando un parser por la pura diversión de programar, probablemente se está desarrollando una aplicación de parser funcional, así que algo útil puede ser realizado con los datos del flujo de entrada. Por ejemplo, la cadena "2*3+4" debería ser evaluada y entregarnos un entero con el valor de 10.

El siguiente paso es determinar qué acciones semánticas realizar una vez que los tokens o las producciones hayan coincidido. No todos los tokens o las producciones deben tener acciones semánticas asociadas con ellos, sólo aquellas que tienen algún sentido en específico para la gramática. Por ejemplo, el objetivo de la gramática puede permitir

tokens de cadena que contengan incluidos caracteres de escape (como los tiene la gramática de Java).

Una acción semántica común es "no-escapar" a los caracteres de escape incluidos inmediatamente. Obviamente, esto no sería necesario para los tokens de tipo boolean (*true* y *false*).

En general, éstos son los pasos que necesitamos seguir para escribir una aplicación que realice un *parsing*:

1. Entender la gramática
2. Construir expresiones regulares que describan las cadenas de tokens.
3. Construir producciones que definan la sintaxis válida de la gramática.
4. Agregue manejo semántico mientras desarrolla.
5. Pruebe, prueba y pruebe aún más.

Los parsers son muy complejos y bastante difíciles de implementar correctamente a la primera vez. Un modelo de desarrollo incremental debe ser considerado. También, asegúrese de tener un suministro de datos disponible para la prueba. Un suministro de datos listo rendirá atractivos dividendos durante desarrollo.

V.7.1.4 Generación de código o ejecución

Una vez que el parser realiza exitosamente su labor, existe una representación interna que puede ser procesada fácilmente por el compilador. A partir de este momento resulta relativamente fácil generar código de máquina a partir de la representación interna lo que se conoce como compilación o ejecutarla directamente a partir de la representación interna conocido como interpretación.

V.7.1.5 JavaCC y la Generación del Parser

JavaCC es un generador de parsers y de analizadores léxicos. Los parsers y analizadores léxicos son componentes de software para tratar con la entrada de flujos de caracteres. Los compiladores e intérpretes incorporan analizadores léxicos y parsers para descifrar archivos que contienen programas, sin embargo pueden usarse en una gran variedad de aplicaciones.

Pero, ¿qué son los analizadores léxicos y los parsers? Los analizadores léxicos pueden dividir una secuencia de caracteres en subsecuencias llamadas tokens, así mismo clasifican dichos tokens. Consideremos un pequeño programa de lenguaje C.

```
int main(){
        return 0;
}
```

El analizador léxico de un compilador de C dividiría esto en la siguiente secuencia de tokens.

“int”, ” ”, ”main”, ”(“, ”)”,
“ ”, ”{“, ”\n”, ”\t”, ”return”
“ ”, ”0”, ” ”, ”;”, ”\n”,
“}”, ”\n”, ” ”

El analizador léxico también identifica el tipo de cada token; en nuestro ejemplo la secuencia de tokens puede ser:

KWINT, ESPACIO, ID, OPAR, CPAR,
ESPACIO, OBRACE, ESPACIO, ESPACIO, KWRETURN,
ESPACIO, OCTALCONST, ESPACIO, SEMICOLON, ESPACIO,
CBRACE, ESPACIO, EOF

El token del tipo EOF representa el fin del archivo original. La secuencia de tokens es pasada al parser. En el caso de C, el parser no necesita todos los tokens; en nuestro ejemplo, aquellos clasificados como ESPACIO no son pasados al parser. Después el parser analiza la secuencia de tokens para determinar la estructura del programa. A menudo en los compiladores los parsers entregan como salida un árbol que representa la estructura del programa. Luego este árbol sirve como entrada para los componentes del compilador responsables del análisis y de la generación de código. Consideremos un enunciado sencillo en un programa

Fahrenheit = 32.0 + 9.0 * Celsius / 5.0;

El parser analiza el enunciado de acuerdo a las reglas del lenguaje y produce un árbol.
El analizador léxico y el parser también son responsables para la generación de mensajes de error, si la entrada no conforma a las reglas léxicas y sintácticas del lenguaje.

JavaCC por sí mismo no es un parser o analizador léxico sino más bien un *generador.* Esto significa que entrega un analizador léxico y un parser de acuerdo a la especificación leída de un archivo. JavaCC produce un analizador léxico y un parser escrito en Java.

Los parsers y los analizadores léxicos tienden a ser componentes largos y complejos. Un ingeniero de software que escribe analizadores léxicos y parsers eficientes directamente en Java tiene que considerar de manera muy cuidadosa las interacciones entre las reglas. Por ejemplo, en un analizador léxico para C, el código para tratar con constantes enteras y de punto flotante no puede ser separado, a partir de que una constante entera comienza de la misma manera que una constante de punto flotante. Usando un generador de parsers como JavaCC, las reglas para las constantes enteras y las constantes de punto flotante están escritas de manera separada y la comunión entre ellas es extraída durante el proceso de generación.

Este incremento de modularidad significa que los archivos de especificación son más fáciles de escribir, leer y modificar comparados con los archivos de Java escritos a mano. Al utilizar un generador de parsers como JavaCC, el ingeniero de software puede ahorrar mucho tiempo y puede producir componentes de software de mejor calidad.

V.7.1.6 JavaCC en toda su gloria

JavaCC ofrece un excelente juego de herramientas para generar las clases del parser en Java. JavaCC genera parsers recursivos descendentes. La naturaleza *top-down* de

JavaCC le permite ser usado por una gran variedad de gramáticas en comparación con otras herramientas tradicionales, como yacc y lex. Otra diferencia entre JavaCC y yacc/lex es que JavaCC contiene toda la información del parsing en un archivo. La convención es llamar a este archivo con la extensión .jj, estos archivos contienen varios elementos:

- Porciones de la clase del parser que será generada
- Porciones de la clase *token manager* que será generada
- Lista de tokens
- Lista de producciones BNF

Una vez que el archivo .jj está disponible, se corre la herramienta *javacc*, dicha herramienta está basada en Java y la tarea que realiza es transformar el contenido del archivo .jj en un conjunto de clases basadas en Java que en conjunto se convierten en el parser. Dichas clases pueden ser compiladas con *javac* e incluirse en algún código base.

V.7.1.6.1 Definiendo Tokens

En JavaCC existen 4 tipos principales de Tokens. El tipo más común de token, está definido por la palabra clave TOKEN, define una secuencia de caracteres que forman un objeto Token válido. Los Tokens son objetos usados por el parser para formular producciones. Por ejemplo, el código siguiente muestra la definición de un número.

```
TOKEN: {
   <PS_ENTERO: (<SIGNO> | <DIGITO>) (<DIGITO>)*>
|  <PS_REAL: (<TIPO1_REAL> | <TIPO2_REAL>) >
|  <#TIPO1_REAL: (<SIGNO> | <DIGITO>) (<DIGITO>)* <DECIMAL>(<DIGITO>)*>
|  <#TIPO2_REAL: <DECIMAL> (<DIGITO>)*>
|  <#DIGITO: [“0” – “9”]>
|  <#SIGNO: [“-“, “+”]>
|  <#DECIMAL: [“.”]>
}
```

Estas definiciones identifican dos tipos de tokens válidos: PS_ENTERO y PS_REAL. Estos dos tipos de tokens pueden ser utilizados cuando se definan producciones válidas. Un número de definiciones de ayuda también se encuentran ilustradas en esta construcción. Las definiciones de ayuda tienen el prefijo “#” y nunca serán utilizados como tokens por sí mismos. Existen únicamente para ayudar en la creación de tokens verdaderos.

El segundo tipo de token es utilizado para saltarse aquello que carece de sentido como son los espacios en blanco que se encuentran alrededor de los tokens válidos. Como se muestra a continuación, este tipo de token es definido por la palabra clave de JavaCC SKIP. Las cadenas que coincidan con SKIP son simplemente desechadas.

```
SKIP : {
   “ “
|  “\r”
|  “\n”
|  “\t”
}
```

Existen dos tipos más de token (MORE y SPECIAL_TOKEN), para conocer más acerca de ellos recomendamos la documentación de JavaCC.

V.7.1.6.2 Definiendo Producciones

Hasta ahora, hemos hablado de la definición de tokens. Los tokens son los bloques de más bajo nivel para la construcción de cualquier parser. El siguiente paso es utilizar estos bloques para construir el árbol. El árbol contiene todas las combinaciones permitidas de tokens por una gramática. Estas combinaciones están definidas como producciones en un archivo .jj.

JavaCC ofrece cuatro tipos de producciones. Ya hemos cubierto uno de ellos, la producción de la expresión regular que se usa para producir objetos del tipo Token. A continuación, nos concentraremos en producciones BNF, las cuales permiten a diseñadores especificar el árbol del parser. Las producciones BNF se especifican como métodos normales de Java que pueden tomar parámetros arbitrarios de entrada y devuelve un valor arbitrario. Esta flexibilidad hace que las producciones BNF sean muy potentes y la clave del desarrollo de un parser.

Para entender cómo estructurar una producción, examinaremos una producción.

```
1. void wml():{ Token t; }
2. {
3. <FILE_HEADER> "<wml" (<XML_LAB><EQ><NMTOKEN>)? ">" ( doc_header() )?
4. (template_element())?  (card_element())+      "</wml>" <EOF>
5. }
```

La línea 1 define una producción llamada wml(). Esta producción nos dará como resultado un método completo al ser generado en el código del parser una vez que sea ejecutado el comando javacc sobre el archivo .jj.

Las producciones pueden aceptar parámetros de entrada y tener un valor de salida, aunque no sean indispensables para generarlas. La habilidad de realizar esto permite a los objetos invocantes pasar valores a través del árbol, así como también recibirlos. Las llaves y los parámetros al final de la línea 1 definen variables locales y código Java que será generado al principio del método. Si el método no requiere ninguna variable extra o código, las llaves pueden estar vacías.

La línea 2 comienza propiamente con la producción. El método será invocado cuando un objeto Token sea generado.

En JavaCC estas secuencias de cadenas tokens y producciones son conocidas como elecciones de expansión. Aunque no es estrictamente necesario tener código Java asociado a las elecciones de expansión, es usualmente requerido si no se quiere que el parser valide simplemente la sintaxis de la gramática. Después de todo, se puede requerir que el parser haga algo.

Generalmente, existe una secuencia de tokens, cadenas y producciones que define una secuencia válida para una producción como por ejemplo:

```
1.  float expresion() : { Token t1, Token t2 }
2.      t1.expresion() "+" t2.expresion(){
3.          return (Float .valueOf(t1.image) + Float.valueOf(t2.image)) ;
4.      }
5.      t1.expresion() "*" t2.expresion(){
6.          return (Float .valueOf(t1.image) * Float.valueOf(t2.image)) ;
7.      }
```

V.7.1.6.3 Manipulando Tokens

Hasta este punto, hemos examinado cómo montar expresiones regulares y producciones BNF. También hemos pasado por alto un tema muy importante: Las clases Token y Token Manager. Estas clases son generadas por JavaCC y son la base para el análisis léxico del parser.

La clase Token Manager es responsable de la parte del analizador léxico del parser. La clase Token Manager contiene métodos y atributos que pueden ser accedidas por las acciones semánticas asociadas con las expresiones regulares. El más importante de estos atributos es *image*, que contiene la secuencia de caracteres que coincide con la expresión regular en la cual se convertirá el objeto Token. El atributo *image* es un objeto del tipo *StringBuffer.* Usualmente es conveniente manejar este búfer de cadena antes de que el objeto Token sea creado. La línea 3 del siguiente código ilustra cómo el atributo *image* es manejado en una expresión regular.

```
1.  TOKEN: {
2.      <PS_STRING: (<LPAREN>) (<NOT_RPAREN>)* (<RPAREN>){
3.          image = expandEscapeCharacters(image);
4.      }
5.    | <#NOT_RPAREN:  (<ESC_LPAREN> | <ESC_RPAREN> | ~[")"] >
6.    | <#ESC_LPAREN: "\\(" >
7.    | <#ESC_RPAREN: "\\)" >
8.    | <#LPAREN: "(" >
9.    | <#RPAREN: ")" >
10. }
```

La razón más apremiante para manejar el atributo *image* del Token Manager es que la secuencia de caracteres coincidentes es una forma mutable. Una vez que la expresión regular completa su tarea y crea el objeto Token, la secuencia de caracteres ya no es mutable. Los objetos Token mantienen su propio atributo *image* público; como sea, este atributo es un objeto cadena inmutable para los Tokens y no un objeto *StringBuffer*. El atributo *image* de un objeto Token puede ser accedido o reemplazado en cualquier momento.

Adicionalmente, los Tokens mantienen un track de su posición en el flujo de entrada por medio de los atributos *beginLine, beginColumn, endLine* y *endColumn*, los tokens también mantienen un atributo *specialToken* para acceder a la lista de los tokens especiales ignorados por el parser.

V.7.1.6.4 AÑADIENDO MÉTODOS AL PARSER

Realizando un recuento, sólo han sido cubiertos dos tipos de producciones:

- Las expresiones regulares que especifican Tokens
- Las producciones BNF que especifican las elecciones de expansión que constituyen una gramática.

Los restantes dos tipos de producciones son usados para añadir código Java a las clases parser y Token Manager. Cuando JavaCC genera el parser, se generan muchas clases, las dos más importantes son las clases PARSER y TOKEN MANAGER. La clase Parser maneja las producciones BNF. Estas clases son nombradas de manera obligada por el constructor PARSER_BEGIN(NombredelParser) en el archivo .jj, por ejemplo, el nombre del parser WMLParser será WMLPARSER, que indica a JavaCC generar los códigos fuente WMLParser.java y WMLParserTokenManager.java.

La declaración de las producciones Token Manager, son identificadas por la palabra clave de JavaCC TOKEN_MGR_DECLS, dichas declaraciones insertan métodos y atributos a la clase Token Manager generada. Un ejemplo donde puede resultar útil pudiera ser insertando un método expandEscapeCharacters() en un Token Manager. Esta rutina estaría disponible para el código Java asociado con la expresión regular.

```
1. TOKEN_MGR_DECLS: {
2.     public String expandEscapeCharacters(String s) {
3.             // Realizar algo útil en el cuerpo del método
4.     }
5. }
```

Las producciones *Javacode* identificadas por la palabra clave de JavaCC JAVACODE, son mecanismos para construir gramáticas basadas en JavaCC. No siempre es fácil romper una gramática en un conjunto de producciones. De hecho en algunas ocasiones no es posible o deseable.

Las producciones *Javacode* permiten a los usuarios definir producciones que pueden hacer lo que quieran. En contraste con la declaración de producciones del Token Manager, las producciones *Javacode* son añadidas a la clase parser.

V.8 Alcance de la propuesta

El alcance de la propuesta definirá el ámbito de trabajo del sistema, esto es, definirá lo que es capaz de hacer, a sabiendas que aquellos aspectos funcionales no mencionados en este apartado no se contemplan dentro del sistema.

Se debe marcar una división dentro de los alcances: alcances del editor y alcances del emulador.

V.8.1 Alcances del editor

El editor estará comprendido como el área de escritura, los botones de acceso a los elementos, los botones de acceso rápido a las funciones de archivo (“abrir”, “guardar”, “guardar como” y “nuevo”), los botones de acceso a las funciones básicas de edición (“copiar”, “cortar”, “pegar”, “deshacer” y “rehacer”), el botón de acceso al emulador (“emular”), el área de despliegue de errores (debugger) y el menú de acceso (“archivo”, “edición”, “elemento”, “proyecto” y “ayuda”).

El editor funcionará de la siguiente manera. Al iniciar el sistema aparecerá un documento en blanco con los encabezados de un documento WML (básicos), la fuente del editor será clara y las palabras propias del WML estarán resaltadas en otro color. Al hacer clic sobre cualquiera de los botones de los elementos o seleccionando por medio del menú de elementos el elemento deseado, aparecerá sobre el área de escritura la sintaxis propia del elemento seleccionado. Antes de emular el proyecto, el sistema pedirá que lo guardemos con un nombre específico con extensión .wml, para hacer esto podemos hacer clic en el boton definido para este efecto o por medio del menú Archivo, se puede efectuar la misma acción presionando ctrl+g o ctrl+ s.

V.8.2 Alcances del emulador

El emulador se compone de una pantalla, con dos botones en la parte inferior de la misma.

El funcionamiento básico será de esta forma, después de presionado el botón de “emular”, se debe evaluar si el archivo ha sido guardado y no presenta ninguna modificación posterior al guardado del mismo, de ser así, entonces, se revisa el documento y si no hay errores se presenta el documento en el emulador.
Si existen errores se mostrarán en el área de despliegue de errores y no permitirá emular hasta que éstos sean corregidos.
No se soporta la conversión de caracteres con “escape”.

V.8.3 Alcances del SEEDWML

El sistema permitirá navegación entre cartas, manejo de variables, manejo de tablas, liga a otros archivos.

Referencias

[Blair et al, 1991] G. Blair, J. Gallagher, D. Hutchison, and D. Sheperd. *Object Oriented Languages Systems and Applications*. Halsted Press, New York, New York, 1991

[Booch, 1991] G. Booch. *Object-Oriented Design UIT Applications*. Benjamín/Cummins, Menlo Park, California,1991

[Booch, 1996] Grady Booch. *Análisis y diseño orientado a objetos con aplicaciones*. 2a edición, Addison Wesley Longman, México, 1996

[Booch, 1999] Grady Booch. James Rumbaugh, Ivar Jacobson. *El Lenguaje Unificado de Modelado*. Addison Wesley Iberoamericana, Madrid, 1999

[Budd,1986] T. Budd. *An Introduction to Object-Oriented Programming*. Addison-Wesley, Reading, Massachusetts, 1986

[ENSELING] Oliver Enseling, Build Your own languages with JavaCC

[IEEE,1983] IEEE, IEEE Standard Glossary of Software Engineering Terminology, The Institute of Electrical and Electronic Engineers, New York, New York, 1983

[Jacobson *et al*, 2000] Ivar Jacobson, James Rumbaugh, Grady Booch. *El proceso unificado de desarrollo de software*. Addison Wesley Iberoamericana, Madrid, 2000

[Rumbaugh *et al*, 2000] James Rumbaugh, Grady Booch, Ivar Jacobson. *El Lenguaje Unificado de Modeloado.Manual de Referencia*. Addison Wesley Iberoamericana, Madrid, 2000

[Wirfs-Brock *et al*,1990] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing Object-Oriented Software.* Prentice Hall, Englewood Cliffs, New Jersey,1990

[TOA] http://www.toa.com/pub/abstraction.html

CAPÍTULO VI
Documentación del Análisis y Diseño del Sistema

VI.1 Introducción a los casos de uso

El modelado con casos de uso es un tipo especializado de modelado estructural concerniente a la funcionalidad del sistema usualmente aplicamos modelado de casos de uso durante las actividades y los requerimientos, para capturar los requerimientos que definen qué es lo que un sistema debe de hacer. Típicamente el modelado de casos de uso empieza temprano en un proyecto y continúa a través del desarrollo del proceso del sistema.

Los casos de uso son realizados usualmente como un trabajo de equipo entre los usuarios y los analistas del sistema, en cuyas ideas se puede explorar y los requerimientos pueden ser definidos contra un tiempo.

Un diagrama de casos de uso puede ser considerado una tabla de contenidos para los requerimientos funcionales de un sistema.

Como ya se había comentado antes, estos diagramas describen la funcionalidad de un sistema y sus elementos son: un actor, un caso de uso y una asociación de comunicación.

Un actor es un usuario o un sistema externo con el cual el sistema que estamos modelando interactúa. Una instancia de un actor es un usuario específico o un sistema específico.

Figura VI.1 *Actor y Caso de Uso*

Debido a que los actores son externos al sistema e interactúan con éste, son ellos lo que definen el límite y alcance del sistema.

Un actor puede también representar un recurso, poseído por otro proyecto o que puede ser comprado en vez de ser construido.

Un caso de uso es un requerimiento funcional que está descrito desde la perspectiva de los usuarios del sistema. Un caso de uso está descrito como una secuencia de pasos, los cuales incluyen acciones desarrolladas, por un sistema e interacciones entre el sistema y los actores. Los casos de uso hacen frente a la pregunta de ¿cómo los actores interactúan con el sistema? Y describe las acciones que el sistema realiza. Una instancia de un caso de uso es generalmente llamada un escenario y es la realización de una secuencia específica de acciones e interacciones. Las instancias de los casos de uso, los

escenarios, enfrentan la pregunta de cómo el actor así como las instancias de los actores interactúan específicamente con un sistema y qué acciones específicas realiza el sistema como regreso o respuesta. Las instancias de los casos de uso comúnmente no se muestran en los diagramas de casos de uso sino que más bien son discutidos con los usuarios para entender mejor sus requerimientos. Frecuentemente tratar de clasificar cada paso o cada acción o interacción no es necesario más bien es más importante considerar un caso de uso como un diálogo entre actores y el sistema para entender cómo los actores interactúan con el sistema y entender también qué acciones son la responsabilidad del sistema. Algunas propiedades que podríamos notar de los casos de uso hasta el momento, podrían ser [Sinan, 1998]:

- El caso de uso capta una función visible para el usuario
- El caso de uso puede ser pequeño o grande
- El caso de uso logra un objetivo discreto para el usuario.

Fue Jacobson quien introdujo los casos de uso como elementos primarios de desarrollo de software y que además diseñó un diagrama para la representación gráfica de los casos de uso, los diagramas de caso de uso. Y éste ya es parte del UML. Además de actores y casos de uso contamos con relaciones entre los elementos, siendo también parte importante de los diagramas de casos de uso. Ya que la manera por medio de la cual se comunicaran los usuarios del sistema con el sistema y el sistema con otras partes de éste.

Tenemos dos tipos de vínculos especiales: *uses* y *extends* [Fowler, 1999].

- Se usa la relación *extends* cuando se tiene un caso de uso que es similar a otro pero que hace un poco más

- Y las relaciones *uses* ocurre cuando se tiene una porción de comportamiento que es similar en más de un caso de uso y no se quiere copiar la descripción de tal conducta.

Para hacerlo más claro se pueden aplicar las siguientes reglas [Fowler, 1999]:

- Utilizar *extends* cuando se describa una variación de conducta normal
- Emplearemos *uses* cuando se tenga un conjunto de características similares en más de un caso de uso.

Algo importante que cabe resaltar es que los vínculos *extends* y *uses* sirven para comunicarse únicamente entre casos de uso. Y su representación gráfica es una flechita con la palabra <<uses>> o <<extends>> junto a una línea punteada.

VI.2 Fundamentos de diagramas de clase

Las clases son un concepto general. Una clase define un tipo de objeto y sus características (incluyendo las características estructurales y las características de su comportamiento); las características estructurales definen lo que un objeto sabe de una clase, las características de comportamiento definen lo que un objeto de una clase puede hacer.

Una clase es un rectángulo de líneas bien definidas que puede estar dividido de manera vertical en tres compartimientos. El compartimiento superior mantiene el nombre de la clase, el segundo compartimiento (el compartimiento intermedio) almacena todos los atributos de la clase y el tercer compartimiento almacena todas las operaciones.

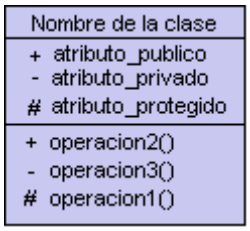


Figura VI.2 *Clase*

VI.2.1 Atributos

Los atributos son el elemento de datos mantenido por el objeto y su sintaxis es:

visibilidad nombre [multiplicidad orden] : tipo = valor inicial

visibilidad.- es opcional, no tiene un valor por default e indica cuándo un atributo es accesible desde fuera de la clase y puede ser alguno de los siguientes [Sinan, 2003]:

- \+ (visibilidad pública.- el atributo puede ser visto desde fuera de la clase)
- \- (visibilidad privada.- el atributo es inaccesible fuera de la clase)
- # (visibilidad protegida.- el atributo es accesible por las clases que tengan una relación de generalización a esta clase, pero sobre todo es inaccesible desde otras clases)

nombre.- es el nombre del atributo que estamos describiendo.

multiplicidad.- es opcional, tiene por default el valor 1 e indica el número de valores que un atributo puede tomar. Si un atributo tiene solamente un valor, entonces la multiplicidad, el orden y los corchetes cuadrados no se muestran; de otra manera la multiplicidad debe mostrarse como una cadena de la forma

[límite_inferior .. límite_superior]

en el cual un asterisco sencillo implica un rango ilimitado, por ejemplo:

[0 .. *]

nos indica que va de cero al infinito el número de valores.

orden.- es opcional, tiene un valor por default de *unordered* y es donde la multiplicidad es mayor a 1 para indicar si los valores de un atributo están ordenados o desordenados. Usa una de las siguientes palabras:

- *unordered*: indica que los valores no tienen orden
- *ordered*: indica que los valores son ordenados

tipo.- es opcional, no tiene valor por default e indica el tipo de dato que un atributo puede tomar. Si no tenemos definido un tipo para este atributo debemos omitir los dos puntos (:). El tipo de un atributo puede ser otra clase. Además UML nos ofrece los siguientes tipos de datos:

- *boolean*: valores de true o false
- *integer*: un valor entero
- *real*: número de punto flotante
- *string*: una secuencia de caracteres

valor inicial.- Es opcional e indica el valor inicial de un atributo. Por default un atributo no tiene valor inicial. Si no está definido un valor inicial debemos omitir el símbolo igual (=).

UML nos da la posibilidad de mostrar un atributo usando pseudocódigo o sintaxis propia de otro lenguaje (Java, C++, C# o algún otro lenguaje de programación)

VI.2.2 Operaciones

Es la especificación de un servicio provisto por el objeto. Cabe resaltar también que un método es el cómo un objeto de una clase realiza sus procesos, es la implementación de un servicio provisto por el objeto.
En las clases en UML, las operaciones se describen en el tercer compartimiento, usando la siguiente sintaxis:

visibilidad nombre_operación (lista_parámetros) : tipo_regreso

visibilidad.- no tiene valor por default e indica si puede ser accesible la operación fuera de la clase; puede ser alguna de las siguientes [Sinan, 2003]:

+ (visibilidad pública.- significa que la operación es accesible fuera de la clase)
- (visibilidad privada.- la operación es inaccesible desde fuera de la clase)
(visibilidad protegida.- la operación es accesible sólo por las clases que tengan una relación de generalización con esa clase, de otra manera es inaccesible desde otra clase fuera de ésta)

nombre_operacion.- es el nombre que le damos al método

lista_parámetros.- es opcional, no tiene un valor por default y es una lista separada por comas (,) que indica los parámetros que toman los valores que vamos a pasar o los recibidos de la operación. Cada parámetro es mostrado como una cadena de texto y tiene la siguiente sintaxis:

tipo_parámetro nombre_parámetro : tipo_dato = valor_default

tipo_parámetro.- es opcional, tiene un valor por default de "in" y puede ser uno de los siguientes:

in	Indica que el parámetro es de entrada solamente y no puede ser modificado por la operación.
out	Indica que el parámetro es únicamente de salida y puede ser modificado por la operación para comunicar información al cliente que invocó la operación.
inout	El parámetro es de entrada y puede ser modificado por la operación para comunicar información al cliente que invocó la operación.

nombre_parámetro.- es el nombre del parámetro

tipo_dato.- es opcional, no tiene un valor por default e indica el tipo de dato que un parámetro puede tomar, si no muestro un tipo para el parámetro, debo omitir los dos puntos (:). El tipo puede ser inclusive otra clase. UML nos ofrece los siguientes tipos de datos:

- *boolean*: puede tomar los valores de "true" o "false"
- *integer*: es un número entero
- *real*: es un número de punto flotante
- *string*: es una secuencia de caracteres.

valor_default: es opcional e indica cual sería el valor inicial de un parámetro. Por default, un parámetro no tiene valor inicial, así que si no se define un valor inicial debo omitir el símbolo igual.

tipo_regreso.- es opcional, no tiene valor por default e indica el tipo de dato que la operación regresa a quien lo llamó. Si no tenemos un tipo de retorno para una operación, debemos omitir el dos puntos (:). Las opciones para el tipo_regreso son las mismas que para el tipo_dato.

VI.2.3 Relaciones

Una relación es una conexión entre elementos.

VI.2.3.1 Dependencias

Una dependencia es una relación semántica entre dos o más elementos del modelo. Esta dependencia indica una situación en la cual un cambio en el elemento proveedor puede requerir un cambio o indicar un cambio en el significado del elemento cliente en la dependencia.

Los estereotipos que se usan con las dependencias son: become, bind, call, create, derive, extend, friend, import, include, instanceOf, instantiate, powertipe, send, trace, use.

VI.2.3.2 Generalización

Una generalización es una relación entre un elemento general (llamado superclase o padre) y un caso más específico de ese elemento (llamado subclase o hijo).

VI.2.3.3 Asociación

Una asociación es una relación estructural que especifica que los objetos de un elemento están conectados con los objetos de otro. Gráficamente, una asociación se representa como una línea continua que conecta la misma o diferentes clases. El fin de las asociaciones es representar relaciones estructurales.
Hay cuatro elementos que se le aplican a las asociaciones: Nombre, rol (papel), multiplicidad, agregación.

VI.2.3.3.1 Nombre
El nombre se usa para describir la naturaleza de la relación, para indicar la dirección del nombre con una flecha y con esto evitamos la ambigüedad en el significado.

VI.2.3.3.2 Rol
Toda clase que participa en una asociación tiene un rol específico. El rol es el papel que juega una clase frente a la que está del otro lado de la relación. Es importante aclarar que la misma clase puede jugar un rol diferente en otras asociaciones.

VI.2.3.3.3 Multiplicidad
La multiplicidad lo que nos indica es: cuántos objetos pueden conectarse a través de una instancia de una asociación.

VI.2.3.3.4 Agregación
Es una forma de asociación que especifica una relación todo-parte entre un agregado (el todo) y las partes que lo componen.

VI.3 Casos de uso y diagramas de clase para el editor WML

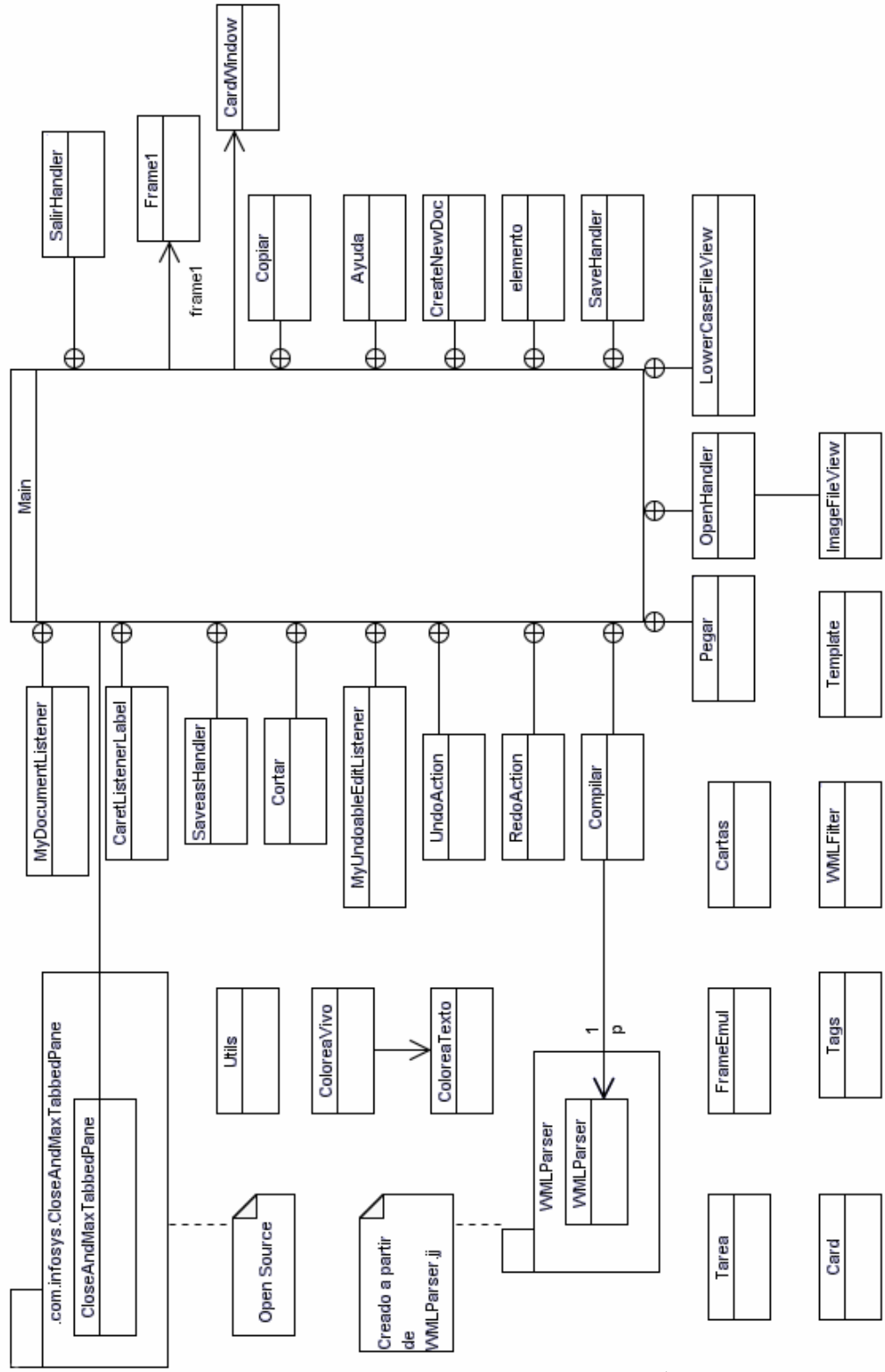


Diagrama VI.1 *Diagrama de clases del editor* [1]

[1] Para mayor información sobre los atributos y métodos de las clases, ver Apéndice A.

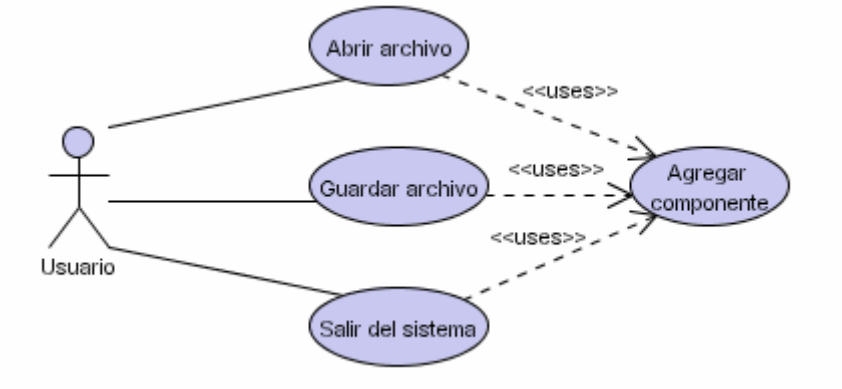


Diagrama VI.2 *Caso de uso de acciones generales sobre archivos y sistema*

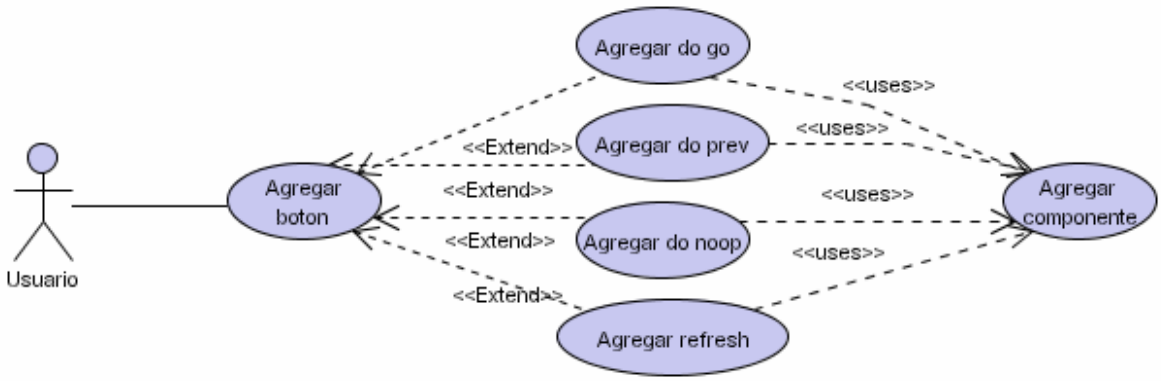


Diagrama VI.3 *Caso de uso de tareas*

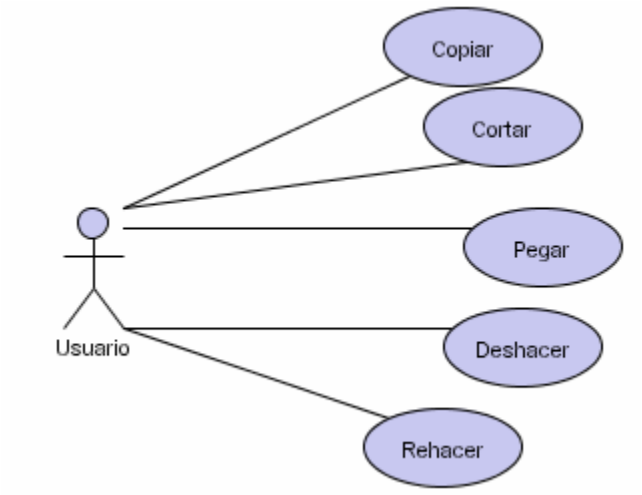


Diagrama VI.4 *Caso de uso de edición*

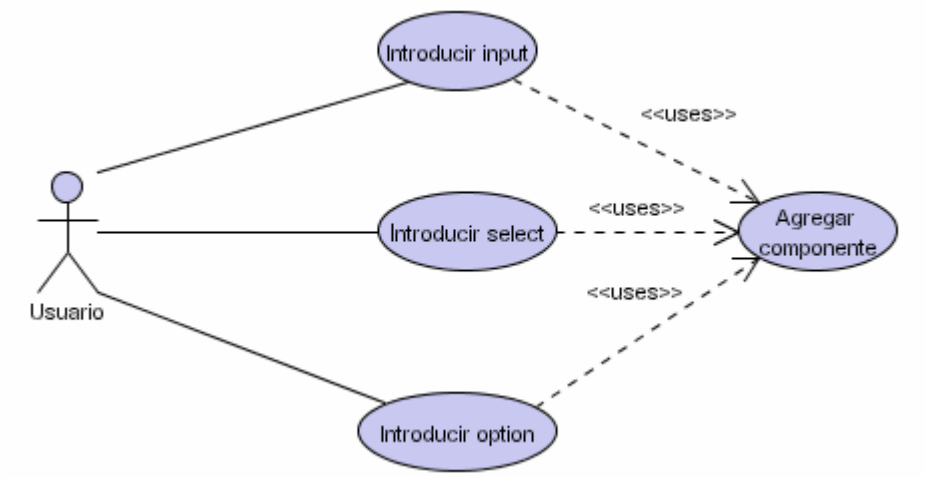


Diagrama VI.5 *Caso de uso para elementos dinámicos*

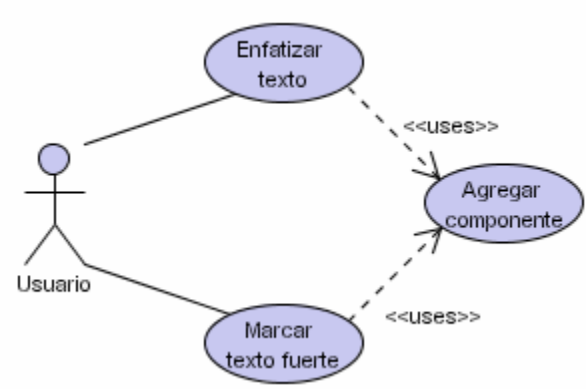


Diagrama VI.6 *Caso de uso de estilo de texto*

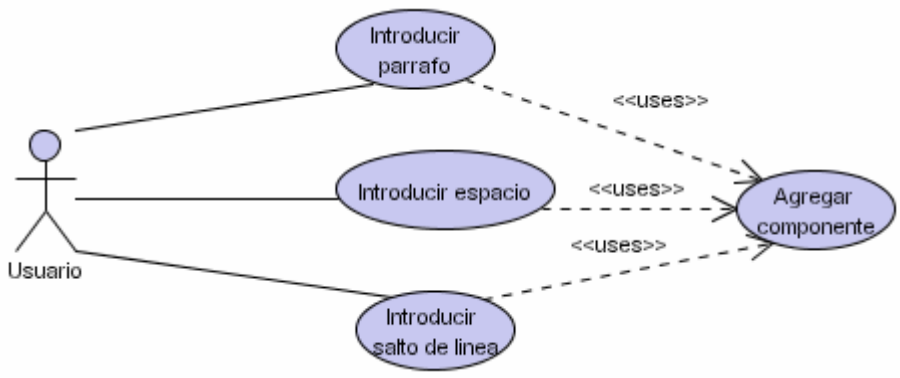


Diagrama VI.7 *Caso de uso de formato de texto*

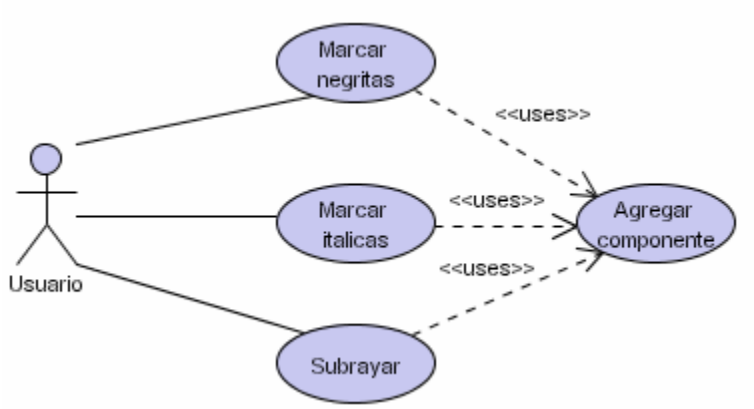


Diagrama VI.8 *Caso de uso de fuente*

Diagrama VI.9 *Caso de uso de Liga*

Diagrama VI.10 *Caso de uso de nueva card*

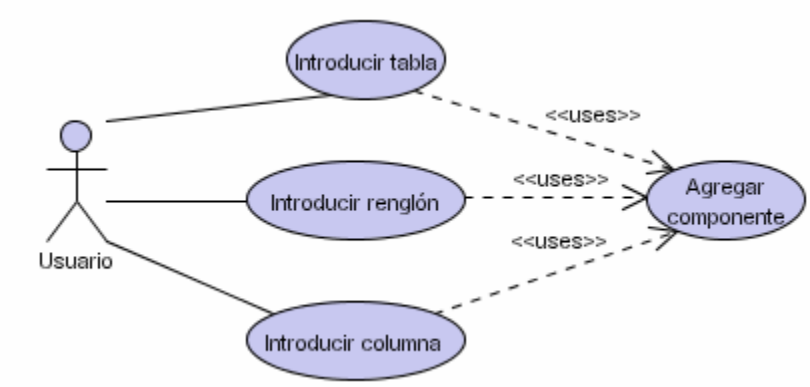


Diagrama VI.11 *Caso de uso de tabla*

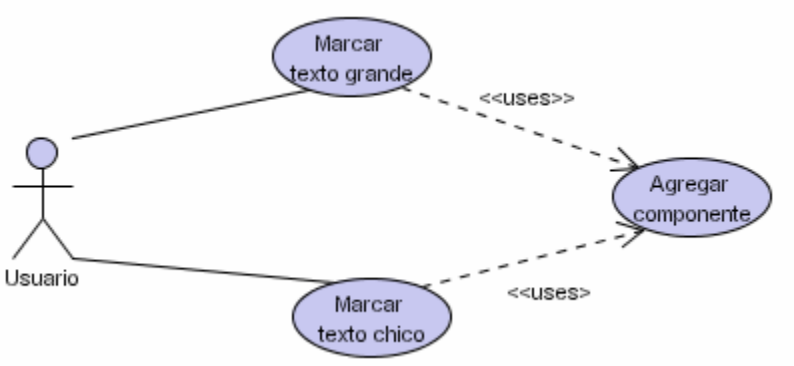


Diagrama VI.12 *Caso de uso de tamaño*

VI.4 Casos de uso y diagramas de clase para el emulador WML

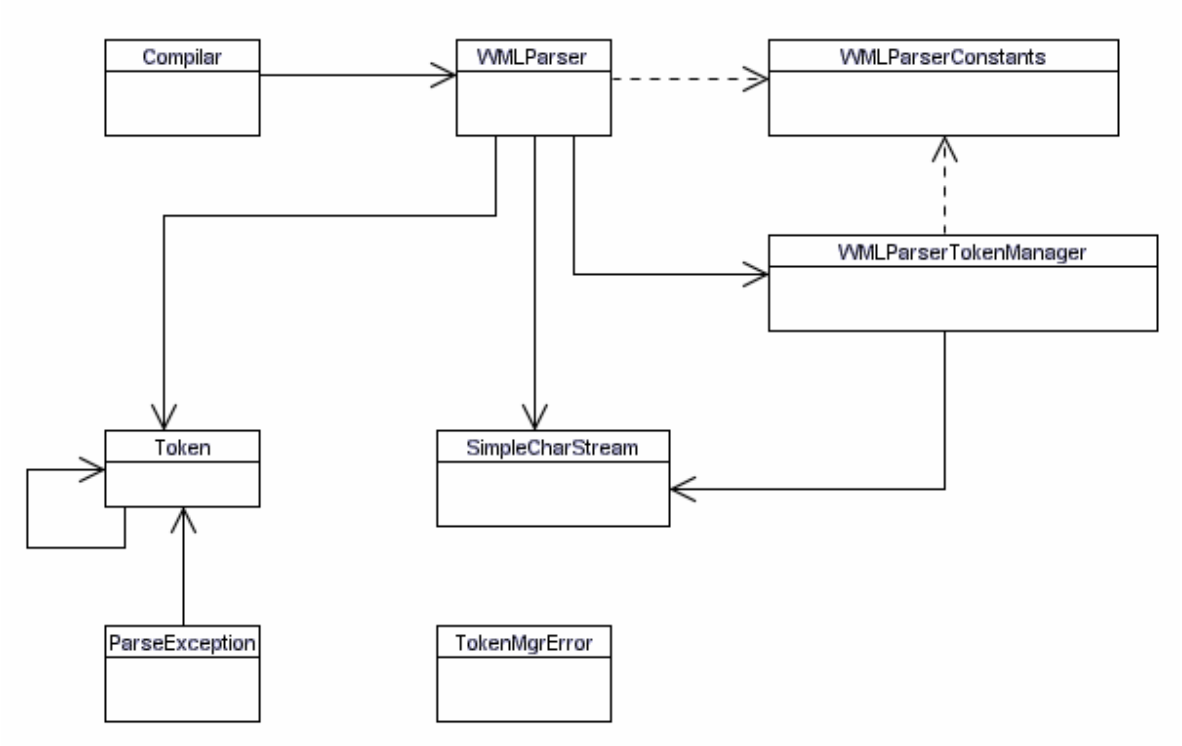


Diagrama VI.13 *Diagrama de clases del emulador*[2]

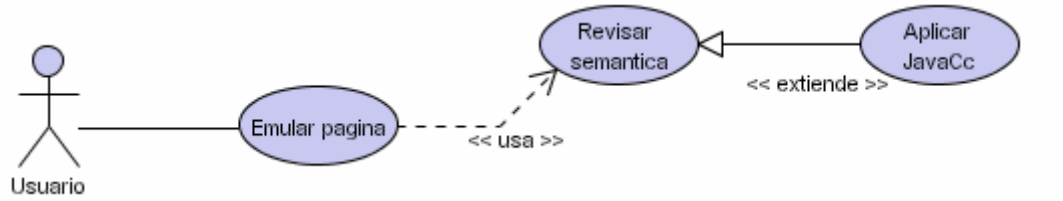


Diagrama VI.14 *Caso de uso emular*

[2] Para mayor información sobre los atributos y métodos de las clases, ver Apéndice A.

VI.5 Introducción a los diagramas de actividad y colaboración

VI.5.1 Diagrama de colaboración

Muestra cómo los elementos interactúan sobre el tiempo y de qué manera se relacionan, esto es, muestra una colaboración o una instancia de una colaboración. Mientras los diagramas de secuencias son orientados al tiempo y enfatizan el flujo total de una interacción, los diagramas de colaboración están orientados tiempo y espacio, y hacen énfasis en la interacción total, los elementos involucrados y sus relaciones.

Los diagramas de secuencia son especialmente útiles para interacciones complejas porque se leen de arriba abajo; en contraste, lo diagramas de colaboración son útiles para ver el impacto de una interacción sobre varios elementos, porque se puede colocar un elemento en un diagrama e inmediatamente ver todos los demás elementos con los cuales interactúa [Sinan, 2003].

Los elementos de los diagramas de colaboración son: roles de clase, objetos específicos, roles de asociación y las ligas específicas. Los objetos y los roles de asociación en un diagrama de colaboración se muestra con la misma notación que un diagrama de secuencia.

En el caso de los objetos tenemos:

Nombre_del_objeto : Nombre_de_la_clase

la otra forma de representar a un objeto es:

:nombre del objeto

Un rol de asociación se muestra usando la notación para relaciones, el nombre de la asociación se precede sólo por una diagonal (/) después el nombre del rol:

/ nombre_del_rol : nombre_asociación

En un diagrama de colaboración los objetos específicos se muestran conforme al rol de la clase y al de otros objetos y usan la misma notación que un diagrama de secuencias.

IV.5.1.1 La comunicación entre los objetos en un diagrama de comunicación

Una comunicación, un mensaje o un estímulo se muestra como una flecha adjunta a una relación que va del emisor al receptor. Una comunicación es etiquetada usando la misma notación que en un diagrama de secuencia, pero se requiere de un número de secuencia que se expresa usando un esquema de numeración especial conocido como “notación de punto”, en la cual un punto (o punto decimal) es usado entre diferentes niveles de comunicación.

VI.5.2 Diagramas de actividad

Describen las actividades y responsabilidades de los elementos que conforman un sistema.

Uno de los elementos de los diagramas de actividad, son las acciones (o actividades) y representan un elemento que cumple una responsabilidad. Hay varios tipos de acciones:

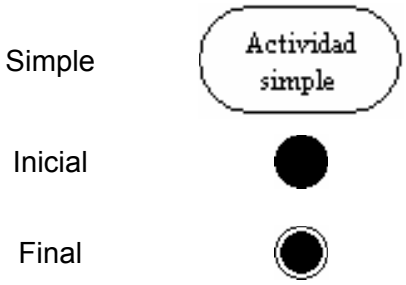


Figura VI.3 *Tipos de Acciones*

El control de flujo de las transiciones se lleva a cabo mediante flechas que van de una actividad a otra. Estas flechas son de línea continua ya que las transiciones del flujo de objetos son flechas con la línea punteada.

Usaremos las transiciones del flujo de objetos cuando un estado actualiza o produce un objeto como salida y la forma de representarlo es con la flecha punteada, de la actividad al objeto y a su vez, del objeto hacia la(s) actividad(es) con la(s) cual(es) va a interactuar.

Los carriles son divisiones que colocamos en el diagrama en el cual agrupamos actividades de la misma índole, su representación es una línea vertical.

Lo que se intenta agrupar con los carriles son actividades con responsabilidades similares o que tienen una interacción directa sobre el mismo tipo de actividad [Sinan, 1998].

Otro de los elementos son las decisiones. Las decisiones involucran seleccionar una transición en el control de flujo entre muchas transiciones, basados en una condición.
Su representación es:

Figura VI.4 *Condición*

Cuando hablamos de concurrencia, nos referimos a la selección simultánea de transiciones múltiples.

VI.6 Diagramas de actividad y colaboración del sistema integrado de editor y emulador WML

VI.6.1 SalirHandler actividad

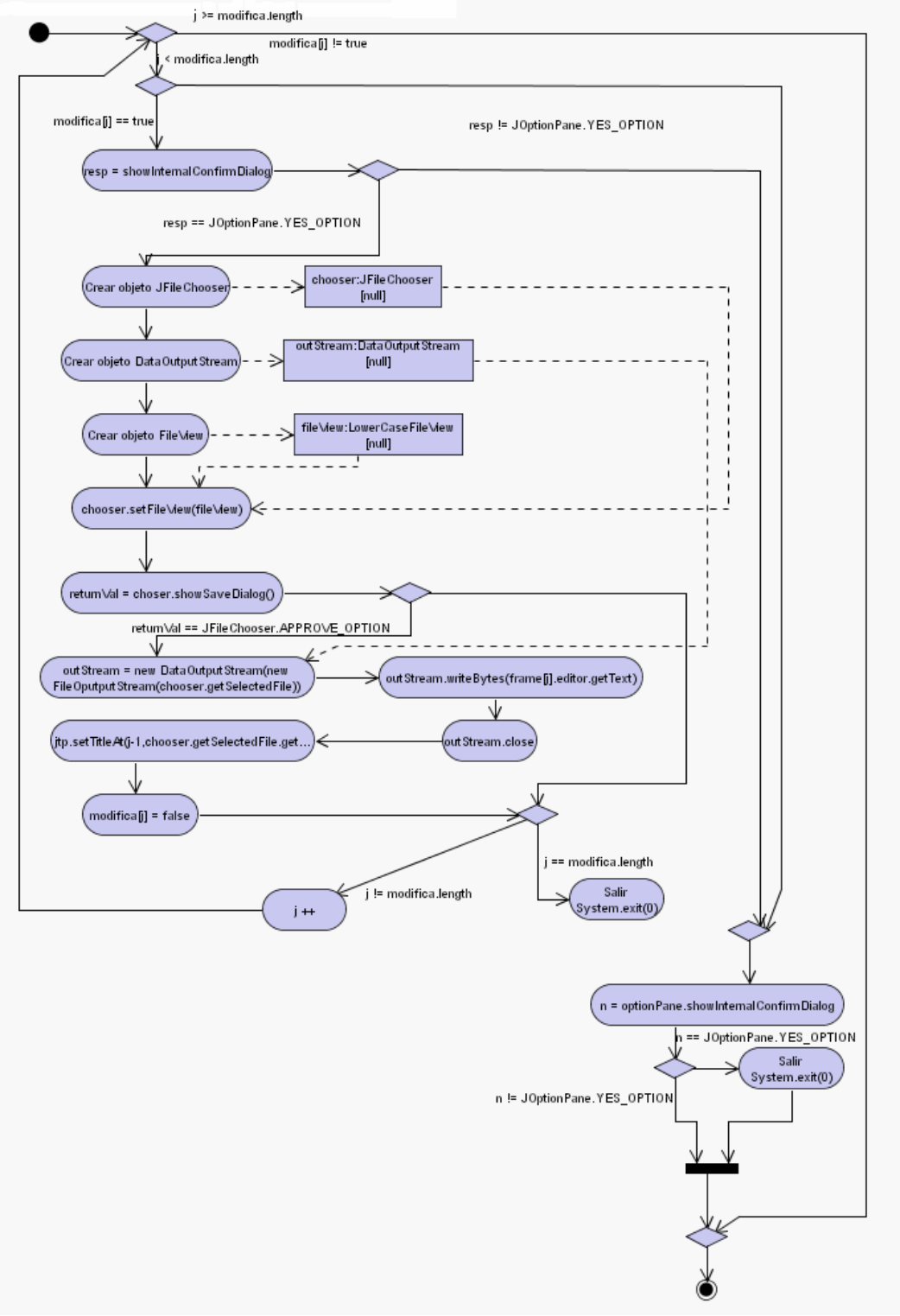


Diagrama VI.15 *Diagrama de actividad del evento SalirHandler*

VI.6.2 SalirHandler colaboración

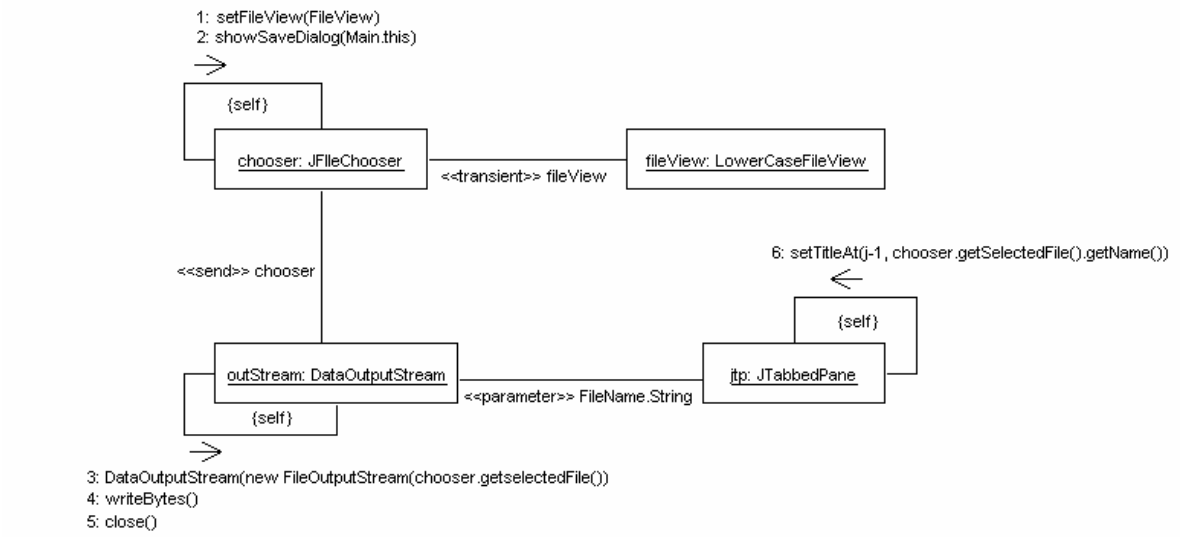


Diagrama VI.16 *Diagrama de colaboración del evento SalirHandler*

VI.6.3 OpenHandler Colaboración

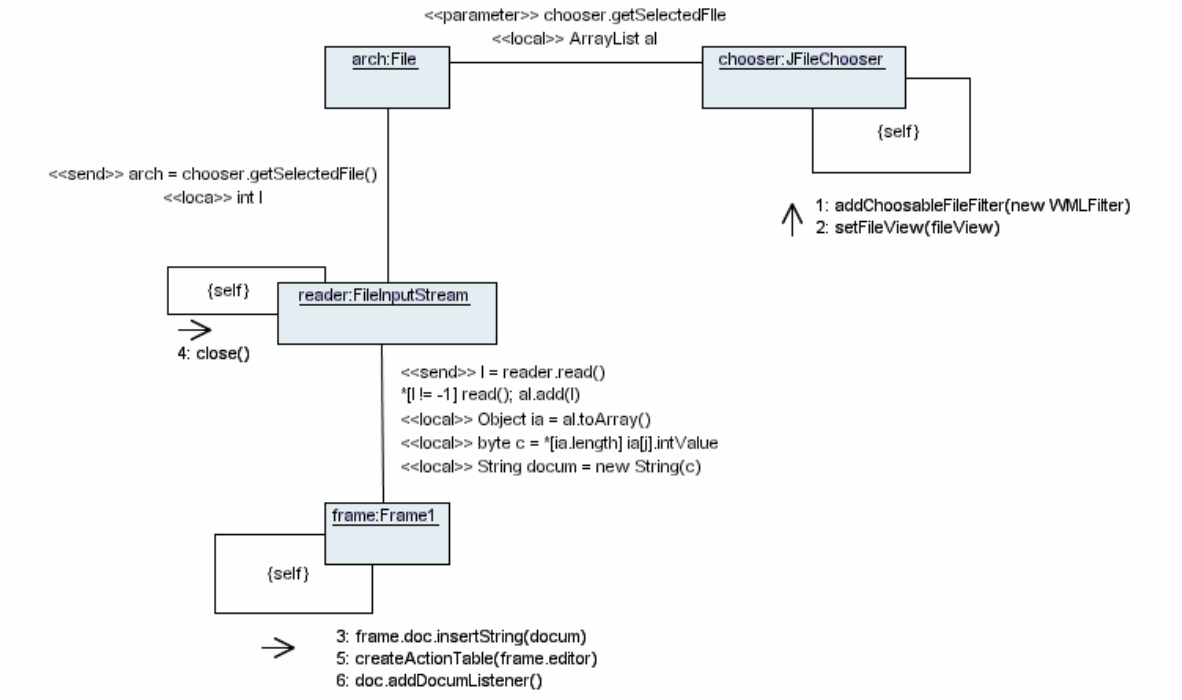


Diagrama VI.17 *Diagrama de colaboración del evento OpenHandler*

VI.6.4 OpenHandler actividad

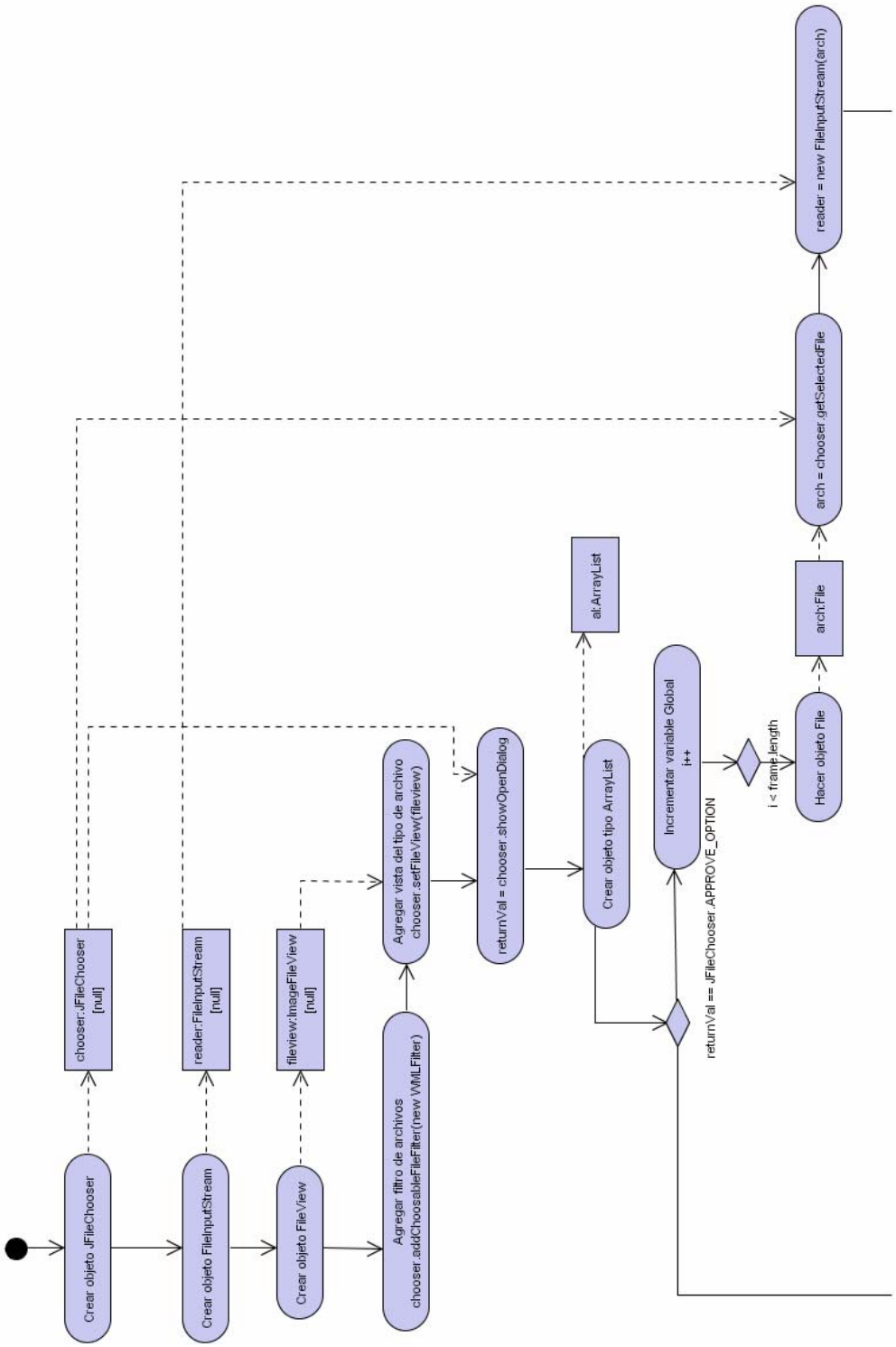


Diagrama VI.18 *Diagrama de actividad del evento OpenHandler*

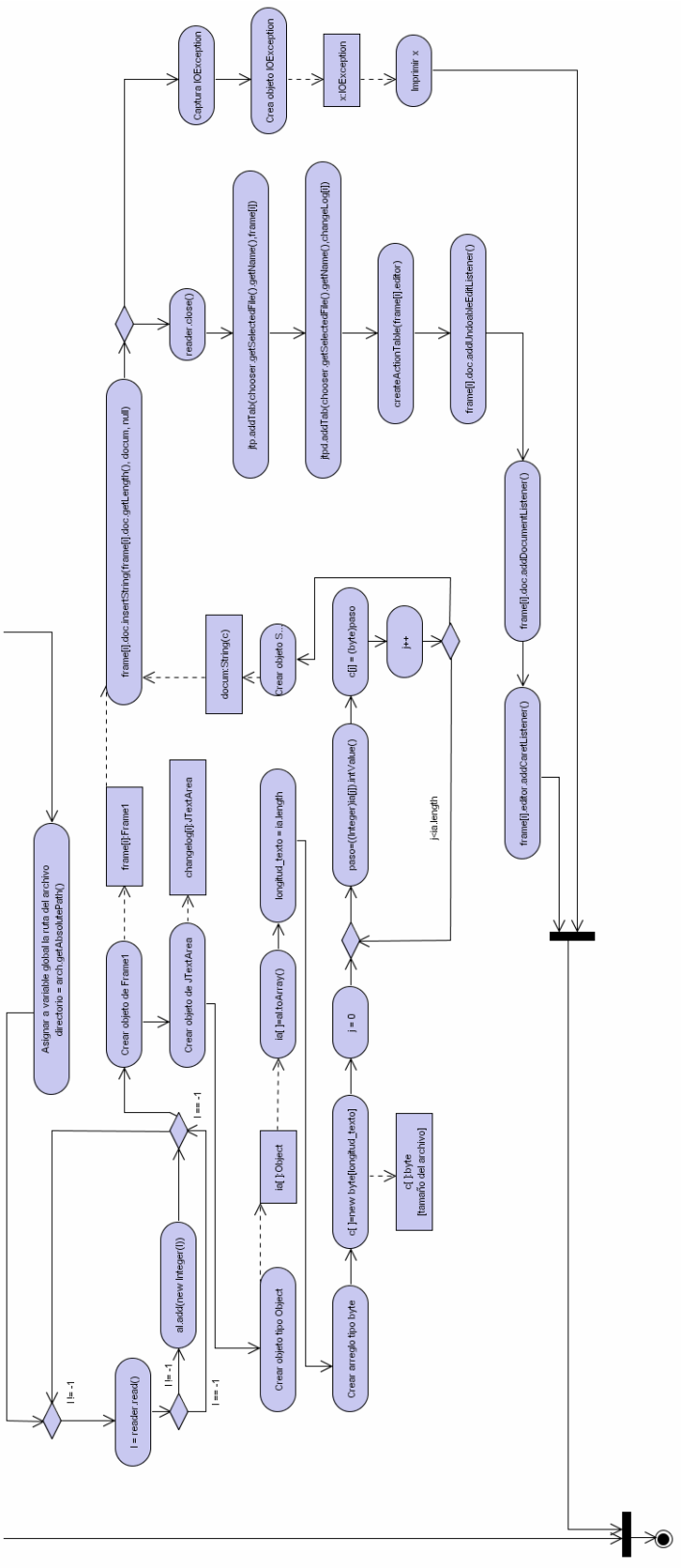


Diagrama VI.19 *Diagrama de actividad del evento OpenHandler(continuación)*

VI.6.5 createNewDoc actividad

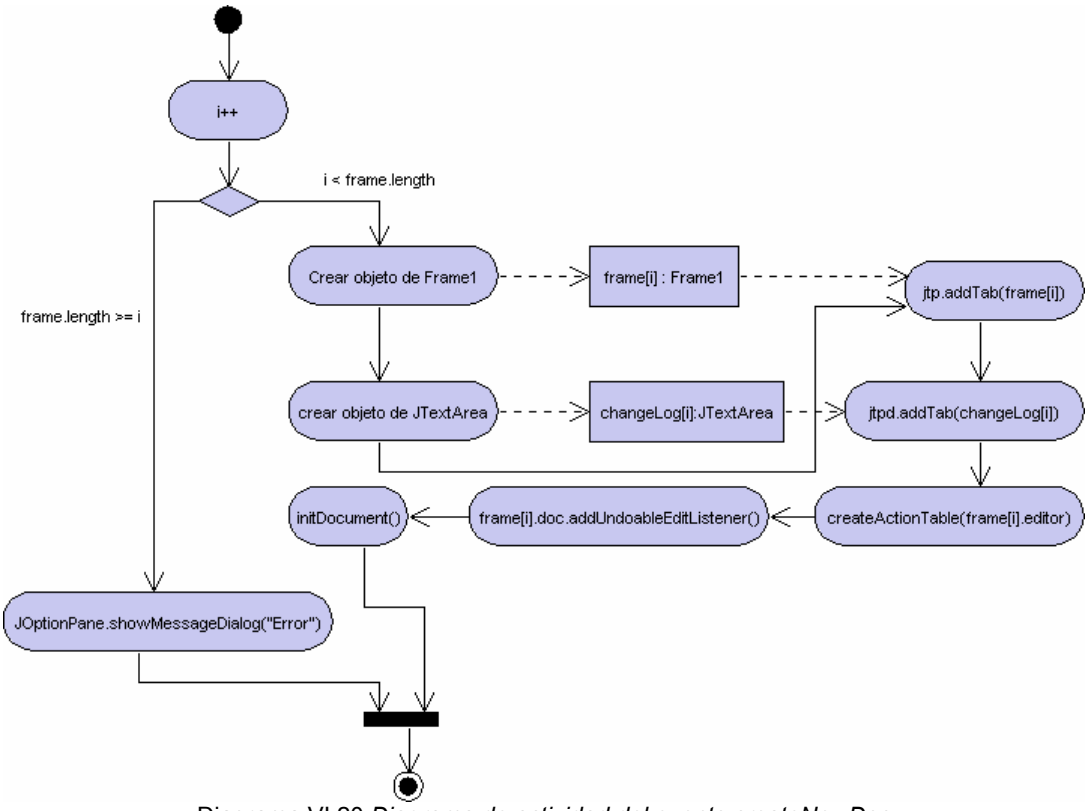


Diagrama VI.20 *Diagrama de actividad del evento createNewDoc*

VI.6.6 createNewDoc colaboración

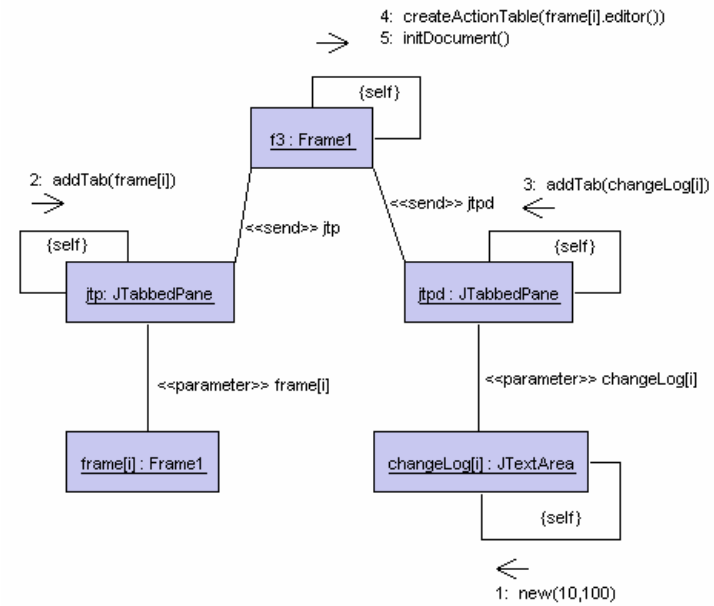


Diagrama VI.21 *Diagrama de colaboración del evento createNewDoc*

VI.6.7 texto actividad

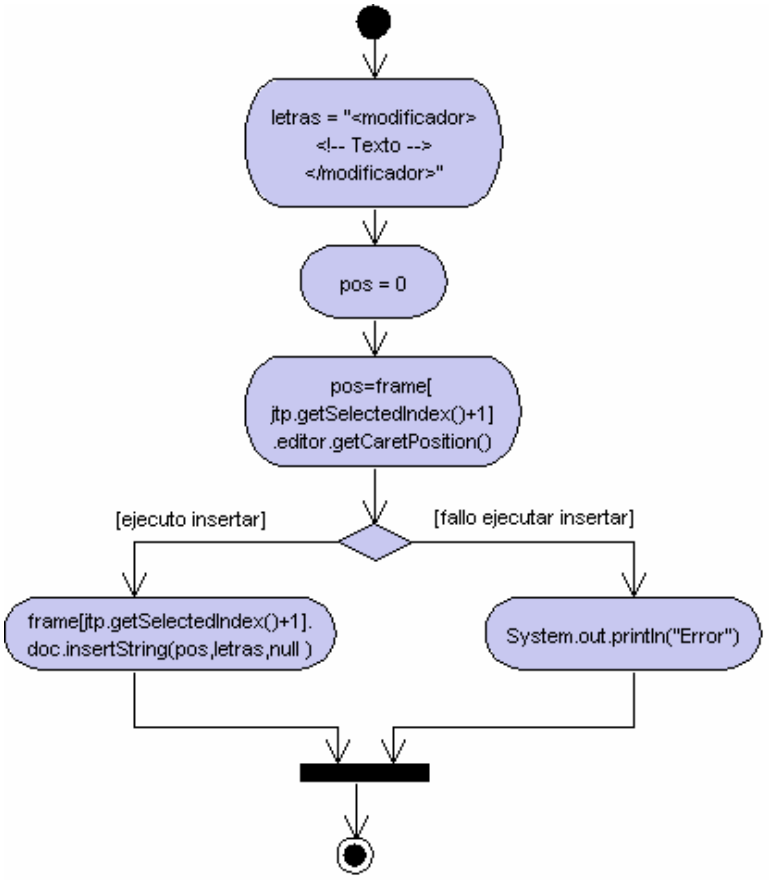


Diagrama VI.22 *Diagrama de actividad de insertarTexto*

VI.6.8 texto colaboración

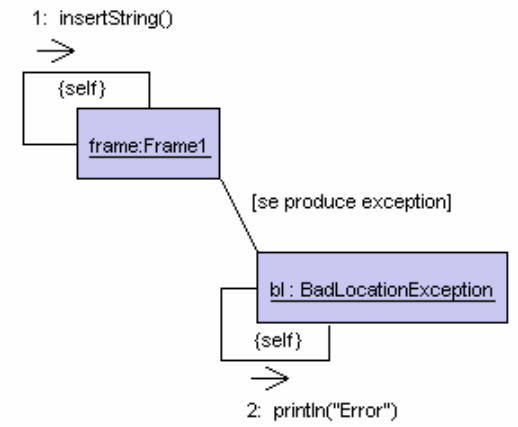


Diagrama VI.23 *Diagrama de colaboración de insertarTexto*

VI.6.9 undoAction actividad

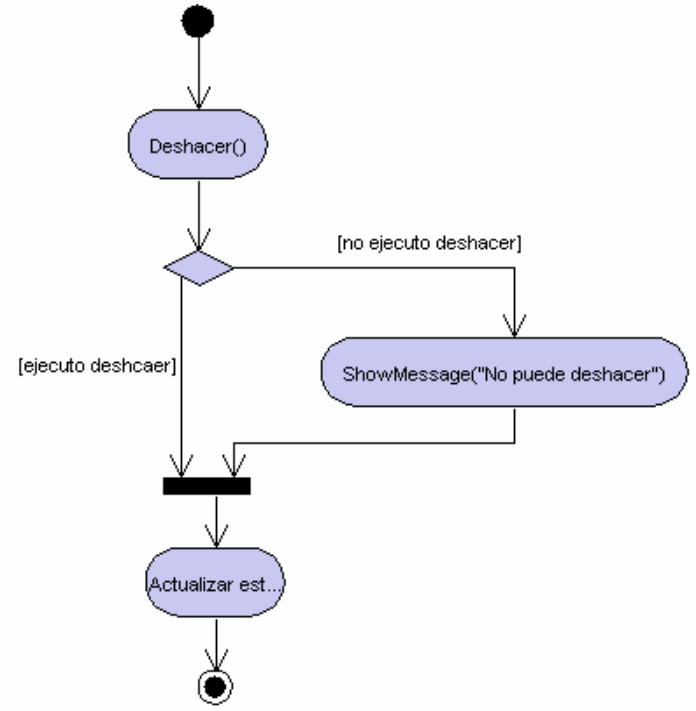


Diagrama VI.24 *Diagrama de actividad del evento undoAction*

VI.6.10 undoAction colaboración

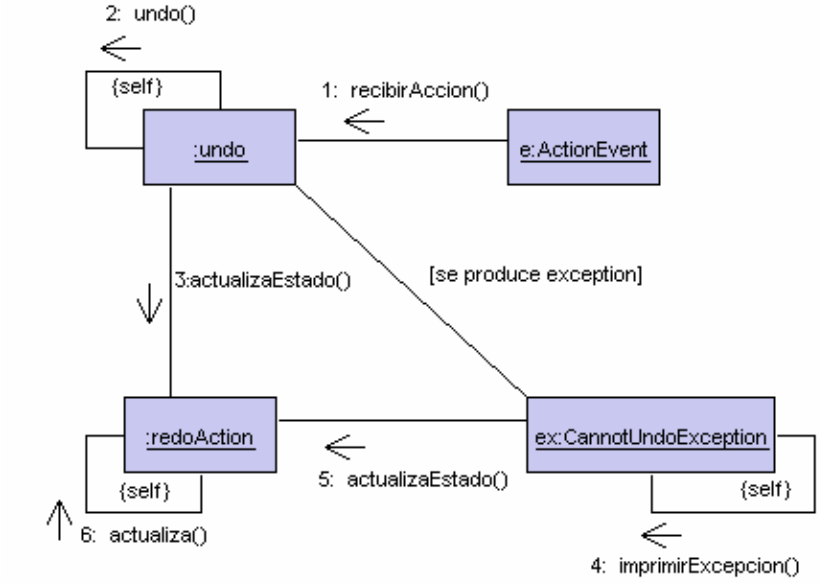


Diagrama VI.25 *Diagrama de colaboración del evento undoAction*

VI.6.11 updateUndoState actividad

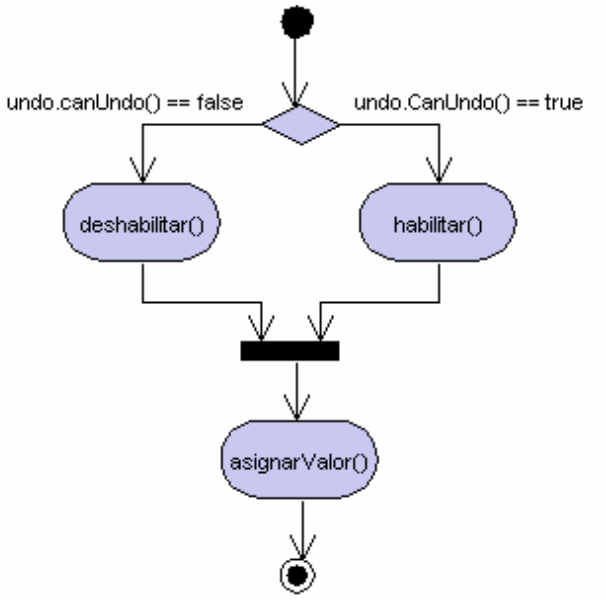


Diagrama VI.26 *Diagrama de actividad del evento updateUndoState*

VI.6.12 updateUndoState colaboración

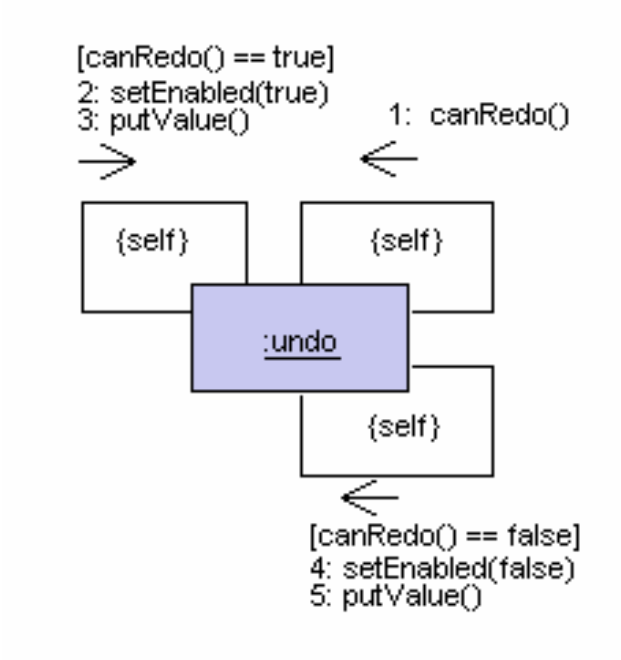


Diagrama VI.27 *Diagrama de colaboración del evento updateUndoState*

VI.6.13 compilar colaboración

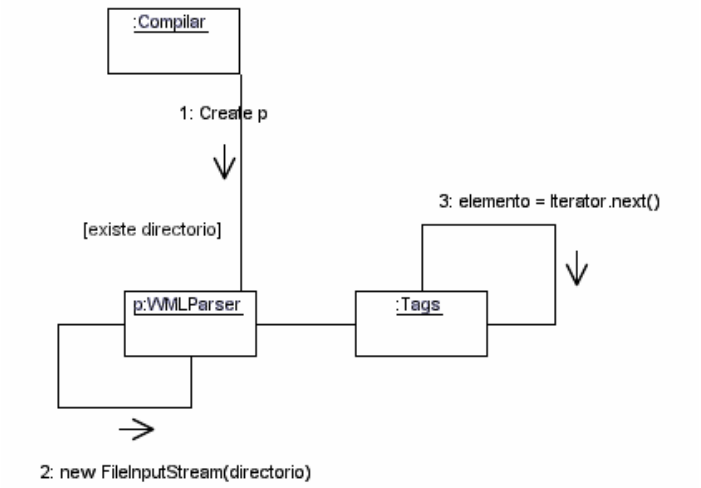


Diagrama VI.28 *Diagrama de colaboración del método compilar*

VI.6.14 wml colaboración

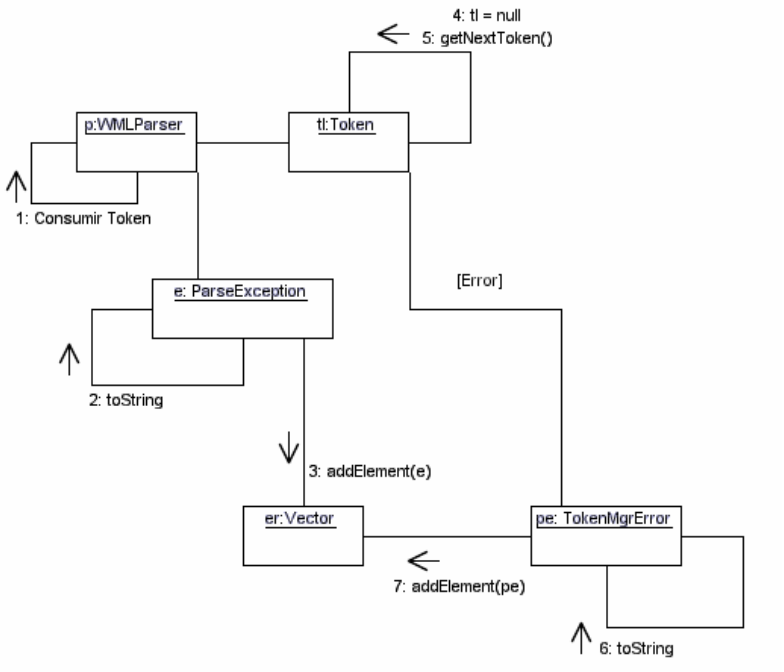


Diagrama VI.29 *Diagrama de colaboración del método wml*

Referencias

[Booch, 1999] Grady Booch, James Rumbaugh, Ivar Jacobson. *El Lenguaje Unificado de Modeloado*. Addison Wesley Iberoamericana, Madrid, 1999

[Fowler, 1999] Martín Fowler. *UML gota a gota*. Addison Wesley Longman de Mexico, S.A. de C.V., Mexico, 1999

[Sinan, 1998] Sinan Si Alhir. *UML in a Nutshell: A Desktop Quick Reference*. O'Reilly & Associates, USA, 1998

[Sinan, 2003] Sinan Si Alhir. *Learning UML*. O'Reilly & Associates, USA, 2003

CAPÍTULO VII
DESARROLLO DEL SISTEMA

VII.1 El entorno de programación JAVA

Existen distintos programas comerciales que permiten desarrollar código *Java*. La Compañía *Sun*, creadora de *Java*, distribuye gratuitamente el *Java Development Kit* (*JDK*). Se trata de un conjunto de programas y librerías que permiten desarrollar, compilar y ejecutar programas en *Java*. Incorpora además la posibilidad de ejecutar parcialmente el programa, deteniendo la ejecución en el punto deseado y estudiando en cada momento el valor de cada una de las variables (con el denominado *Debugger*). Cualquier programador con un mínimo de experiencia sabe que una parte muy importante del tiempo destinado a la elaboración de un programa se destina a la detección y corrección de errores. Existe también una versión reducida del *JDK*, denominada *JRE* (*Java Runtime Environment*) destinada únicamente a ejecutar código *Java* (no permite compilar).

Los *IDE*s (*Integrated Development Environment*), tal y como su nombre indica, son entornos de desarrollo integrados. En un mismo programa es posible escribir el código *Java*, compilarlo y ejecutarlo sin tener que cambiar de aplicación. Algunos incluyen una herramienta para realizar *Debug* gráficamente, frente a la versión que incorpora el *JDK* basada en la utilización de una consola bastante difícil y pesada de utilizar. Estos entornos integrados permiten desarrollar las aplicaciones de forma mucho más rápida, incorporando en muchos casos librerías con componentes ya desarrollados, los cuales se incorporan al proyecto o programa. Entre éstos podemos encontrar Visual J++, NetBeans IDE 3.6 que se distribuye con la versión J2SE v1.4.2_04. Como inconvenientes se pueden señalar algunos fallos de compatibilidad entre plataformas, lentitud en la carga de los IDEs y archivos resultantes de mayor tamaño que los basados en clases estándar.

VII.1.2 El compilador de JAVA

Se trata de una de las herramientas de desarrollo incluidas en el *JDK*. Realiza un análisis de sintaxis del código escrito en los archivos fuente de *Java* (con extensión **.java*). Si no encuentra errores en el código genera los archivos compilados (con extensión **.class*) que contienen la representación intermedia del programa que contiene las instrucciones que el intérprete Java tiene que ejecutar. En otro caso muestra la línea o líneas erróneas. En el *JDK* de *Sun* dicho compilador se llama *javac.exe*; se ejecuta especificando el nombre del archivo fuente en la línea de comando, tal y como se muestra a continuación:

```
C:\> javac Ejemplo.java
```

VII.1.3 La Java Virtual Machine

La existencia de distintos tipos de procesadores y ordenadores llevó a los ingenieros de *Sun* a la conclusión de que era muy importante conseguir un software que no dependiera del tipo de procesador utilizado. Se planteó la necesidad de conseguir un código capaz de ejecutarse en cualquier tipo de máquina. Una vez compilado no debería ser necesaria ninguna modificación por el hecho de cambiar de procesador o de ejecutarlo en otra máquina. La clave consistió en desarrollar un código "neutro" el cual estuviera preparado para ser ejecutado sobre una "máquina hipotética o virtual", denominada *Java Virtual Machine* (*JVM*). Es esta *JVM* quien interpreta este código neutro convirtiéndolo a código particular de la CPU utilizada. Se evita tener que realizar un programa diferente para cada CPU o plataforma.

La *JVM* es el intérprete de *Java*. Ejecuta los *bytecodes* (archivos compilados con extensión **.class*) creados por el compilador de *Java* (*javac.exe*). Tiene numerosas opciones entre las que destaca la posibilidad de utilizar el denominado *JIT* (*Just-In-Time Compiler*), que puede mejorar entre 10 y 20 veces la velocidad de ejecución de un programa.

Cuando se compila código fuente, cada clase individual se almacena en su propio archivo, con el mismo nombre de la clase y utilizando la extensión **.class*. Ésta es la razón por la que nos conviene nombrar los archivos fuente con el mismo nombre que la clase que contienen, ya que así el nombre del archivo fuente coincidirá con el nombre del archivo **.class*. Cuando se ejecute el intérprete Java, se especificará realmente el nombre de la clase que se quiere que el intérprete ejecute. Automáticamente buscará un archivo con ese nombre y con la extensión **.class*. Si encuentra el archivo, ejecutará el código contenido en la clase especificada.

VII.1.4 Las variables PATH y CLASSPATH

El desarrollo y ejecución de aplicaciones en *Java* exige que las herramientas para compilar (*javac.exe*) y ejecutar (*java.exe*) se encuentren accesibles. La computadora, desde una ventana de MS-DOS, sólo es capaz de ejecutar los programas que se encuentran en los directorios indicados en la variable *PATH* del equipo (o en el directorio activo). Si se desea compilar o ejecutar código en *Java*, el directorio donde se encuentran estos programas (*java.exe* y *javac.exe*) deberá encontrarse en el *PATH*.

Java utiliza además una nueva variable de entorno denominada *CLASSPATH*, la cual determina dónde buscar tanto las clases o librerías de *Java* (el *API* de *Java*) como otras clases de usuario. La variable *CLASSPATH* puede incluir la ruta de directorios o archivos **.zip* o **.jar* en los que se encuentren los archivos **.class*. En el caso de los archivos **.zip* hay que observar que los archivos en él incluidos no deben estar comprimidos. En el caso de archivos **.jar* existe una herramienta (*jar.exe*), incorporada en el *JDK*, que permite generar estos archivos a partir de los archivos compilados **.class*. Los archivos **.jar* son archivos comprimidos y por lo tanto ocupan menos espacio que los archivos **.class* por separado o que el archivo **.zip* equivalente.

VII.2 Estructura General de un programa en Java

En la estructura habitual de un programa realizado en cualquier lenguaje orientado a objetos u *OOP* (*Object Oriented Programming*), y en particular en el lenguaje *Java*, aparece una clase que contiene el programa principal (aquel que contiene la función *main()*) y algunas clases de usuario (las específicas de la aplicación que se está desarrollando) que son utilizadas por el programa principal. Los archivos fuente tienen la extensión **.java*, mientras que los archivos compilados tienen la extensión **.class*.

Un archivo fuente (**.java*) puede contener más de una clase, pero sólo una puede ser *public*. El nombre del archivo fuente debe coincidir con el de la clase *public* (con la extensión **.java*). Si por ejemplo en un archivo aparece la declaración (*public class MiClase {...}*) entonces el nombre del archivo deberá ser *MiClase.java*. Es importante que coincidan mayúsculas y minúsculas ya que *MiClase.java* y *miclase.java* serían clases diferentes para *Java*.

De ordinario una aplicación está constituida por varios archivos **.class*. Cada clase realiza unas funciones particulares, permitiendo construir las aplicaciones con gran modularidad e independencia entre clases. La aplicación se ejecuta por medio del nombre de la clase que contiene la función *main()* (sin la extensión **.class*). Las clases de *Java* se agrupan en *packages*, que son librerías de clases. Si las clases no se definen como pertenecientes a un *package*, se utiliza un *package* por defecto (*default*) que es el directorio activo.

VII.2.1 Concepto de Clase

Las clases son el núcleo de Java. La construcción lógica sobre la que se basa el lenguaje Java define la forma y naturaleza de un objeto y forma la base de la programación orientada a objetos en Java. Cualquier concepto que se quiera implementar en Java debe estar encapsulado dentro de una clase.

Probablemente la característica más importante de una clase es que define un nuevo tipo de dato. Una vez definido, este nuevo tipo de dato se puede utilizar para crear objetos de ese mismo tipo o clase. De este modo, una clase es un modelo para un objeto, y un objeto es una instancia de una clase.

VII.2.1.1 Forma general de una clase

Cuando se define una clase, se declara su forma y naturaleza exactas, especificando los datos que contiene y el código que opera sobre los mismos. Una clase se declara mediante la palabra clave *class*. La forma general de definir una clase es la siguiente:

class *nombre de la clase*{
tipo variable de instancia1;
tipo variable de instancia2;
// ...
tipo variable de instanciaN;

tipo nombre del método1(lista de parámetros){

```
//cuerpo del método
}
tipo nombre del método2(lista de parámetros){
//cuerpo del método
}
// ...
tipo nombre del métodoN(lista de parámetros){
//cuerpo del método
}
}
```

Los datos, o variables, definidos en una clase se denominan variables de instancia. El código está contenido dentro de los métodos. El conjunto de los métodos y las variables definidos dentro de una clase se denominan miembros de la clase. En la mayor parte de las clases, los métodos definidos acceden y actúan sobre las variables de instancia, es decir, los métodos determinan cómo se deben utilizar los datos de una clase.

Las variables definidas en una clase se llaman variables de instancia debido a que cada objeto de la clase, contiene su propia copia de estas variables. Además, los datos de un objeto son distintos de los de otro.

VII.2.2 Herencia

La herencia es el proceso por el cual un objeto adquiere las propiedades de otro. La herencia es una de las piedras angulares de la programación orientada a objetos, ya que permite la creación de clasificaciones jerárquicas. Mediante la herencia se puede crear una clase general que define rasgos generales para un conjunto de términos relacionados. Esta clase puede ser heredada por otras clases más específicas, cada una de las cuales añadirá aquellos elementos que la distinguen. En la terminología de Java, una clase que es heredada se denomina superclase. La clase que hereda se denomina subclase.

La herencia permite que se puedan definir nuevas clases basadas en clases existentes, lo cual facilita re-utilizar código previamente desarrollado. Si una clase deriva de otra (*extends*) hereda todas sus variables y métodos. La clase derivada puede añadir nuevas variables y métodos y/o redefinir las variables y métodos heredados.

En *Java*, a diferencia de otros lenguajes orientados a objetos, una clase sólo puede derivar de una única clase, con lo cual no es posible realizar herencia múltiple con base en clases. Sin embargo es posible “simular” la herencia múltiple con base en las interfaces.

VII.2.3 Concepto de Interfaz

Una interfaz es un conjunto de declaraciones de funciones. Si una clase implementa (*implements*) una interfaz, debe definir todas las funciones especificadas por la interfaz. Una clase puede implementar más de una interfaz, representando una forma alternativa de la herencia múltiple.

A su vez, una interfaz puede derivar de otra o incluso de varias interfaces, en cuyo caso incorpora todos los métodos de las interfaces de las que deriva.

Mediante la palabra clave interfaz, se puede abstraer completamente la interfaz de una clase de su implementación, es decir, usando interfaz se puede especificar lo que una clase debe hacer, pero no cómo debe hacerlo. Las interfaces son sintácticamente semejantes a las clases, pero carecen de las variables de instancia, y sus métodos se declaran sin ningún cuerpo. En la práctica, esto implica que se pueden definir interfaces que no hagan suposiciones sobre cómo se implementan. Una vez definida, cualquier número de clases pueden implementar una interfaz.

Para implementar una interfaz, una clase debe crear el conjunto completo de métodos definidos por la interfaz. Sin embargo, cada clase tiene la libertad de determinar los detalles de su implementación. Mediante la palabra clave interfaz, Java permite utilizar el aspecto del polimorfismo, “una interfaz, múltiples métodos”.

VII.2.3.1 Definición de una Interfaz

Una interfaz se define como una clase. La forma general de definir una interfaz es la siguiente:

acceso interfaz *nombre*{
tipo_devuelto método1(*lista_de_parámetros*);
tipo_devuelto método2(*lista_de_parámetros*);
tipo var_final1 = valor;
tipo var_final2 = valor;
// ...
tipo_devuelto métodoN(*lista_de_parámetros*);
tipo var_finalN = valor;
}

donde acceso es *public* o no se usa. Cuando no se incluye el especificador de acceso se utiliza el acceso por defecto, y la interfaz está disponible sólo para otros miembros del paquete en que se declara.

VII.2.4 Concepto de Paquete (Package)

Un *package* es una agrupación de clases. Existen una serie de *packages* incluidos en el lenguaje (ver jerarquía de clases que aparece en el *API* de *Java*).

Además el usuario puede crear sus propios *packages*. Lo habitual es juntar en *packages* las clases que estén relacionadas. Todas las clases que formen parte de un *package* deben estar en el mismo directorio.

VII.2.5 La jerarquía de clases de Java (API)

Durante la generación de código en *Java*, es recomendable y casi necesario tener siempre a la vista la documentación *on-line* del *API* de *Java 1.1* ó *Java 1.2*. En dicha documentación es posible ver tanto la jerarquía de clases, es decir la relación de herencia entre clases, como la información de los distintos *packages* que componen las librerías base de *Java*.

Es importante distinguir entre lo que significa herencia y *package*. Un *package* es una agrupación arbitraria de clases, una forma de organizar las clases. La herencia sin embargo consiste en crear nuevas clases con base en otras ya existentes. Las clases incluidas en un *package no derivan* por lo general de una única clase.

En la documentación *on-line* se presentan ambas visiones: "*Package Index*" y "*Class Hierarchy*", tanto en *Java 1.1* como en *Java 1.2*, con pequeñas variantes. La primera presenta la estructura del *API* de *Java* agrupada por *packages*, mientras que en la segunda aparece la jerarquía de clases. Hay que resaltar una vez más el hecho de que todas las clases en *Java* son derivadas de la clase *java.lang.Object*, por lo que heredan todos los métodos y variables de ésta.

Si se selecciona una clase en particular, la documentación muestra una descripción detallada de todos los métodos y variables de la clase. A su vez muestra su herencia completa (partiendo de la clase *java.lang.Object*).

VII.3 Implementación de funciones

Una vez que hemos realizado una revisión sobre el lenguaje Java, estamos en la posibilidad de conocer algunos de los métodos que utilizamos en el SEEDWML, ante cualquier duda recomendamos leer el libro "Java 2 Manual de Referencia de Herbert Schildt Edit. McGraw-Hill"

VII.3.1 clase OpenHandler

```
protected class OpenHandler implements ActionListener
{
	public void actionPerformed(ActionEvent e)
	{
		JFileChooser chooser = new JFileChooser();
		FileInputStream reader =null;
		FileView fileview = new ImageFileView();
		chooser.addChoosableFileFilter(new WMLFilter());
		chooser.setFileView(fileview);
		int returnVal = chooser.showOpenDialog(Main.this);
		ArrayList al = new ArrayList();
		int l;
		int longitud_texto;

		if(returnVal == JFileChooser.APPROVE_OPTION)
		{
			try
			{
				i++;
				if (i<frame.length)
				{
					File arch=chooser.getSelectedFile();
					reader=new FileInputStream(arch);
					directorio[i]=arch.getAbsolutePath();
					do
					{
						l=reader.read();
						if(l!=-1)
						al.add(new Integer(l));
					}while(l!=-1);
					frame[i]=new Frame1();
					changeLog[i]=new JTextArea(10,100);
					Object ia[]= al.toArray();
					longitud_texto=ia.length;
					byte c[]=new byte[longitud_texto];
					int paso;
					for (int j=0;j<ia.length;j++)
					{
						paso=((Integer)ia[j]).intValue();
						c[j]=(byte)paso;
```

```
                                        }

                                        String docum = new String(c);
                                        try
                                        {
                                    frame[i].doc.insertString(frame[i].doc.getLength(),docum,null);
                                        }
                                catch(BadLocationException ble)
                                {}

                                }
                        }
                        catch(IOException ioe)
                        {}
                        finally
                        {
                                try
                                {
                                        reader.close();
                                        System.out.println("Se cerro el reader");
                                        jtp.addTab(chooser.getSelectedFile().getName(),frame[i]);
                                jtpd.addTab(chooser.getSelectedFile().getName(),changeLog[i]);
                                        createActionTable(frame[i].editor);
                                frame[i].doc.addUndoableEditListener(new
                                MyUndoableEditListener());
                        frame[i].doc.addDocumentListener(new MyDocumentListener());
                                        frame[i].editor.addCaretListener(caretListenerLabel);
                                }
                                catch(IOException x){
                                        System.out.println("El error es: "+x);
                                        }
                        }
                }
        }
}
```

VII.3.2 clase CreateNewDoc

```
protected class CreateNewDoc implements ActionListener
{
        public void actionPerformed(ActionEvent e)
        {
                i++;
                if (i<frame.length)
                {
                  frame[i]=new Frame1();
                  changeLog[i] = new JTextArea(10,100);
                  jtp.addTab("Documento "+i,frame[i]);
                  jtpd.addTab("Documento "+i,changeLog[i]);
                  createActionTable(frame[i].editor);
                  frame[i].doc.addUndoableEditListener(new MyUndoableEditListener());
                  frame[i].doc.addDocumentListener(new MyDocumentListener());
                  frame[i].editor.addCaretListener(caretListenerLabel);
                  initDocument();
                }
                else
                {
                    JOptionPane.showMessageDialog(null, "No se pueden crear mas documentos",
"..: Cuidado!! :..", JOptionPane.ERROR_MESSAGE);

                }
        }
}
```

VII.3.3 clase salirHandler

```
protected class salirHandler implements ActionListener
{
        public void actionPerformed(ActionEvent e)
        {
                for(int j=0;j<modifica.length;j++)
                {
                 if (modifica[j]==true)
                 {
                    int resp = optionPane.showInternalConfirmDialog(
                    desk, "Se modifico el documento "+jtp.getTitleAt(j-1)+", desea guardarlo?",
                                "Guardar documento de SEEDWML",
                    JOptionPane.YES_NO_OPTION);
                    if (resp == JOptionPane.YES_OPTION)
                    {
                        JFileChooser chooser = new JFileChooser();
                        DataOutputStream outStream;
                        FileView fileView = new LowerCaseFileView();
                         chooser.setFileView(fileView);
                         int returnVal = chooser.showSaveDialog(Main.this);
                         if(returnVal == JFileChooser.APPROVE_OPTION)
                         {
                            try
                            {
                               outStream = new DataOutputStream(new
                               FileOutputStream(chooser.getSelectedFile()));
                              outStream.writeBytes(frame[j].editor.getText());
                              outStream.close();
                              jtp.setTitleAt(j-1,chooser.getSelectedFile().getName());
                              modifica[j]=false;
                            }
                            catch(IOException ioe)
                            {
                            }
                         }
                         if(j==(modifica.length))
                        {
                           System.out.println("Salida");
                           System.exit(0);
                         }
                }
                else
                {
                          if(j==(modifica.length))
                          {
                                    System.out.println("Salida 2");
                                    System.exit(0);
                          }
```

```
                }
        }
}
        int n = optionPane.showInternalConfirmDialog(desk, "Desea salir de SEEDWML?","Salir
de SEEDWML",JOptionPane.YES_NO_OPTION);
if (n == JOptionPane.YES_OPTION) {
        System.exit(0);
}
}
}
```

VII.3.4 clase MyUndoableEditListener

```
protected class MyUndoableEditListener implements UndoableEditListener
{
    public void undoableEditHappened(UndoableEditEvent e)
    {
        undo.addEdit(e.getEdit());
        undoAction.updateUndoState();
        redoAction.updateRedoState();
    }
}
```

VII.3.5 clase Elemento

```
protected class elemento implements ActionListener
{
        String cadena = "";
        SimpleAttributeSet letraBold = new SimpleAttributeSet();

        public elemento(String selem){
                cadena = selem;
        }
        public void actionPerformed(ActionEvent e)
        {
                int pos=0;
                pos=frame[jtp.getSelectedIndex()+1].editor.getCaretPosition();
                try
                {
                        StyleConstants.setBold(letraBold, true);
                        frame[jtp.getSelectedIndex()+1].doc.insertString(pos,cadena,letraBold);
                }
                catch( BadLocationException bl ) {
                System.err.println( bl );
                System.out.println("Error en: "+ (jtp.getSelectedIndex()+1));
        }
}
```

VII.4 El SEEDWML (Sistema de Edición y Emulación de Documentos WML)

Después de haber realizado las cuatro primeras fases del flujo de trabajo de vida del software, de acuerdo al proceso de desarrollo, lo que obtenemos es un producto funcional, listo para pasar por las pruebas necesarias (capítulo VIII).

VII.4.1 Apariencia del SEEDWML

El sistema ya terminado tiene las siguientes características:

- Cuenta con una área para la edición de documentos WML.
- Se tiene acceso a la sintaxis de los elementos por medio de botones y los menús.
- Se tienen botones de acceso rápido a las funciones más comunes con archivos.
- Se cuenta con una área de despliegue de errores.

Como se ve en la figura. VII.1

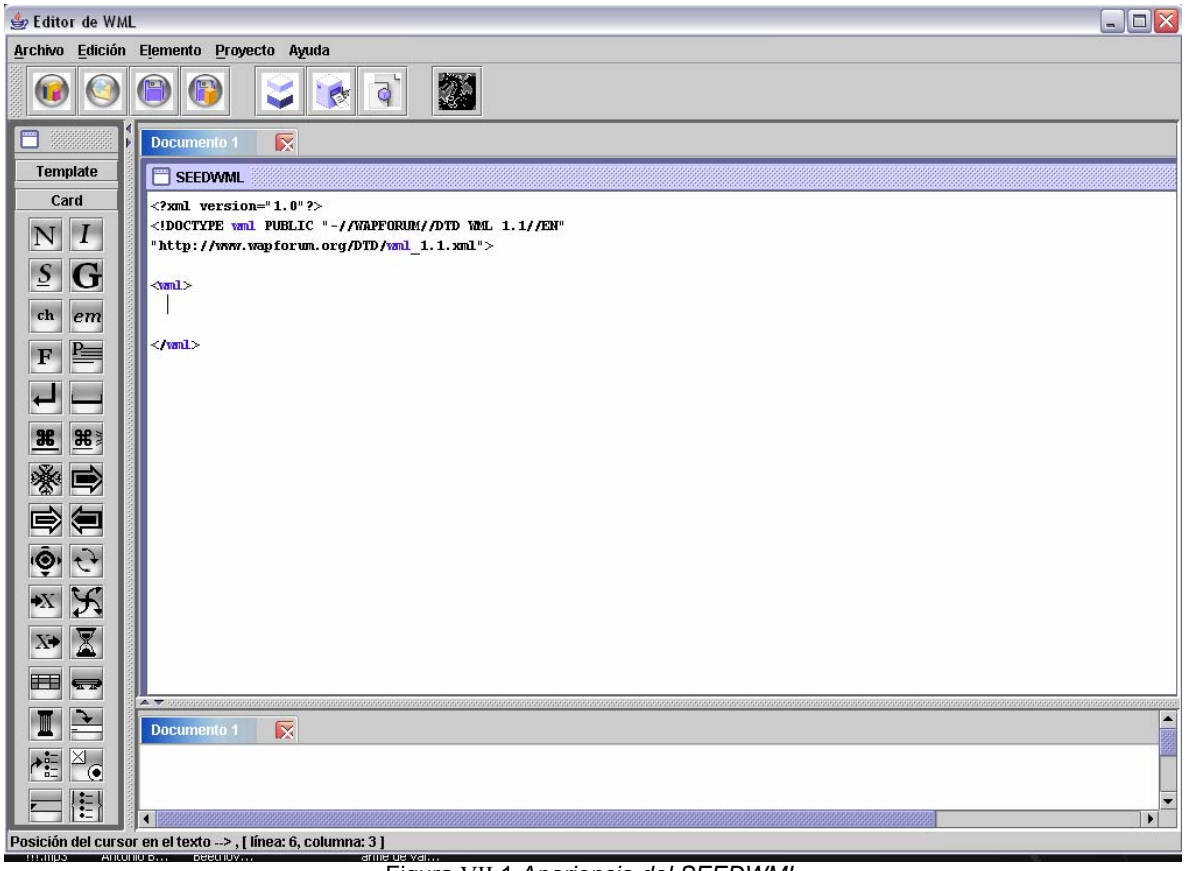


Figura VII.1 *Apariencia del SEEDWML*

VII.4.2 Funcionabilidad del SEEDWML

Primero para conocer la funcionabilidad del SEEDWML, debemos conocer las partes que lo integran.
Quizá la parte más importante es el área de escritura, que es el área más grande y se muestra en la figura VII.2

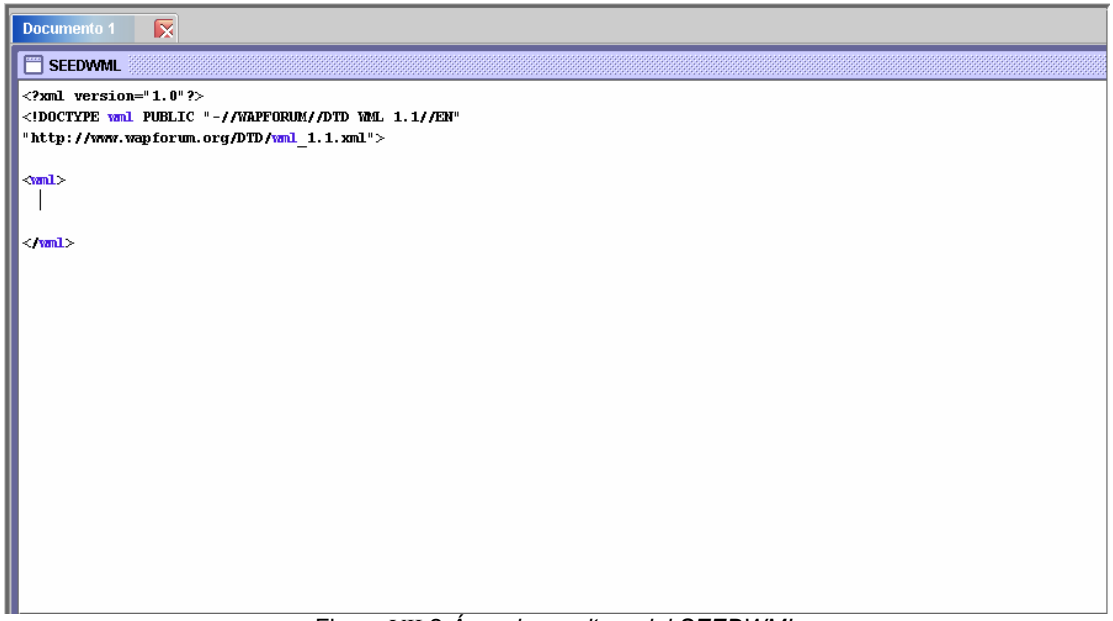


Figura VII.2 *Área de escritura del SEEDWML*

Podemos escribir directamente sobre esta área o, para agilizar la inserción del texto de los elementos WML, podemos hacer uso de los botones de los elementos ubicados en la parte izquierda del área de escritura, como se muestra en la figura VII. 3

Figura VII.3 *Botones de elementos del SEEDWML*

También podemos agregar la sintaxis de los elementos por medio de los menús, como se muestra en la figura VII.4

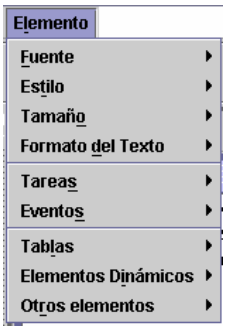


Figura VII.4 *Menús de elementos del SEEDWML*

Cuenta el sistema también con funciones básicas para cualquier editor, como son: *abrir*, *guardar*, *guardar como* y *nuevo*, así como acciones básicas de edición de texto, *copiar*, *cortar* y *pegar*. Y de igual manera se puede llevar a cabo estas acciones ya sea por medio de los menús o los botones que se encuentran inmediatamente arriba del área de escritura y los botones, como se ve en la figura VII. 5

Figura VII.5 *Botones de acceso rápido a las funciones básicas del SEEDWML*

Para evitar confusión, cada botón despliega información respecto a su función al colocar el apuntador del ratón por encima de éstos.

Contamos con un botón mas, como se ve en la figura VII.6

Figura VII.6 *Botón de acceso rápido al emulador del SEEDWML*

Este botón al igual que el menú de acceso que se aprecia en la figura VII.7 nos dan acceso al emulador.

Figura VII.7 *Menú de acceso al emulador del SEEDWML*

La única condición para poder ejecutar el emulador es que el archivo seleccionado esté guardado y no haya sufrido ninguna modificación entre el momento del guardado y el presionado del botón de emular.
De haber algún error, éste se reflejaría en el área de despliegue de errores, definida para este efecto, y que se puede apreciar en la figura VII.8

Figura VII.8. *Área de despliegue de errores del SEEDWML*

Para poder ubicar rápidamente la columna y el renglón en el cual se localizó el error, el SEEDWML cuenta en la barra de estado con un indicador de renglón y columna. Como se ve en la figura VII.9

Figura VII.9 *Barra de estado del SEEDWML*

Hay algunas condiciones que deben conocerse respecto al SEEDWML respecto a su funcionamiento. La primera es que siempre habrá un documento abierto, esto es, no se podrá cerrar el último documento. El número máximo de documentos que puede un usuario tener abiertos es 14. Al cambiarse de documentos en el área de escritura también se produce el cambio en el área de despliegue de errores y viceversa. Lo mismo sucede cuando se cierra un documento. En caso de querer generar o abrir mas documentos que los permitidos, aparece un cuadro de diálogo como el de la figura VII.10.

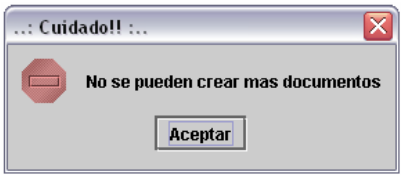


Figura VII.10 *Cuadro de dialogo, aviso de no más documentos*

Y en caso de intentar emular y no estar guardado el documento, entonces el cuadro que aparece es como el de la figura VII.11

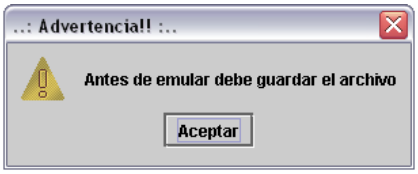


Figura VII.11 *Cuadro de diálogo, advertencia de archivo no guardado*

E inmediatamente después de aparecido este cuadro de diálogo, parece una ventana de salvar archivo, como en la figura VII.12

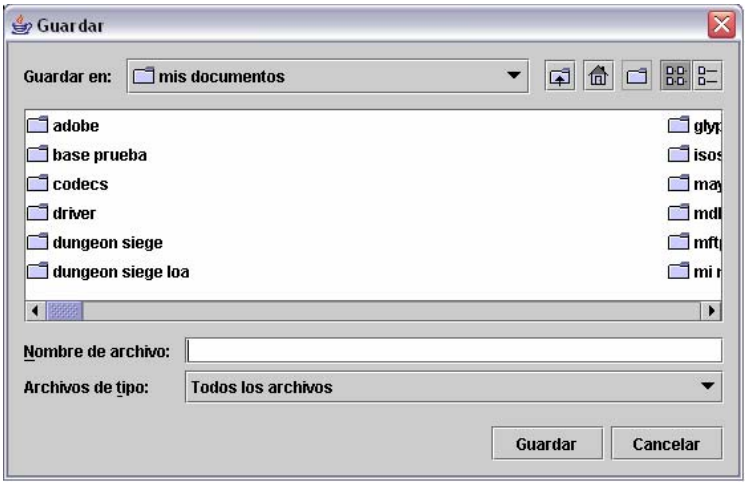


Figura VII.12 *Cuadro de diálogo, salvar archivo*

En caso de no haber ningún error en el documento WML y estar guardado, entonces se llama al emulador que tiene una apariencia como se muestra en la figura VII.13

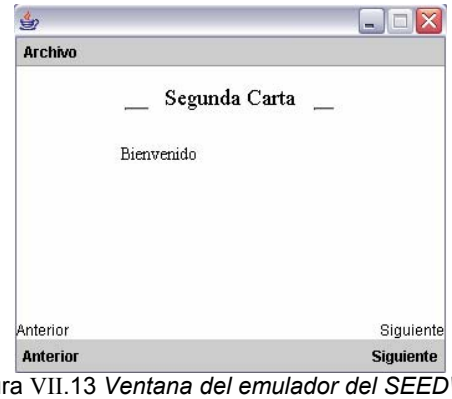


Figura VII.13 *Ventana del emulador del SEEDWML*

Y así se definen las funciones básicas del emulador, es demasiado intuitivo y en caso de requerir apoyo técnico sobre el lenguaje de marcado inalámbrico (WML), se incluye una ayuda en formato HTML, a la cual se accede por medio del menú *Ayuda,* y el cual incluye una serie de ejemplos y descripciones para hacer más fácil el desarrollo de documentos WML.

Referencias

[García de Jalón *et al*, 2000] Javier García de Jalón, José Ignacio Rodríguez, Iñigo Mingo, Aitor Imaz, Alfonso Brazález, Alberto Larzabal, Jesús Calleja, Jon García. *Aprenda Java como si estuviera en primero*. Escuela Superior de Ingenieros Industriales de San Sebastián, Universidad de Navarra, 2000.

[Loy et al, 2002] Marc Loy, Robert Eckstein, Dave Wood, James Elliott. *Java Swing*. 2a edición, O'Reilly & Associates, USA, 2002

[Schildt, 2001] Schildt, Herbert. *Java 2 Manual de Referencia*. McGraw-Hill, España, 2001.

CAPÍTULO VIII
Pruebas y evaluación

VIII.1 Introducción

En este flujo de trabajo verificamos el resultado de la implementación realizada (véase capítulo VII). Es importante el flujo de pruebas porque gracias a éste podemos diseñar e implementar los casos de pruebas que nos indiquen qué probar y bajo qué procedimiento hacerlo. Y en caso de haberlos, cómo utilizar los componentes de prueba. [Jacobson et al, 2000]

VIII.2 Modelo de pruebas

El modelo de pruebas describe cómo se prueban los componentes. El modelo de pruebas cambia constantemente a lo largo del ciclo de desarrollo iterativo debido a que:

- Se van eliminando los casos de prueba obsoletos
- Se vuelven más específicos los casos de prueba

El esquema general del modelo de pruebas se muestra en la figura VIII.1

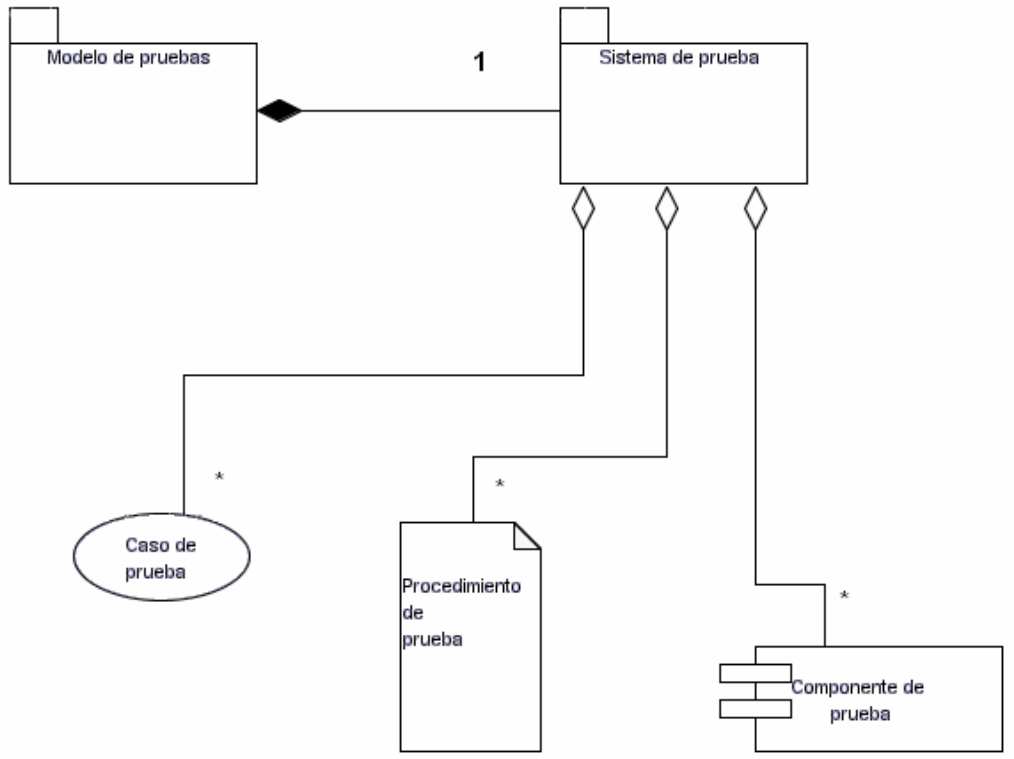


Figura VIII.1 *Modelo de pruebas*

VIII.2.1 Caso de prueba

Un caso de prueba especifica la forma de probar el sistema, incluyendo la entrada o resultado que se ha de probar y las condiciones bajo las cuales ha de probarse. Un caso de prueba básicamente se deriva de un caso de uso, así que un caso de prueba especifica cómo probar un caso de uso o un escenario específico de un caso de uso.[Jacobson et al, 2000]

Nuestros casos de prueba son el evento abrir y el de emular (en el caso de los errores provocados – *ver VIII.2.4*- el evento guardar).

VIII.2.2 Procedimiento de prueba

Un procedimiento de prueba especifica cómo realizar uno a varios casos de prueba o partes de éstos.
Nuestro procedimiento de prueba será, abrir un archivo .wml y ejecutarlo sobre los programas elegidos, con el fin de evaluar el manejo de formatos de texto, la navegabilidad y el manejo de variables.

VIII.2.3 Componente de prueba

Un componente de pruebas automatiza uno o varios procedimientos de prueba (o partes de ellos).
Los componentes de prueba pueden ser desarrollados por medio de un lenguaje de programación o por medio de un lenguaje de guiones (scripts). [Jacobson et al, 2000].

Para el flujo de trabajo de prueba del SEEDWML y acorde al procedimiento de prueba, utilizamos tres archivos de nombre: pruebatabla.wml, pfinal.wml (que se liga a foo.wml) y pruebavar.wml (véase Apéndice B).

VIII.2.4 Plan de prueba

Describe las estrategias, recursos y planificación de la prueba.

Para el SEEDWML el plan de prueba es el siguiente:

- Se eligieron dos productos comerciales Wapaka ® de la compañía Digital Airways™ y WinWap 3.1 Pro © de Slob Trot Software Oy Ab (versión de evaluación) para realizar una comparación de productos contra el SEEDWML.
- Las pruebas pensadas y diseñadas para el SEEDWML se aplican a los programas citados en el punto anterior.
- Se evalúa el desempeño del programa paso a paso por medio de *screenshots* (imágenes).
- En el *script* se provocan errores de sintaxis deliberados de los cuales se espera una reacción y se comenta lo observado.
- Los recursos necesarios en hardware son: procesador PIII a 1.2 GHz con 256 MB de RAM y 40 GB de disco duro con 1 MB libre; en software es necesario tener

instalado el JSDK 1.4b04 de Java© Sun ™, así como Wapaka ® y WinWap 3.1 Pro © (para la evaluación).

En este apartado, se mostrarán los resultados de aplicar el plan de prueba al SEEDWML así como a los productos comerciales antes mencionados (a manera de comparación). Antes de mostrar las imágenes de los programas y su desempeño se explicará brevemente el funcionamiento que deseablemente debe tener. Al terminar de evaluar los archivos sobre los tres programas, se explicará el error provocado y la respuesta esperada por parte de los programas.

VIII.2.4.1 Primera prueba

En la primera prueba (archivo pruebatabla.wml) se evaluará la capacidad para representar los diferentes tipos de formateo de texto así como la representación de tablas. Deberá ser clara la diferencia entre un formato y otro, y el borde de la tabla debe ser visible sin ser grueso, siempre ajustado al ancho de la pantalla del emulador.

Sobre WinWap © . Primeramente debemos abrir el archivo, como se ve en la figura VIII.2:

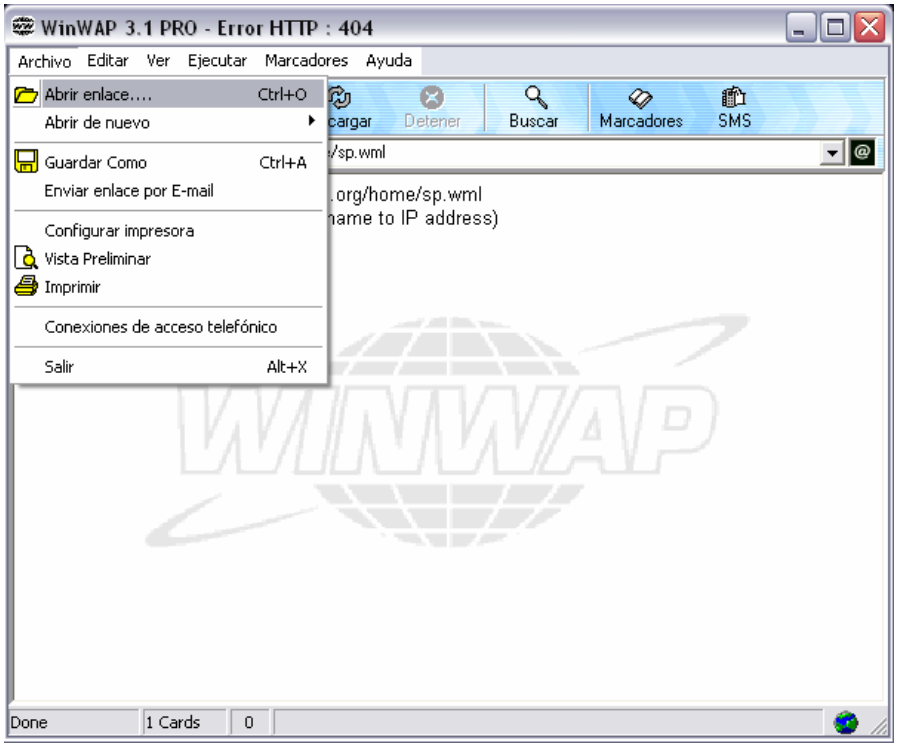


Figura VIII.2 *Abrir archivo en WinWap ©*

Después seleccionamos el archivo (figura VIII.3):

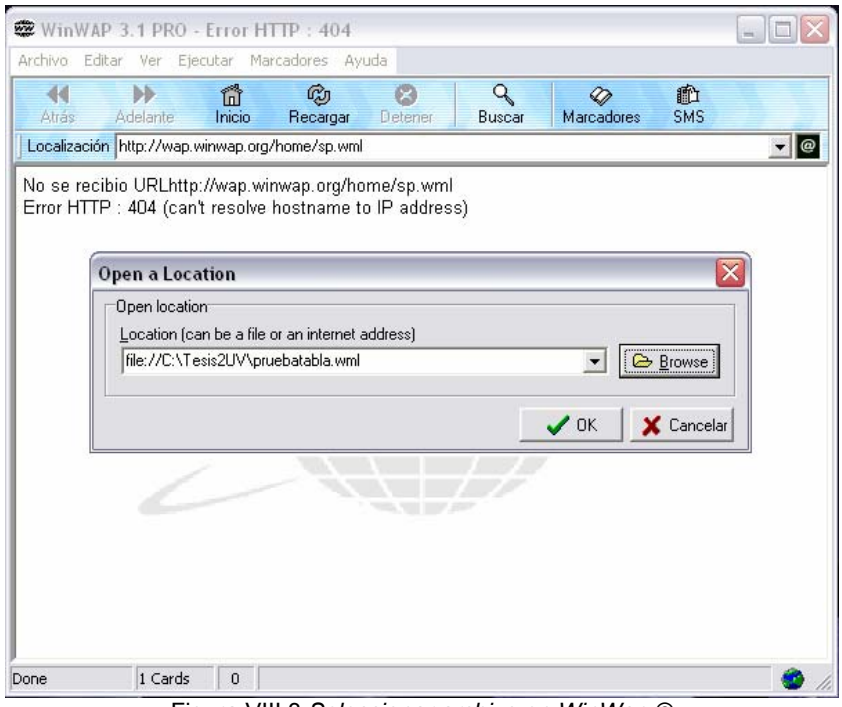


Figura VIII.3 *Seleccionar archivo en WinWap* ©

La forma de despliegue es la que se muestra en la figura VIII.4

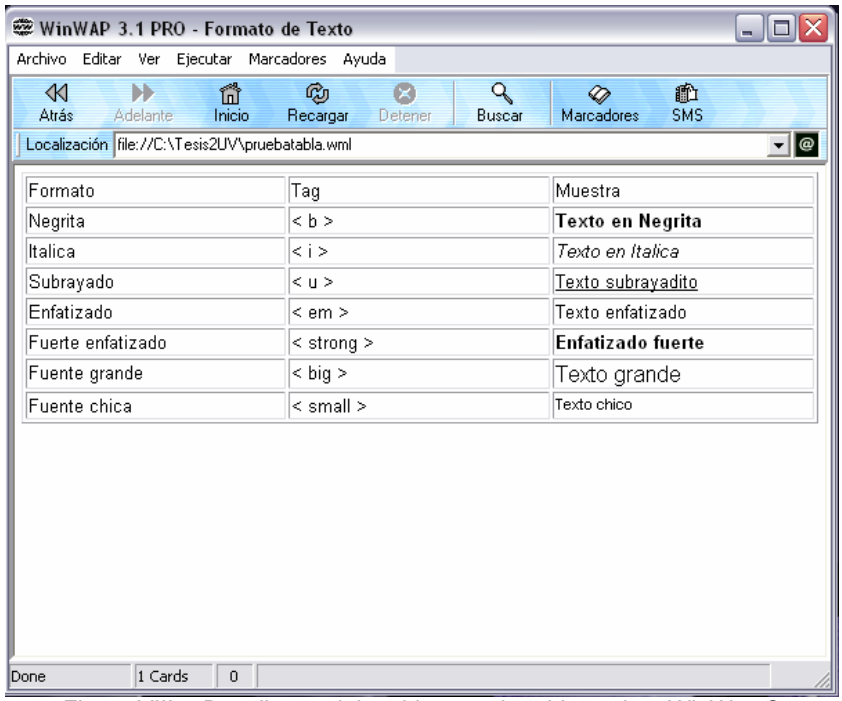


Figura VIII.4 *Despliegue del archivo pruebatabla.wml en WinWap* ©

Sobre Wapaka © . Primeramente seleccionamos el archivo, como en la figura VIII.5:

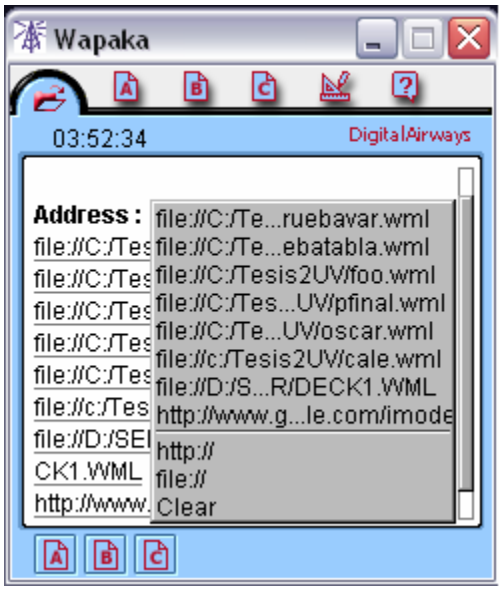


Figura VIII.5 *Abrir archivo en Wapaka ©*

La forma de despliegue es la que se muestra en las figuras VIII.6 y VIII.7

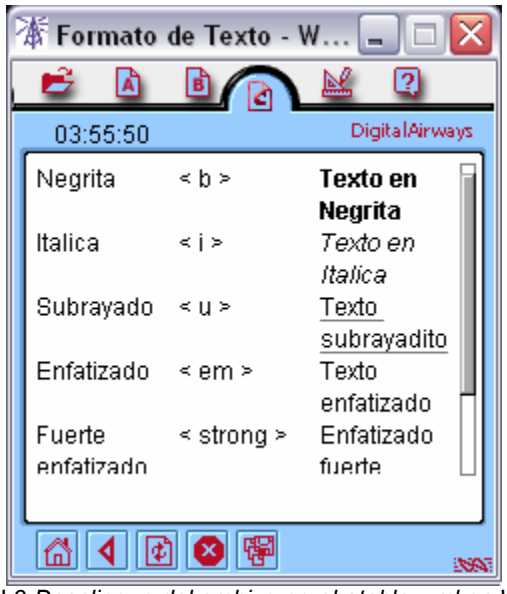


Figura VIII.6 *Despliegue del archivo pruebatabla.wml en Wapaka ©*

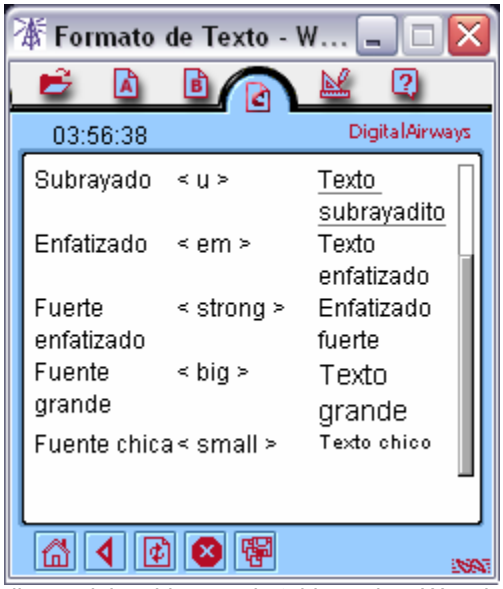


Figura VIII.7 *Despliegue del archivo pruebatabla.wml en Wapaka © (continuación)*

Sobre el SEEDWML. Primeramente abrimos el archivo, como se ve en la figura VIII.8:

Figura VIII.8 *Abrir archivo en el SEEDWML*

Después seleccionamos el archivo (figura VIII.9):

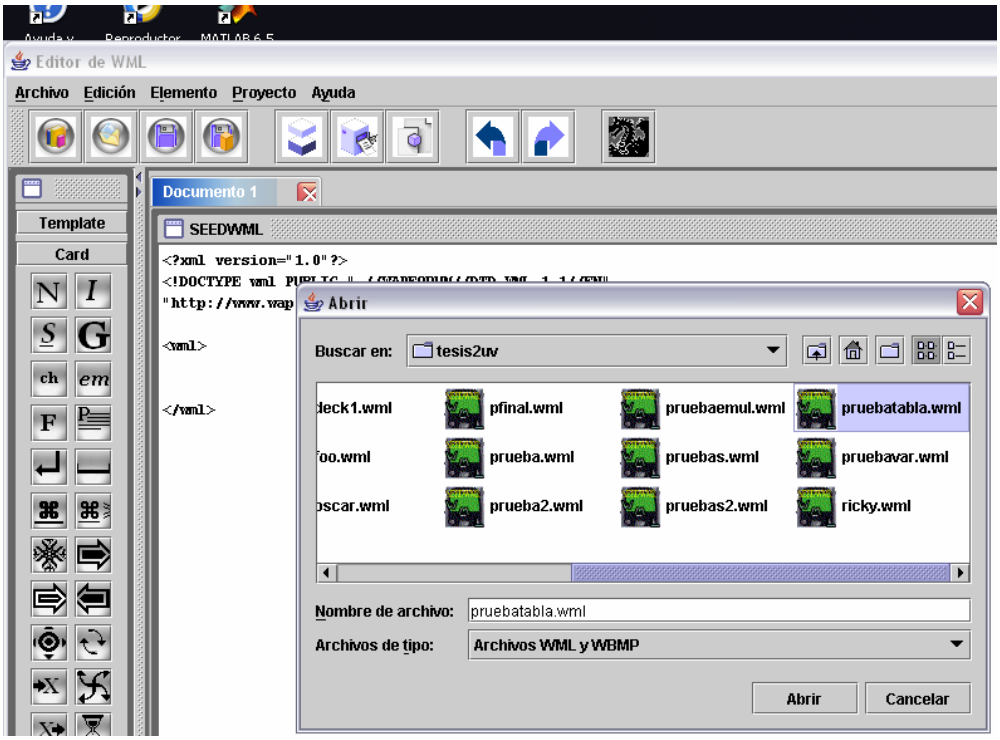


Figura VIII.9 *Seleccionar archivo en el SEEDWML*

La forma de despliegue es la que se muestra en la figura VIII.10

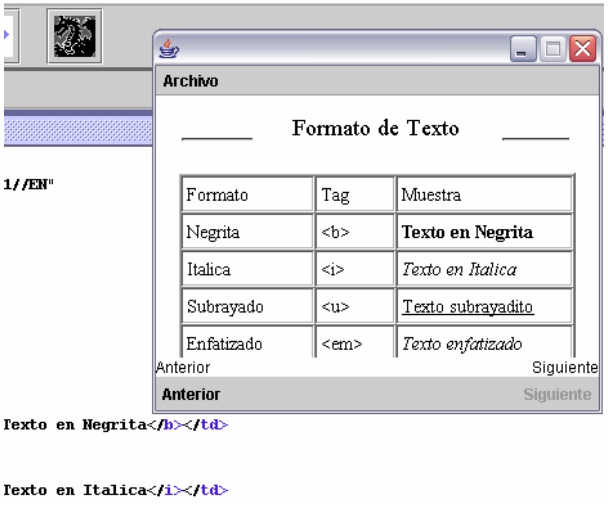


Figura VIII.10 *Despliegue del archivo pruebatabla.wml en el SEEDWML*

En la segunda prueba (archivo pfinal.wml) se evalúa la capacidad de navegabilidad del browser del emulador.
Las primeras tres cards (que forman una baraja) pertencen a un mismo archivo (pfinal.wml), a partir de la liga que se hace a "java" la acción se lleva a cabo sobre el archivo foo.wml. Lo interesante viene en la liga de la tercera carta hacia la cuarta, o sea, la liga de un archivo a otro.

Sobre WinWap © . Después de seleccionar y abrir el archivo (como se vio anteriormente), se despliega la primera carta, que se ve en la figura VIII.11:

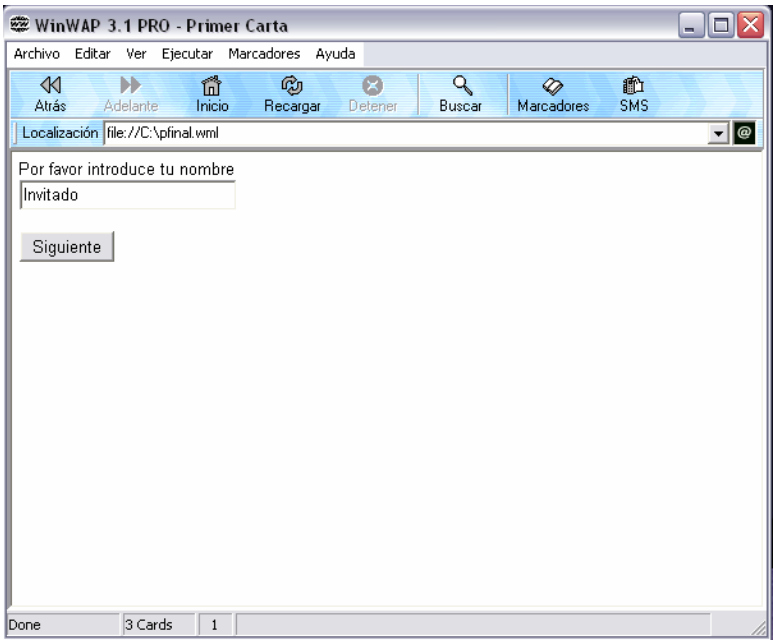


Figura VIII.11 *Primera carta del archivo pfinal.wml en Winwap ©*

Al presionar siguiente, el emulador se muestra como en la figura VIII.12:

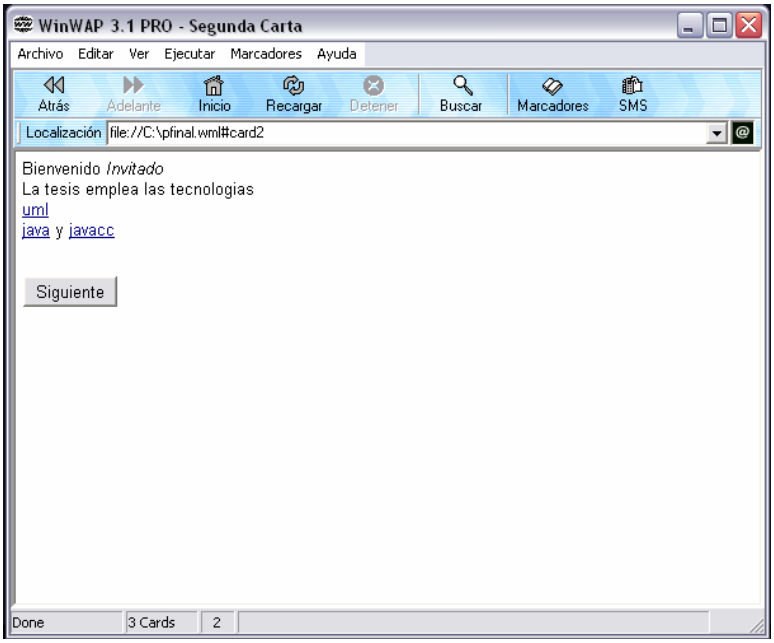


Figura VIII.12 *Segunda carta del archivo pfinal.wml en Winwap ©*

En este momento tenemos dos opciones para avanzar a la siguiente carta, presionando el boton “Siguiente” o dando clic en la liga de “uml”, logramos así llegar a la carta tres, como se muestra en la figura VIII.13. Aún seguimos en el archivo pfinal.wml.

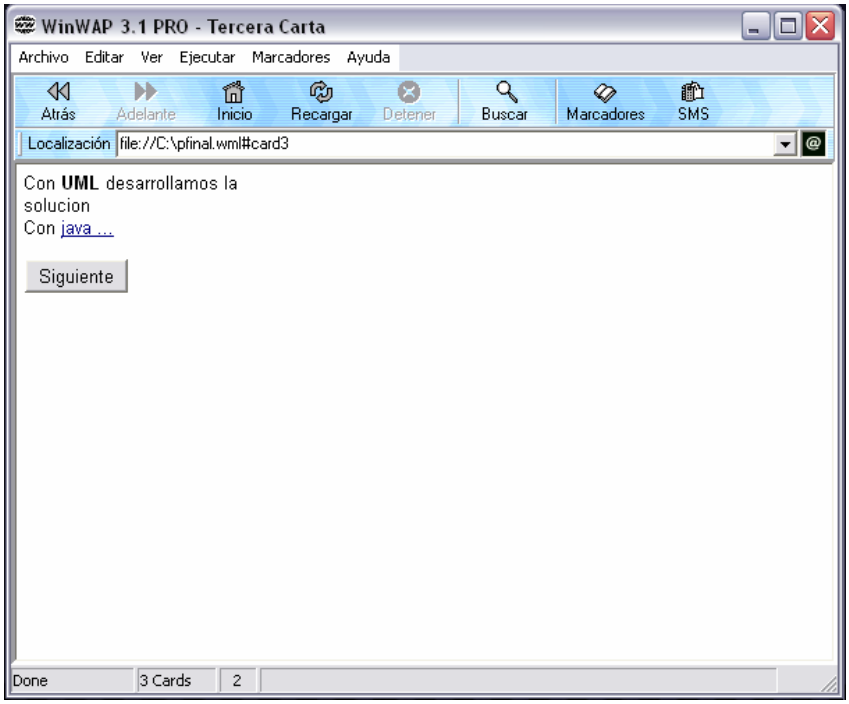


Figura VIII.13 *Tercera carta del archivo pfinal.wml en Winwap ©*

En este momento es cuando vamos a pasar a otro archivo (foo.wml) y para hacerlo debemos dar clic en "java" o en el botón "Siguiente", haciendo esto conseguimos llegar a la carta cuatro que se muestra en la figura VIII.14:

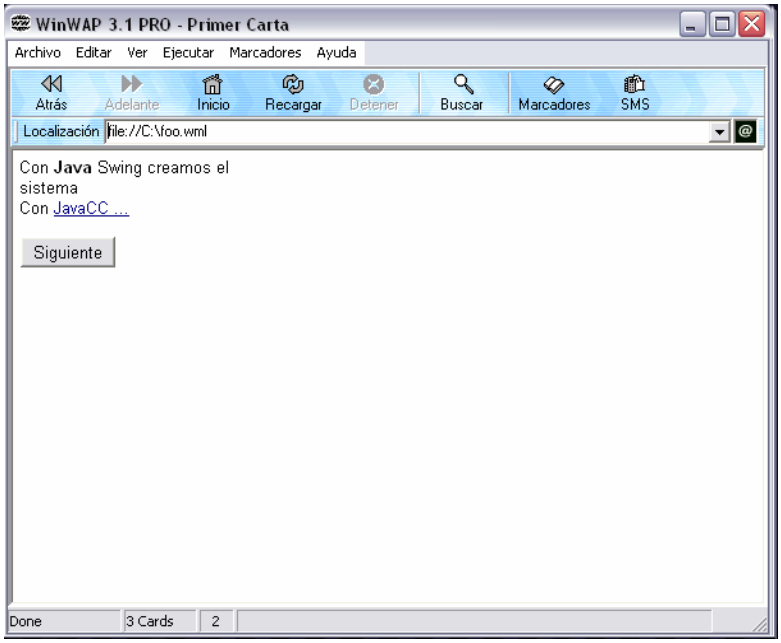


Figura VIII.14 *Cuarta carta, archivo foo.wml en Winwap ©*

La siguiente carta es la quinta en orden cronológico, pero la segunda en el archivo foo.wml. Se puede ver en la figura VIII.15

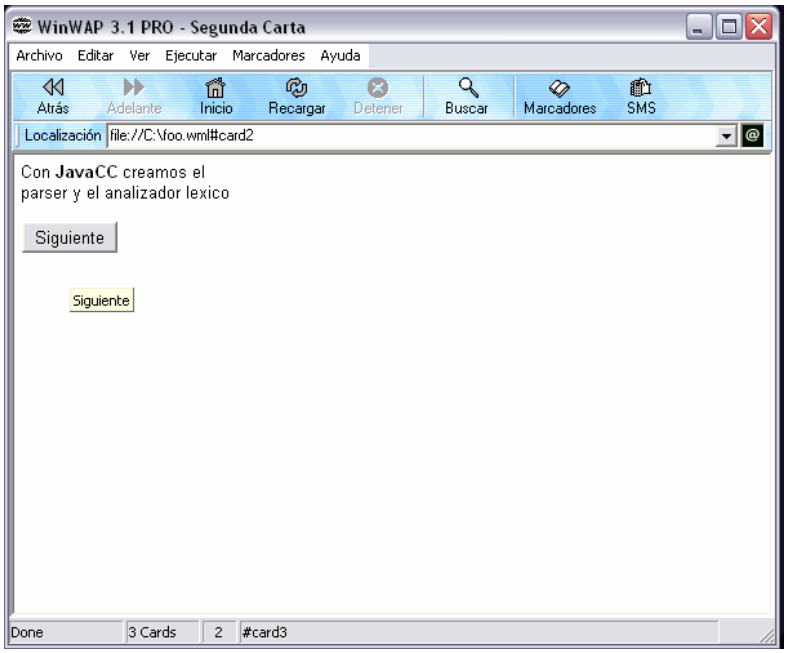


Figura VIII.15 *Quinta carta, segunda en el archivo foo.wml en Winwap ©*

La siguiente carta sólo puede ser alcanzada haciendo clic en el botón “Siguiente”, lo que obtenemos se muestra en la figura VIII.16

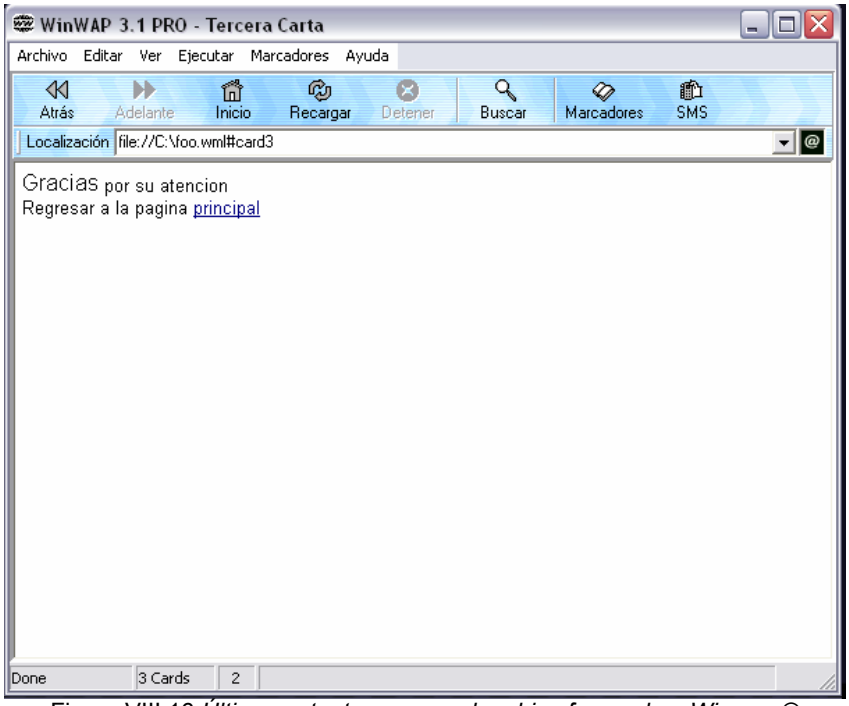


Figura VIII.16 *Última carta, tercera en el archivo foo.wml en Winwap ©*

De esta última carta se hace una liga hacia la primera carta en el archivo pfinal.wml.

Sobre Wapaka ©. El procedimiento será el mismo, cambiando solamente la apariencia. Así, la primera carta, después de cargar el archivo, será como se muestra en la figura VIII.17

Figura VIII.17 *Primera carta, archivo pfinal.wml en Wapaka ©*

Al dar clic en el botón siguiente llegamos a la segunda carta, como se muestra en la figura VIII.18

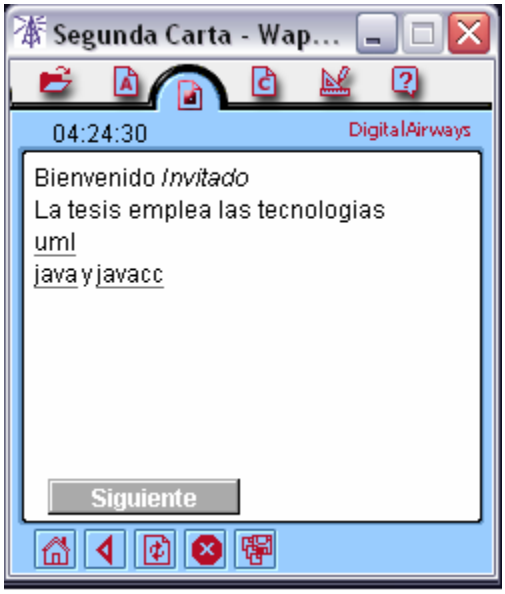


Figura VIII.18 *Segunda carta, archivo pfinal.wml en Wapaka ©*

Del mismo modo, al dar clic en “uml” o en el botón “Siguiente”, llegamos a la tercera carta, como se aprecia en la figura VIII.19

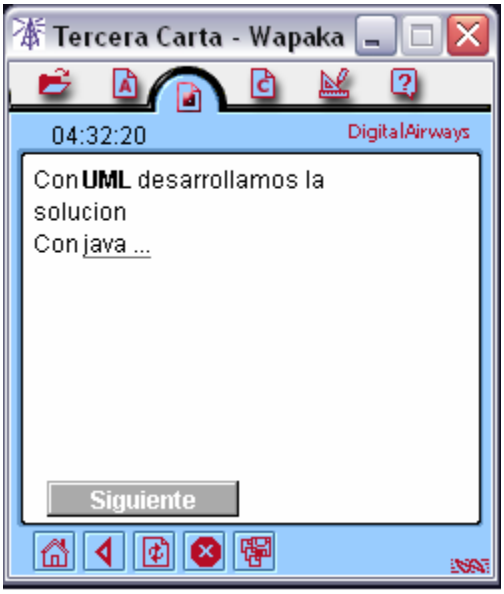


Figura VIII.19 *Tercera carta, archivo pfinal.wml en Wapaka* ©

La siguiente carta, como se había explicado anteriormente, se encuentra en el archivo foo.wml, la carta la podemos ver en la figura VIII.20

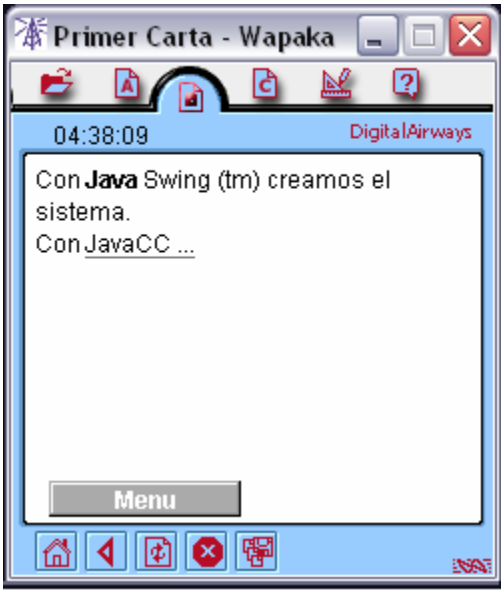


Figura VIII.20 *Cuarta carta, archivo foo.wml en Wapaka* ©

La siguiente carta se puede apreciar el la figura VIII.21

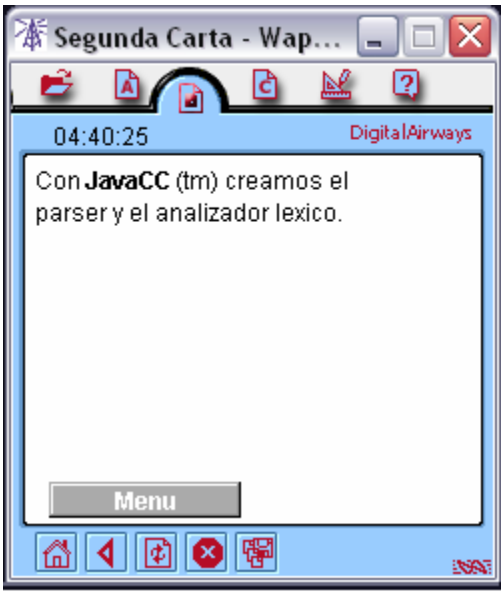


Figura VIII.21 *Quinta carta, archivo foo.wml en Wapaka* ©

De igual manera al dar clic en “Siguiente” logramos llegar a la última carta, como se ve en la figura VIII. 22.

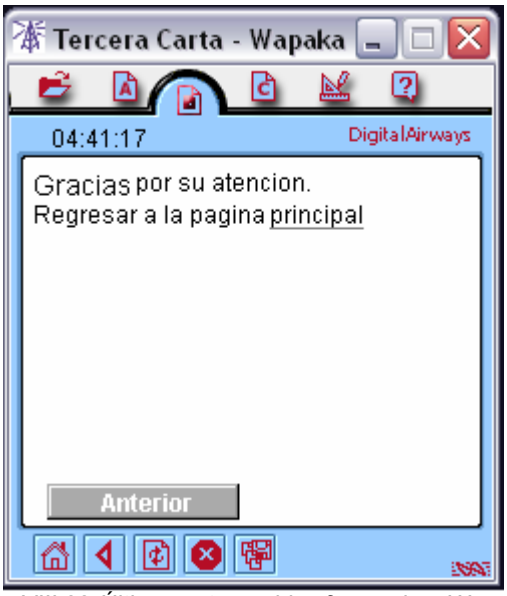


Figura VIII.22 *Última carta, archivo foo.wml en Wapaka* ©

En el SEEDWML. El procedimiento sigue siendo el mismo, de la figura VIII. 23 a la figura VIII. 25, se muestra la navegación del SEEDML sobre el archivo pfinal.wml.

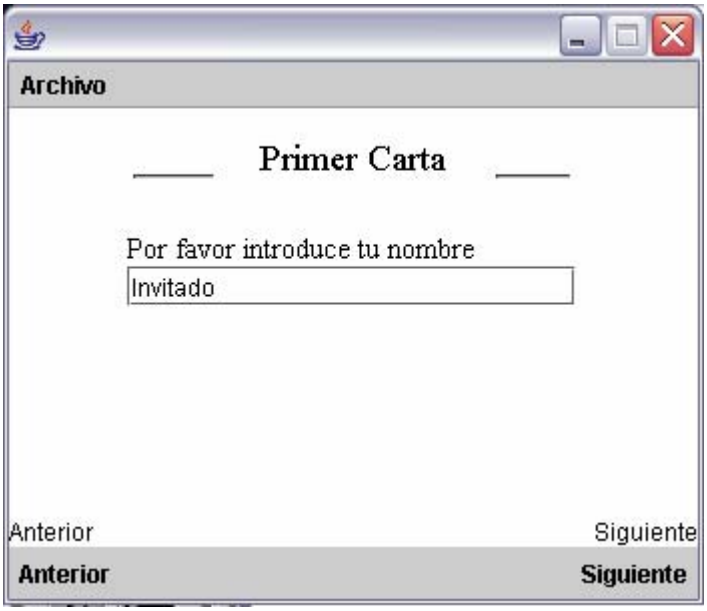


Figura VIII.23 *Primera carta, archivo pfinal.wml en SEEDWML*

Figura VIII.24 *Segunda carta, archivo pfinal.wml en SEEDWML*

Figura VIII.25 *Tercera carta, archivo pfinal.wml en SEEDWML*

Con dar clic en el botón "Siguiente" o en "java", pasamos a la siguiente carta que partenece al archivo foo.wml. Lo podemos ver de las figuras VIII.26 a VIII.28

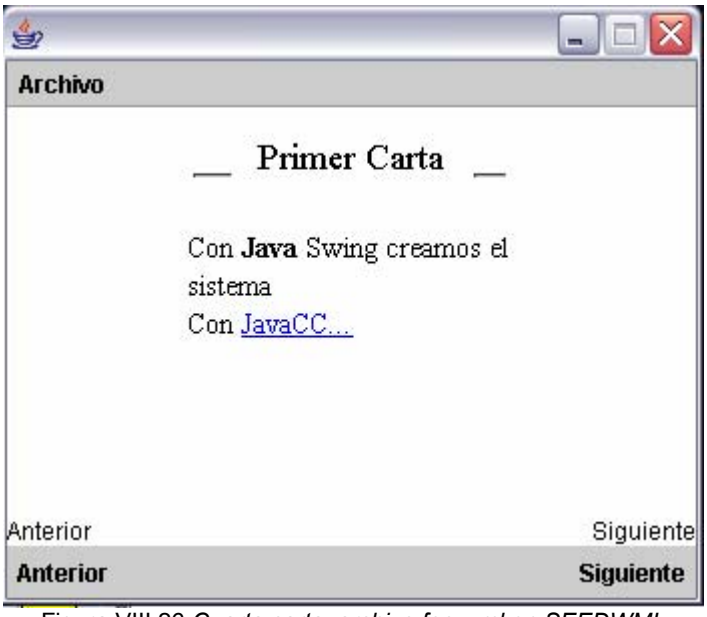


Figura VIII.26 *Cuarta carta, archivo foo.wml en SEEDWML*

Figura VIII.27 *Quinta carta, archivo foo.wml en SEEDWML*

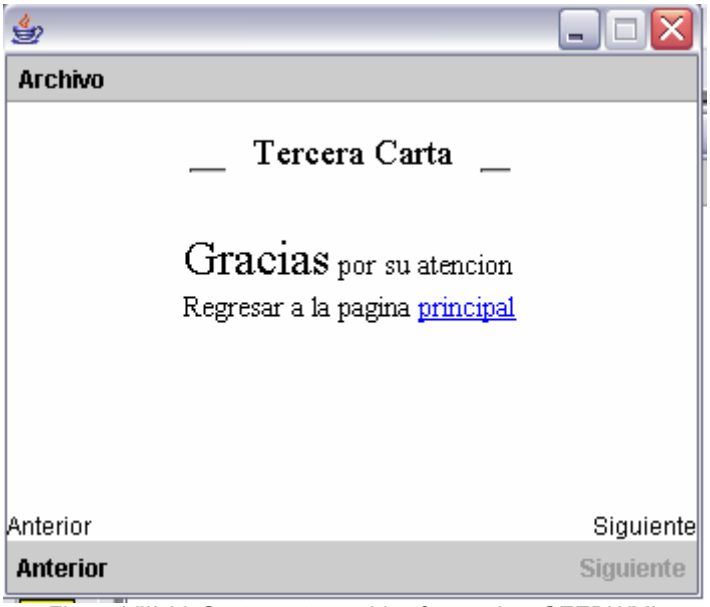


Figura VIII.28 *Sexta carta, archivo foo.wml en SEEDWML*

En la tercera y última prueba, se evalúa la capacidad para el manejo de variables, se emplea el archivo pruebavar.wml. El archivo está diseñado para que el funcionamiento sea como sigue: en la primera carta se debe desplegar a un lado de la etiqueta que lo indica, el valor de variable1. Al pasar a la siguiente carta, dado que tenemos en el evento *onenterforward* la asignación del valor de la variable2, no se deberá desplegar éste en el browser, así que seguimos mostrando solamente el valor de variable1. Al pasar a la siguiente carta, se debe desplegar el valor de variable3 y mostrarse al igual que el valor de variable1 a un lado de la etiqueta que indica su nombre. En este punto sólo podemos regresar a la carta anterior (carta dos) y es en este momento cuando se asigna el valor a variable2 y se despliega.

En Winwap © podemos ver que en las tres primeras pantallas el manejo de memoria se lleva correctamente sin embargo, la administración de las variables con eventos muestra un error de diseño en este programa. Como se puede apreciar en las figuras VIII.29, VIII.30 y VIII.31, es correcto el funcionamiento, en la figura VIII.32 se puede apreciar más claramente el error.

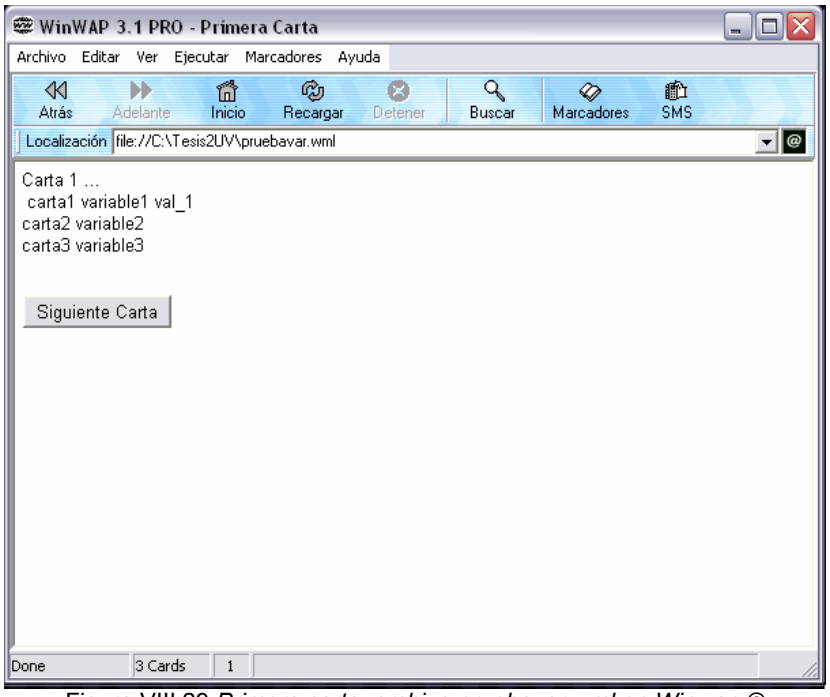


Figura VIII.29 *Primera carta, archivo pruebavar.wml en Winwap ©*

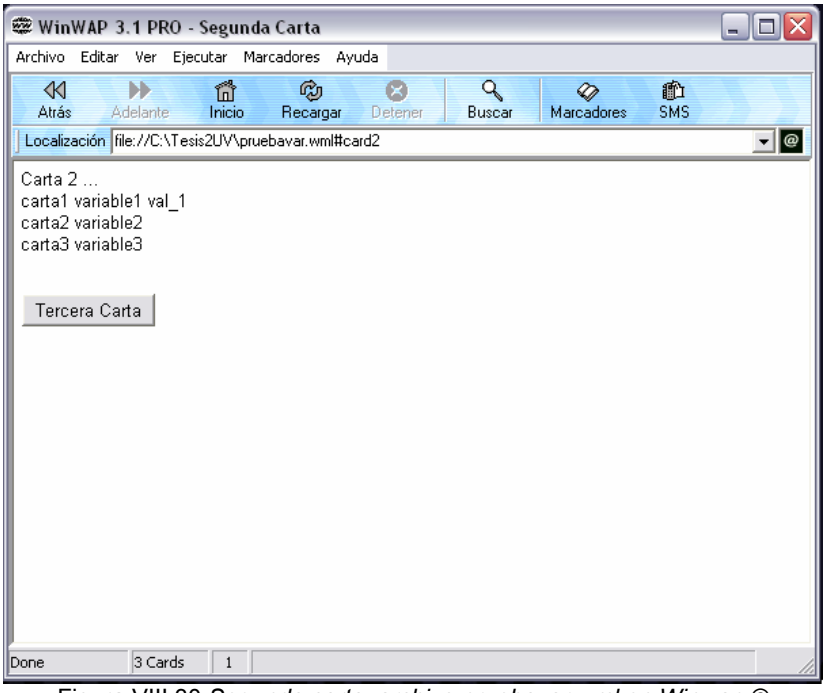


Figura VIII.30 *Segunda carta, archivo pruebavar.wml en Winwap ©*

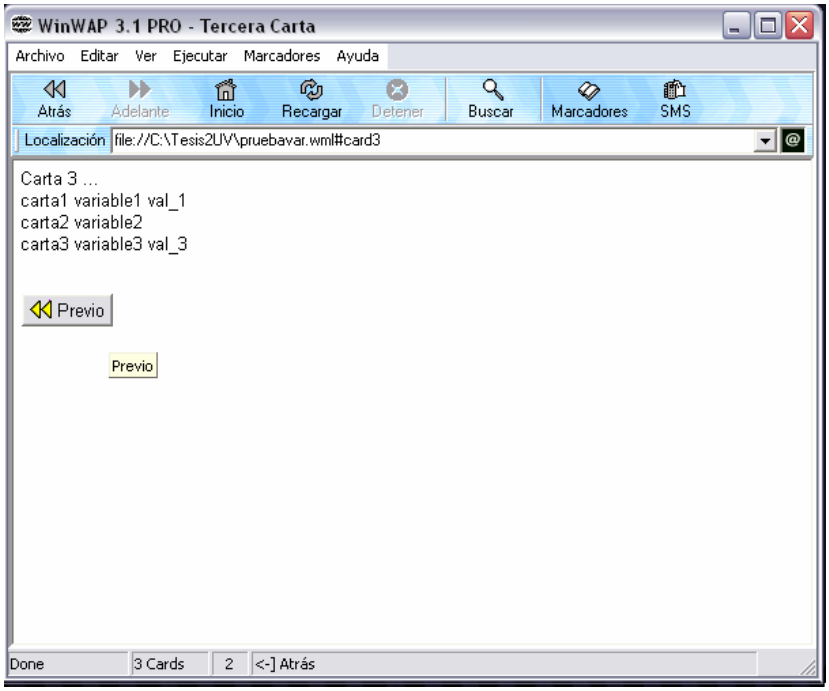


Figura VIII.31 *Tercera carta, archivo pruebavar.wml en Winwap ©*

Hasta aquí el funcionamiento es correcto, el problema aparece en el momento en que regresamos a la carta dos, como se ve en la figura VIII.32.

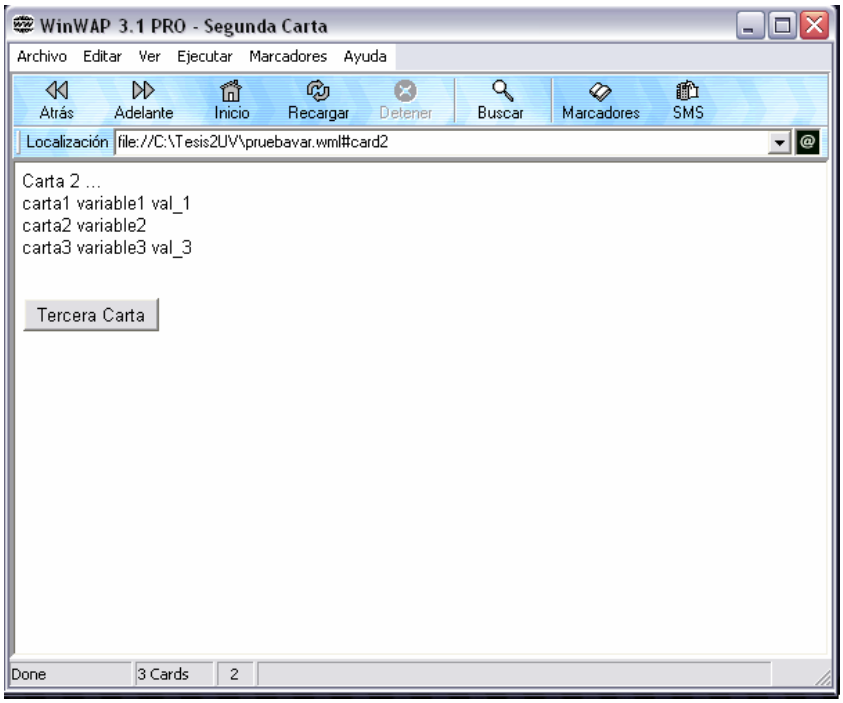


Figura VIII.32 *Regreso a la segunda carta, archivo pruebavar.wml en Winwap ©*

En este momento debería aparecer desplegado el valor de variable2, lo cual refleja un error de diseño.

En Wapaka ©, a diferencia de Winwap ©, el funcionamiento es correcto, las figuras VIII.33 a VIII.36 muestran el manejo correcto de las variables:

Figura VIII.33 *Primera carta, archivo pruebavar.wml en Wapaka ©*

Figura VIII.34 *Segunda carta, archivo pruebavar.wml en Wapaka ©*

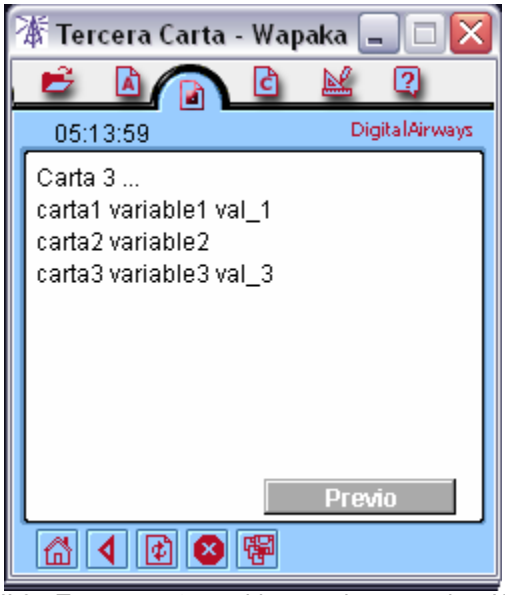


Figura VIII.35 *Tercera carta, archivo pruebavar.wml en Wapaka* ©

En este momento es cuando, por medio del evento *onenterbackward* de la carta dos, se asigna el valor de variable2, como se muestra en figura VIII.36

Figura VIII.36 *Segunda carta, todos los valores asignados, archivo pruebavar.wml en Wapaka* ©

En el SEEDWML el manejo de las variables se lleva de manera correcta, como se puede apreciar en la siguientes cuatro figuras.

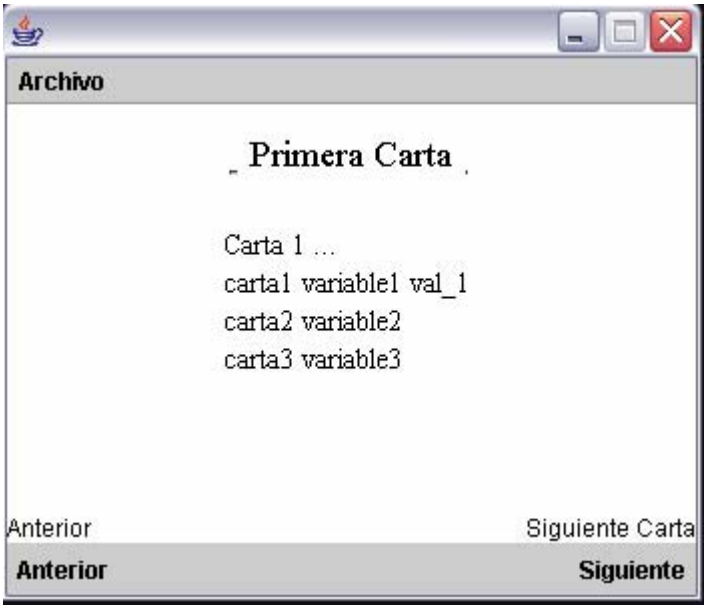


Figura VIII.37 *Primera carta, archivo pruebavar.wml en SEEDWML*

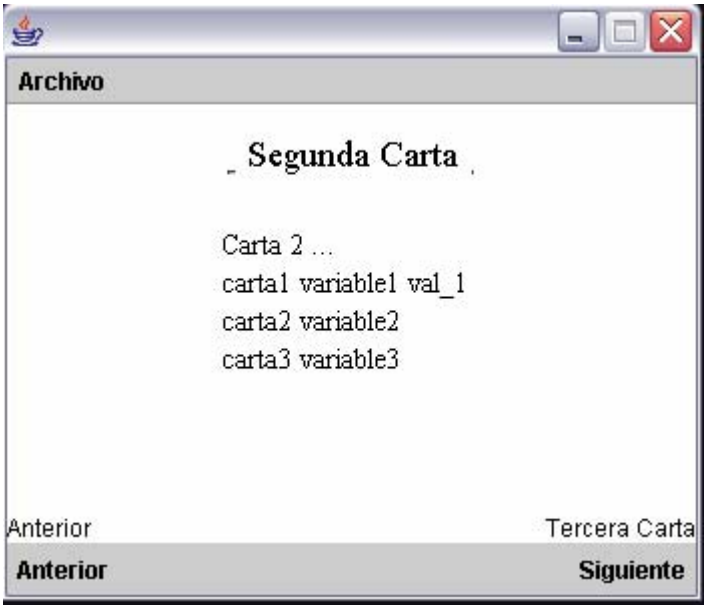


Figura VIII.38 *Segunda carta, archivo pruebavar.wml en SEEDWML*

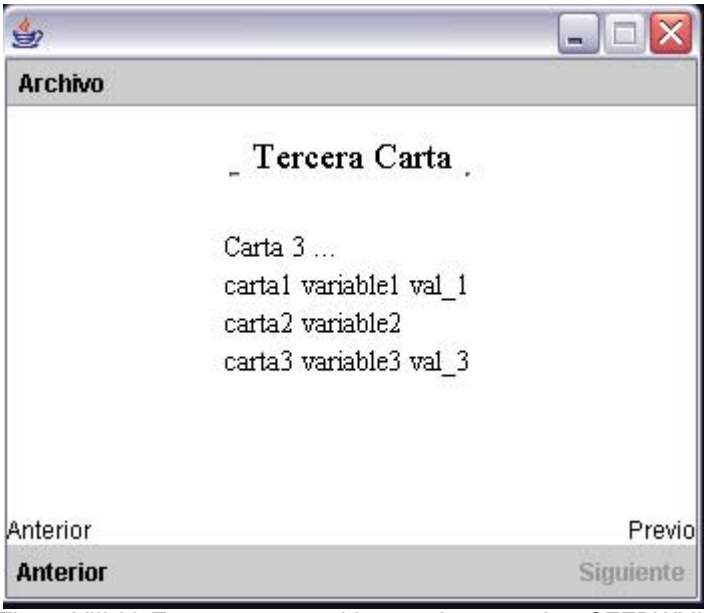


Figura VIII.39 *Tercera carta, archivo pruebavar.wml en SEEDWML*

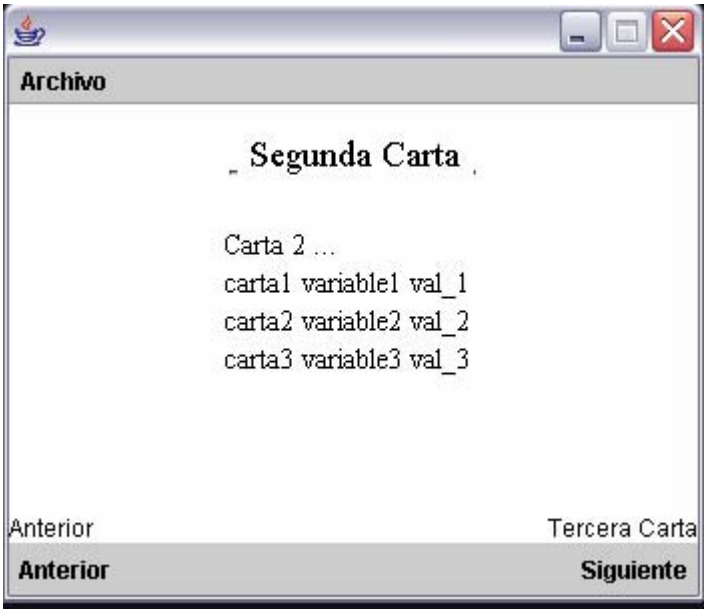


Figura VIII.40 *Segunda carta, valores bien asignados, archivo pruebavar.wml en SEEDWML*

Para evaluar el parser de los emuladores, deliberadamente se cometieron errores en la sintaxis de cada archivo.

En el archivo pruebatabla.wml donde debía cerrarse un elemento p ( ), se eliminó el cierre del elemento dejando la sintaxis de nuevo elemento p ( ).

En el caso de Winwap © se muestra el resultado en la figura VIII. 41:

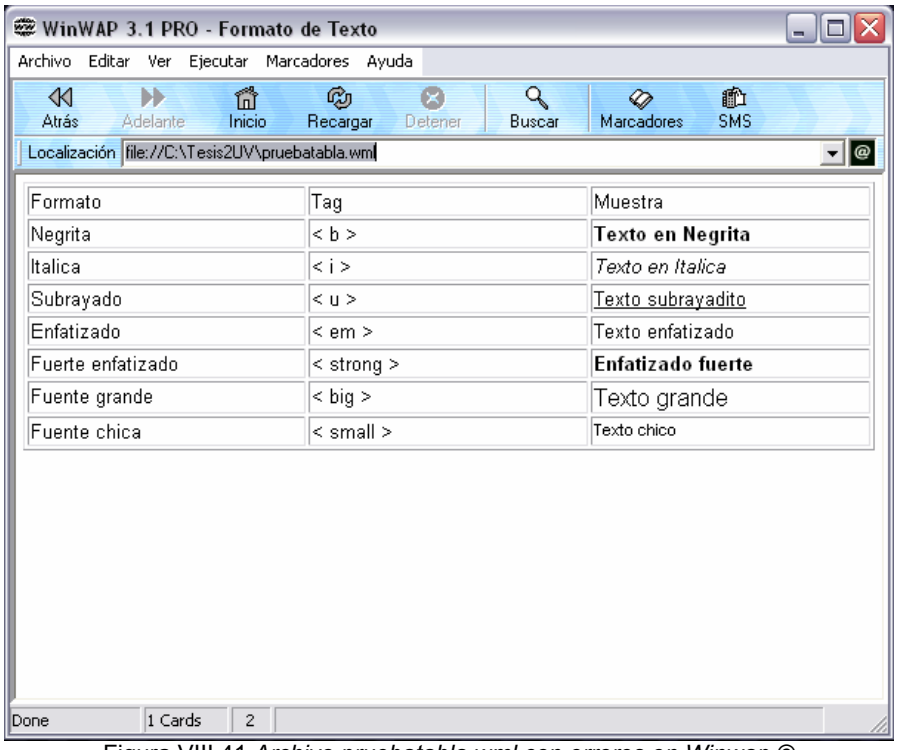


Figura VIII.41 *Archivo pruebatabla.wml con errores en Winwap ©*

Donde es claro que no detectó el error.

En el caso de Wapaka © el resultado se ve en la figura VIII.42

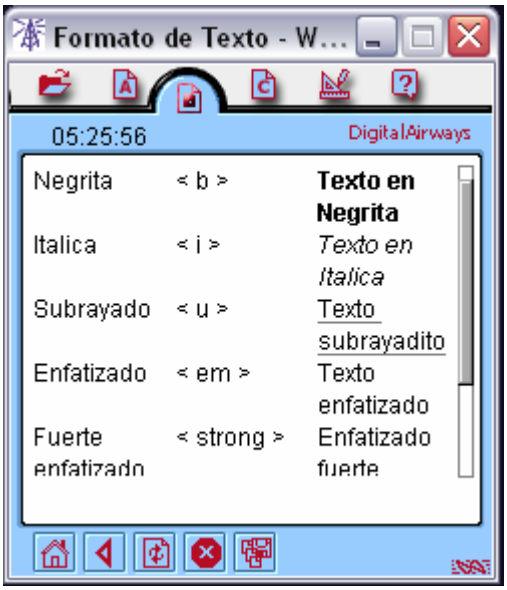


Figura VIII.42 *Archivo pruebatabla.wml con errores en Wapaka ©*

Tampoco es detectado el error.

En la figura VIII.43, se muestra la respuesta del SEEDWML

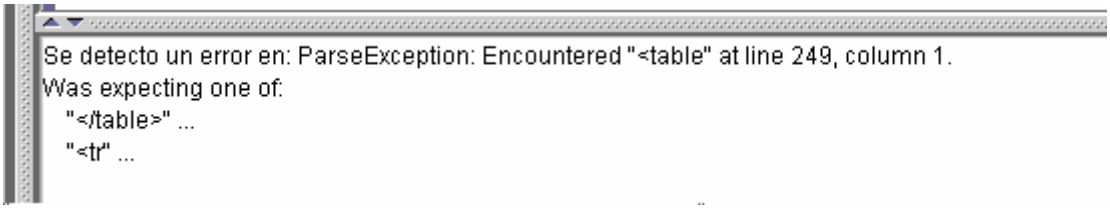


Figura VIII.43 *Archivo pruebatabla.wml con errores en SEEDWML*

En el SEEDWML mientras existan errores sintácticos o léxicos no se podrá mostrar la página emulada. El error se presenta en el área inferior del editor.

En el archivo pfinal.wml donde el enunciado era <go href="#card2"/> se cambió por: <go href="#card2">

La respuesta de Winwap© se presenta en la figuraVIII.44

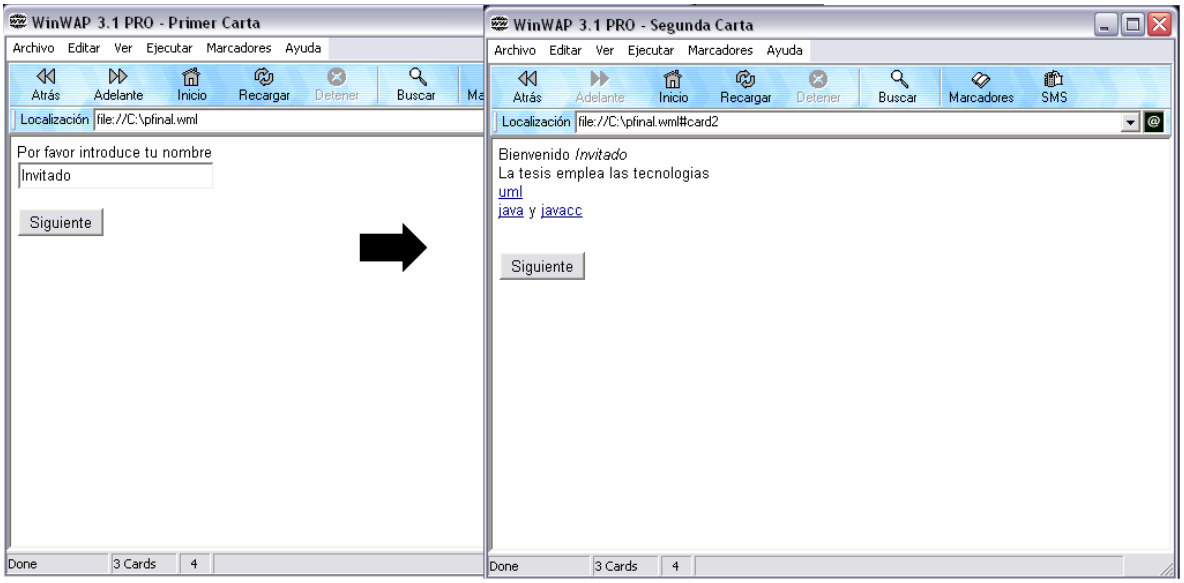


Figura VIII.44 *Archivo pfinal.wml con errores en Winwap©*

No obstante el error provocado, la página se muestra igual que si no hubiera error.

La respuesta de Wapaka © se muestra en la figura VIII. 45

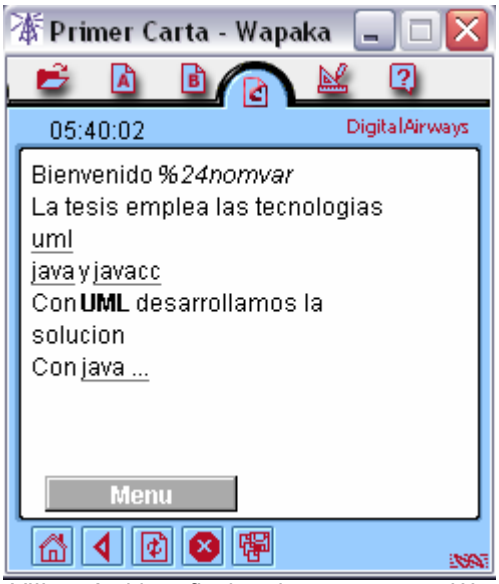


Figura VIII.45 *Archivo pfinal.wml con errores en Wapaka ©*

La figura VIII.46 muestra el desempeño del SEEDWML ante este error

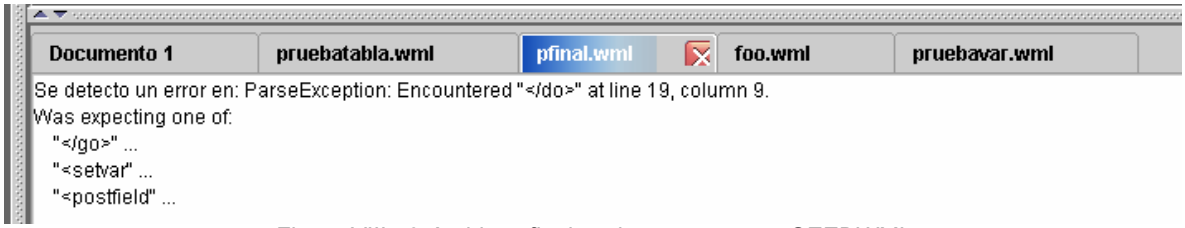


Figura VIII.46 *Archivo pfinal.wml con errores en SEEDWML*

Por último, en el archivo pruebavar.wml se cambió el enunciado </onevent> por <onevent>

La figura VIII.47 muestra la respuesta de Winwap ©

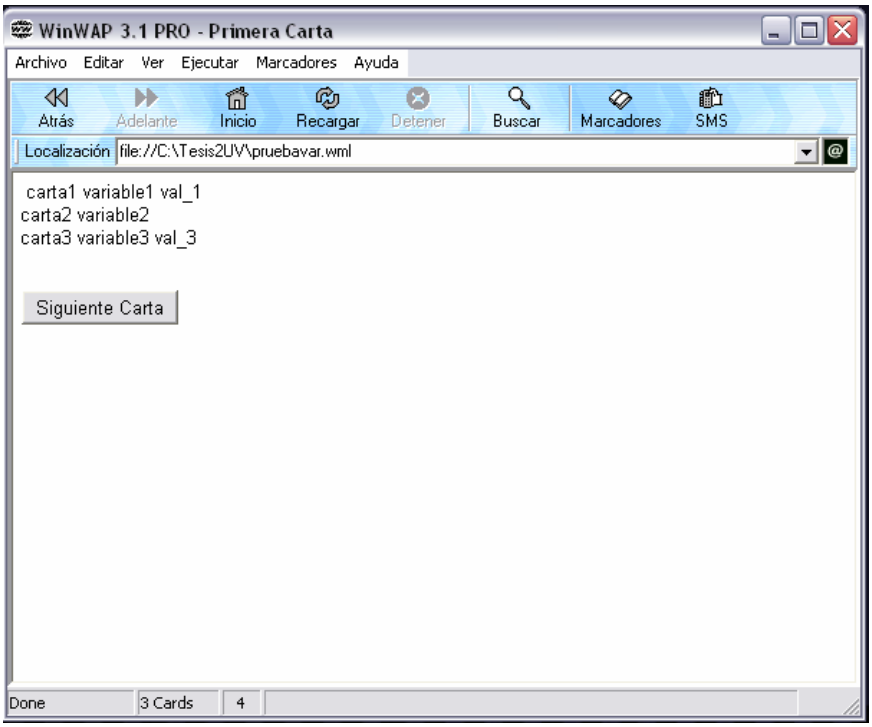


Figura VIII.47 *Archivo pruebavar.wml con errores en Winwap ©*

La respuesta de Wapaka © ante este error se muestra en la figura VIII.48

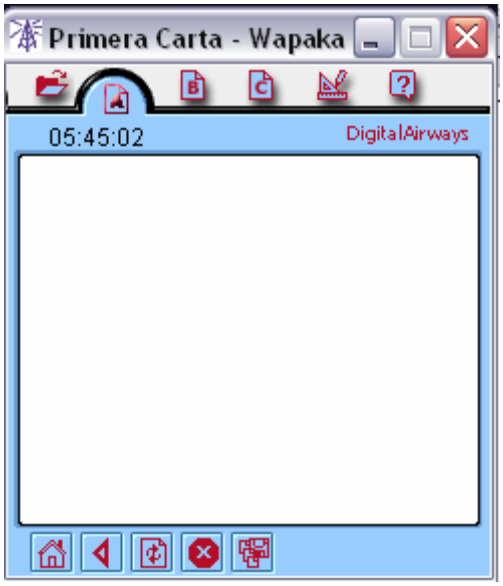


Figura VIII.48 *Archivo pruebavar.wml con errores en Wapaka ©*

En el SEEDWML la respuesta es como la muestra la figura VIII.49

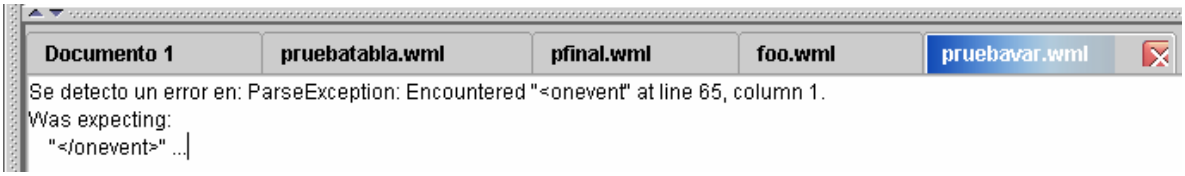


Figura VIII.49 *Archivo pruebavar.wml con errores en SEEDWML*

VIII.2.5 Defecto

Un defecto es una anomalía del sistema. Un defecto puede ser utilizado para localizar los puntos que se deben controlar y resolver.

En el caso del SEEDWML, encontramos que presta atención a detalles que otros productos como Winwap © y Wapaka© pasaron por alto; sin mencionar que el hecho de contar con una área de despliegue de errores en la cual se reflejen los problemas encontrados por el parser o el analizador léxico, es una ventaja enorme sobre aquellos productos que dependen de un editor externo y además molesto para un usuario estar pasando de un programa a otro, siendo que en uno solo se encuentra todo lo que necesita, como el SEEDWML, por ejemplo.

VIII.2.6 Evaluación de prueba

Una evaluación de prueba, es básicamente una evaluación de los resultados de los esfuerzos de prueba.

Como pudimos apreciar, el SEEDWML aprobó satisfactoriamente las pruebas diseñadas para su evaluación. Debemos tener en cuenta que el modelo de pruebas está pensado dentro del alcance del proyecto.

Si comparamos contra los otros dos productos comerciales, veremos que dentro del alcance del proyecto y basados en el modelo de pruebas, el SEEDWML es un producto que está a la altura de ellos. Cabe mencionar que la licencia para el Winwap © cuesta $35 USD. El Wapaka es un producto de distribución libre (freeware) pero sustentado por un grupo de desarrollo de más de cinco años que se dedican a la creación de programas orientados a brindar soluciones para dispositivos de comunicación.

En el caso del SEEDWML el proceso llevado a cabo hasta el momento es un proceso iterativo e incremental, siendo este último flujo de trabajo (prueba) el más sencillo (en comparación con los anteriores) pero no por eso el menos importante, ya que por medio de las pruebas es como nos vamos dando cuenta de las limitaciones del sistema y es así también como vamos generando nuevas especificaciones y el nivel de detalle va aumentando, creando así nuevas metas sobre los modelos de análisis y diseño, provocando que, dentro de los alcances de la propuesta, logremos un producto funcional de calidad, que indudablemente siempre puede mejorarse bajo una serie de nuevas iteraciones, y un buen diseño, así como un buen análisis y planificar bien las pruebas.

Referencias

[Jacobson *et al*, 2000] Ivar Jacobson, James Rumbaugh , Grady Booch. *El proceso unificado de desarrollo de software*. Addison Wesley Iberoamericana, Madrid, 2000

Conclusiones

Hemos diseñado e implementado un prototipo de interfaz de desarrollo gráfico para la edición de documentos WML que abarca un espectro considerable de requerimientos que tienen los programadores y desarrolladores de software en general; basándose en las habilidades y expectativas de éstos.

Después de observar el desarrollo iterativo e incremental del proceso de desarrollo y sus productos a lo largo de éste, podemos comprobar que no importa si el grupo de desarrollo es pequeño, el producto final será de una calidad innegable, como fue el caso del SEEDWML.

El SEEDWML combina un editor gráfico poderoso y de fácil acceso, y un emulador que se asemeja en un porcentaje muy grande al desempeño de un dispositivo móvil, cuya interacción con el usuario se lleva de manera sencilla e intuitiva, de manera que si requiere de ayuda, la tendrá al alcance de un clic (ver capítulo VII), pero si los requerimientos son más profesionales, el SEEDWML podrá satisfacer a cualquier programador gracias a la interfaz amigable y el rápido aprendizaje del cual nos provee este entorno de desarrollo.

Este proyecto se obtuvo a partir de la unión de diferentes tecnologías, todas poderosas y robustas. Éstas son Java ™ (con Java Swing ©), JavaCC © y UML™.

Cada tecnología tiene un papel fundamental en el desarrollo del SEEDWML, como se describe a continuación:

- Con java Swing © se genera el entorno gráfico y el desempeño del sistema se debe a la base de programación Java.

- Con JavaCC generamos el parser, el analizador léxico, y anexamos la funcionalidad del emulador. Gracias a la flexibilidad de JavaCC, podemos incluir código Java de manera embebida en el archivo .jj, el cual genera los archivos .java que a la postre serán el parser y el analizador léxico.

- Con UML tenemos un sustento para la planeación y desarrollo del proceso, dentro del cual se incluyen los requerimientos, el diseño, el análisis y las pruebas. UML™ nos dota también de un lenguaje gráfico para representación tanto del análisis como del diseño, con estructuras definidas para cada uno.

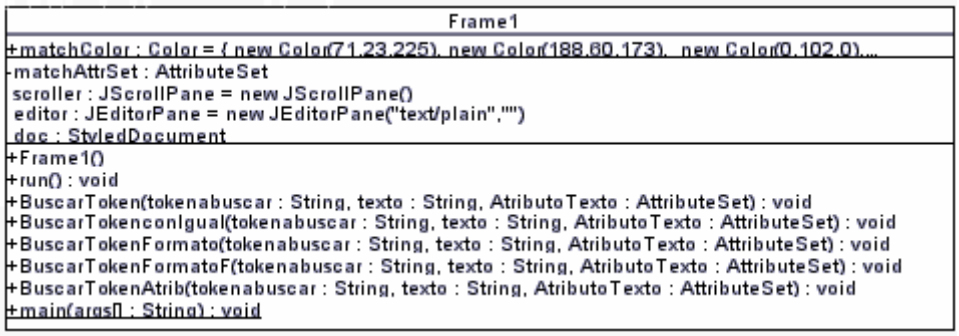
APÉNDICE A
Especificaciones de las clases

Main


tdItem : JMenuItem = new JMenuItem()
inputItem : JMenuItem = new JMenuItem()
selectItem : JMenuItem = new JMenuItem()
optionItem : JMenuItem = new JMenuItem()
optgroupItem : JMenuItem = new JMenuItem()
fieldsetItem : JMenuItem = new JMenuItem()
ayudaItem : JMenuItem = new JMenuItem()
frame : Frame1 = new Frame1[15]
desk : JDesktopPane = new JDesktopPane()
-optionPane : JOptionPane
changeLog : JTextArea = new JTextArea[15]
-btnlist : JList
-listButtons : Vector = new Vector()
-splitPane : JSplitPane
-splitPane2 : JSplitPane
-splitPane3 : JPanel
-botones : JPanel
Negrita : ImageIcon = new ImageIcon("images/negrita.PNG")
Italica : ImageIcon = new ImageIcon("images/italica.PNG")
Sub : ImageIcon = new ImageIcon("images/subrayado.PNG")
Grande : ImageIcon = new ImageIcon("images/grande.PNG")
Chica : ImageIcon = new ImageIcon("images/chica.PNG")
Enfatizada : ImageIcon = new ImageIcon("images/enfatizada.PNG")
Fuerte : ImageIcon = new ImageIcon("images/strong.PNG")
Parrafo : ImageIcon = new ImageIcon("images/parrafo.PNG")
Brinco : ImageIcon = new ImageIcon("images/br.PNG")
SEspacio : ImageIcon = new ImageIcon("images/nb.PNG")
Boton : ImageIcon = new ImageIcon("images/boton.PNG")
Ancla : ImageIcon = new ImageIcon("images/anchor.PNG")
idogo : ImageIcon = new ImageIcon("images/do_go.PNG")
idogoe : ImageIcon = new ImageIcon("images/do_goe.PNG")
idoprev : ImageIcon = new ImageIcon("images/do_prev.PNG")
idonoop : ImageIcon = new ImageIcon("images/do_noop.PNG")
irefresh : ImageIcon = new ImageIcon("images/refresh.PNG")
isetvar : ImageIcon = new ImageIcon("images/setvar.PNG")
ionevent : ImageIcon = new ImageIcon("images/onevent.PNG")
ipostfield : ImageIcon = new ImageIcon("images/postfield.PNG")
ianchor : ImageIcon = new ImageIcon("images/anchor_big.PNG")
iimg : ImageIcon = new ImageIcon("images/img.PNG")
itimer : ImageIcon = new ImageIcon("images/timer.PNG")
itable : ImageIcon = new ImageIcon("images/table.PNG")
itr : ImageIcon = new ImageIcon("images/tr.PNG")
itd : ImageIcon = new ImageIcon("images/td.PNG")
iinput : ImageIcon = new ImageIcon("images/input.PNG")
iselect : ImageIcon = new ImageIcon("images/select.PNG")
ioption : ImageIcon = new ImageIcon("images/option.PNG")
ioptgroup : ImageIcon = new ImageIcon("images/optgroup.PNG")
ifieldset : ImageIcon = new ImageIcon("images/fieldset.PNG")
-btnTemplate : JButton = new JButton("Template")
-btnCarta : JButton = new JButton("Card")
-btnNegrita : JButton = new JButton(Negrita)
-btnItalica : JButton = new JButton(Italica)
-btnSubrayado : JButton = new JButton(Sub)
-btnGrande : JButton = new JButton(Grande)
-btnChica : JButton = new JButton(Chica)
-btnEnfatizada : JButton = new JButton(Enfatizada)
-btnFuerte : JButton = new JButton(Fuerte)
-btnParrafo : JButton = new JButton(Parrafo)
-btnBrinco : JButton = new JButton(Brinco)
-btnSEspacio : JButton = new JButton(SEspacio)
-btnBoton : JButton = new JButton(Boton)
-btnAncla : JButton = new JButton(Ancla)
-btndogo : JButton = new JButton(idogo)
-btndogoe : JButton = new JButton(idogoe)

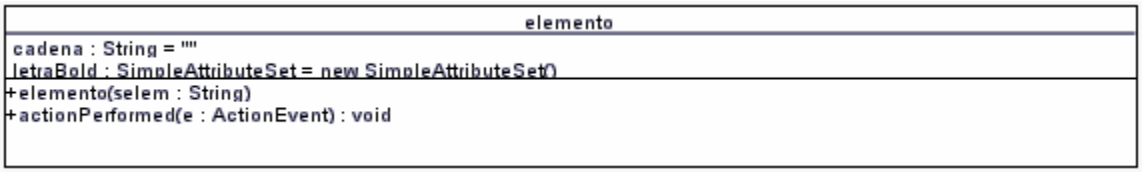
Compilar

Frame1

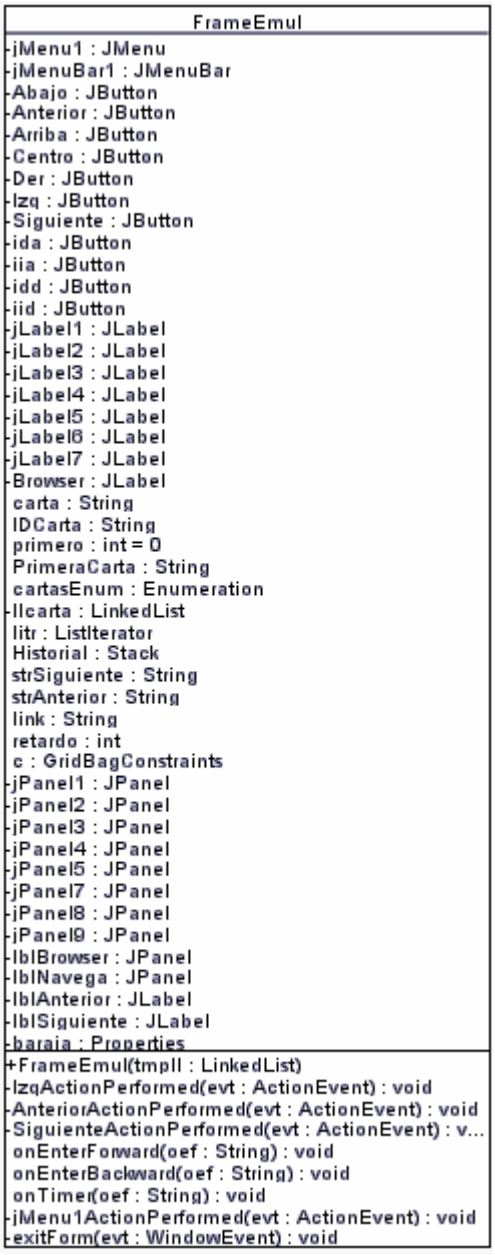


CardWindow

elemento



FrameEmul



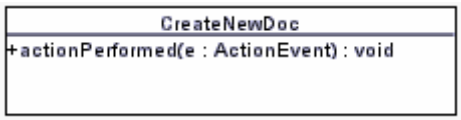
OpenHandler

SaveasHandler

SaveHandler

SalirHandler

CreateNewDoc



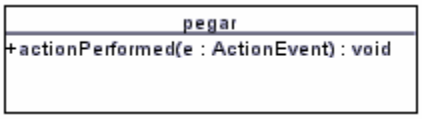
CaretListenerLabel

MyDocumentListener

Cortar

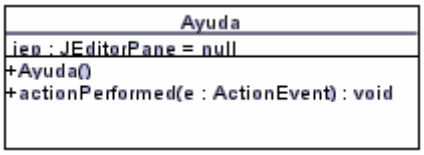
Copiar

Pegar

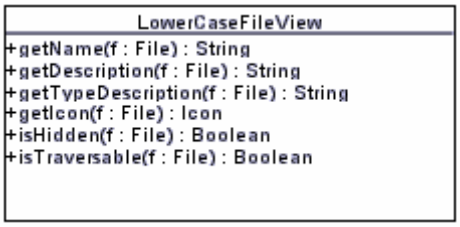


MyUndoableEditListener

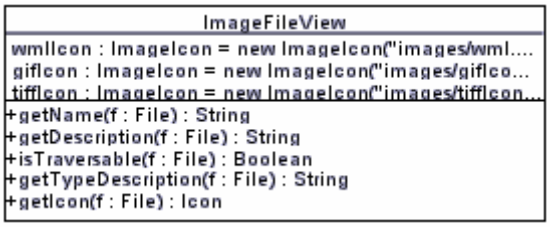
Ayuda



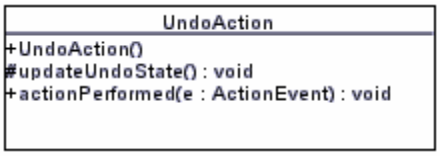
LowerCaseFileView



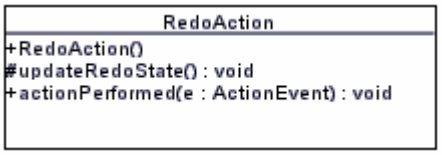
ImageFileView



UndoAction

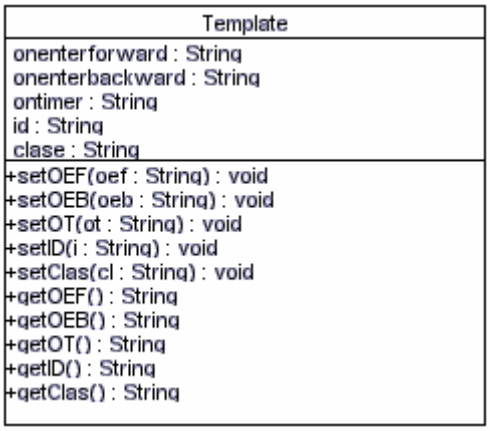


RedoAction



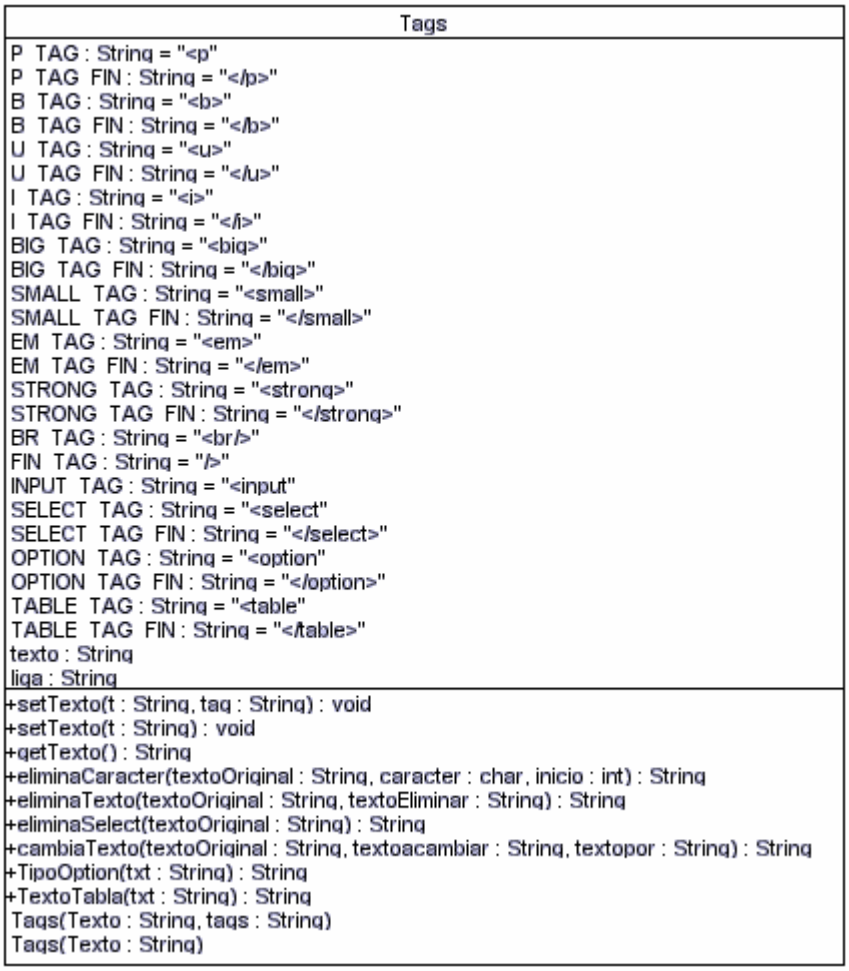
Cartas

Template



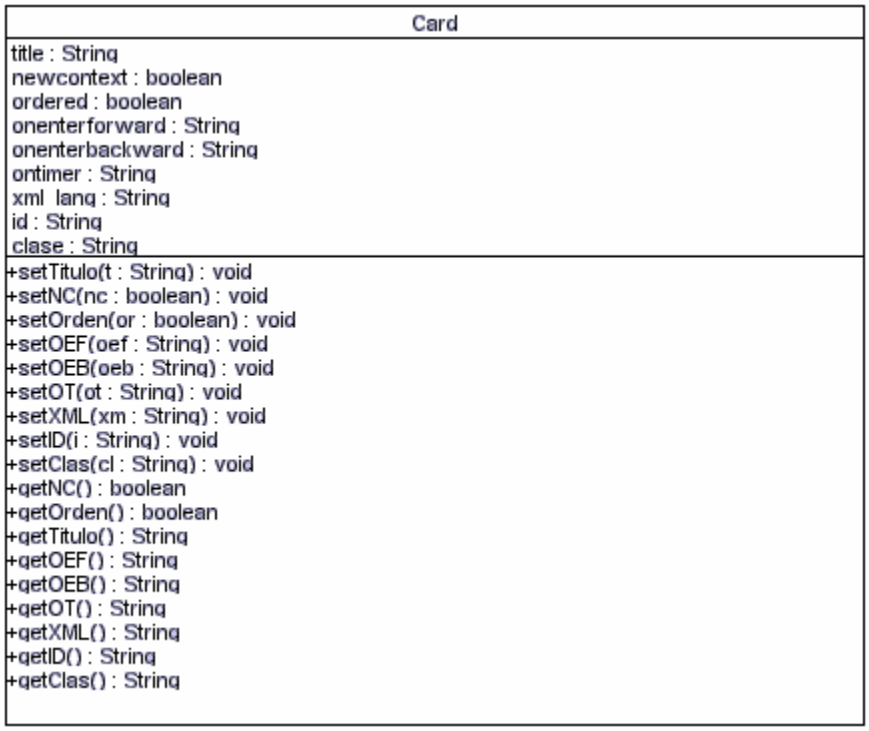
WMLFilter

Tags

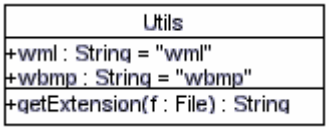


Tarea

Card

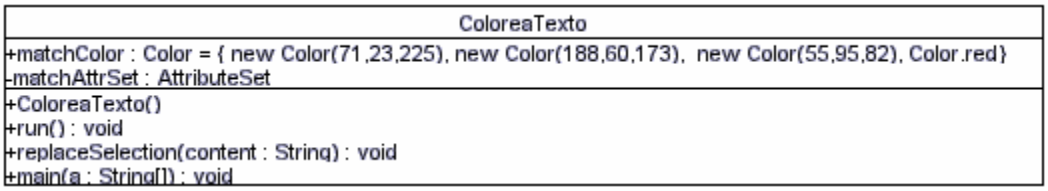


Utils

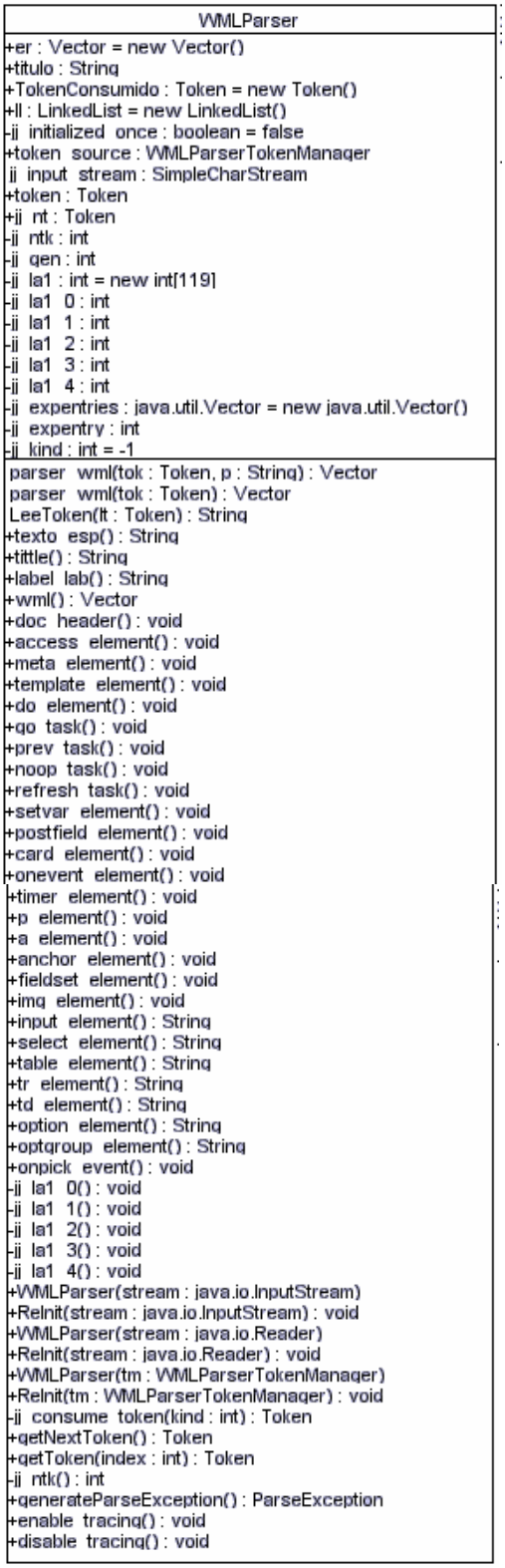


ColoreaVivo

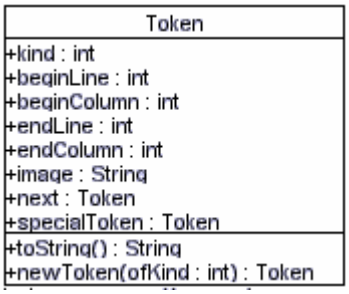
ColoreaTexto



WMLParser


WMLParser
+er : Vector = new Vector()
+titulo : String
+TokenConsumido : Token = new Token()
+ll : LinkedList = new LinkedList()
-jj_initialized_once : boolean = false
+token_source : WMLParserTokenManager
jj_input_stream : SimpleCharStream
+token : Token
+jj_nt : Token
-jj_ntk : int
-jj_gen : int
-jj_la1 : int = new int[119]
-jj_la1_0 : int
-jj_la1_1 : int
-jj_la1_2 : int
-jj_la1_3 : int
-jj_la1_4 : int
-jj_expentries : java.util.Vector = new java.util.Vector()
-jj_expentry : int
-jj_kind : int = -1
parser_wml(tok : Token, p : String) : Vector
parser_wml(tok : Token) : Vector
LeeToken(lt : Token) : String
+texto_esp() : String
+tittle() : String
+label_lab() : String
+wml() : Vector
+doc_header() : void
+access_element() : void
+meta_element() : void
+template_element() : void
+do_element() : void
+go_task() : void
+prev_task() : void
+noop_task() : void
+refresh_task() : void
+setvar_element() : void
+postfield_element() : void
+card_element() : void
+onevent_element() : void
+timer_element() : void
+p_element() : void
+a_element() : void
+anchor_element() : void
+fieldset_element() : void
+img_element() : void
+input_element() : String
+select_element() : String
+table_element() : String
+tr_element() : String
+td_element() : String
+option_element() : String
+optgroup_element() : String
+onpick_event() : void
-jj_la1_0() : void
-jj_la1_1() : void
-jj_la1_2() : void
-jj_la1_3() : void
-jj_la1_4() : void
+WMLParser(stream : java.io.InputStream)
+ReInit(stream : java.io.InputStream) : void
+WMLParser(stream : java.io.Reader)
+ReInit(stream : java.io.Reader) : void
+WMLParser(tm : WMLParserTokenManager)
+ReInit(tm : WMLParserTokenManager) : void
-jj_consume_token(kind : int) : Token
+getNextToken() : Token
+getToken(index : int) : Token
-jj_ntk() : int
+generateParseException() : ParseException
+enable_tracing() : void
+disable_tracing() : void

Token



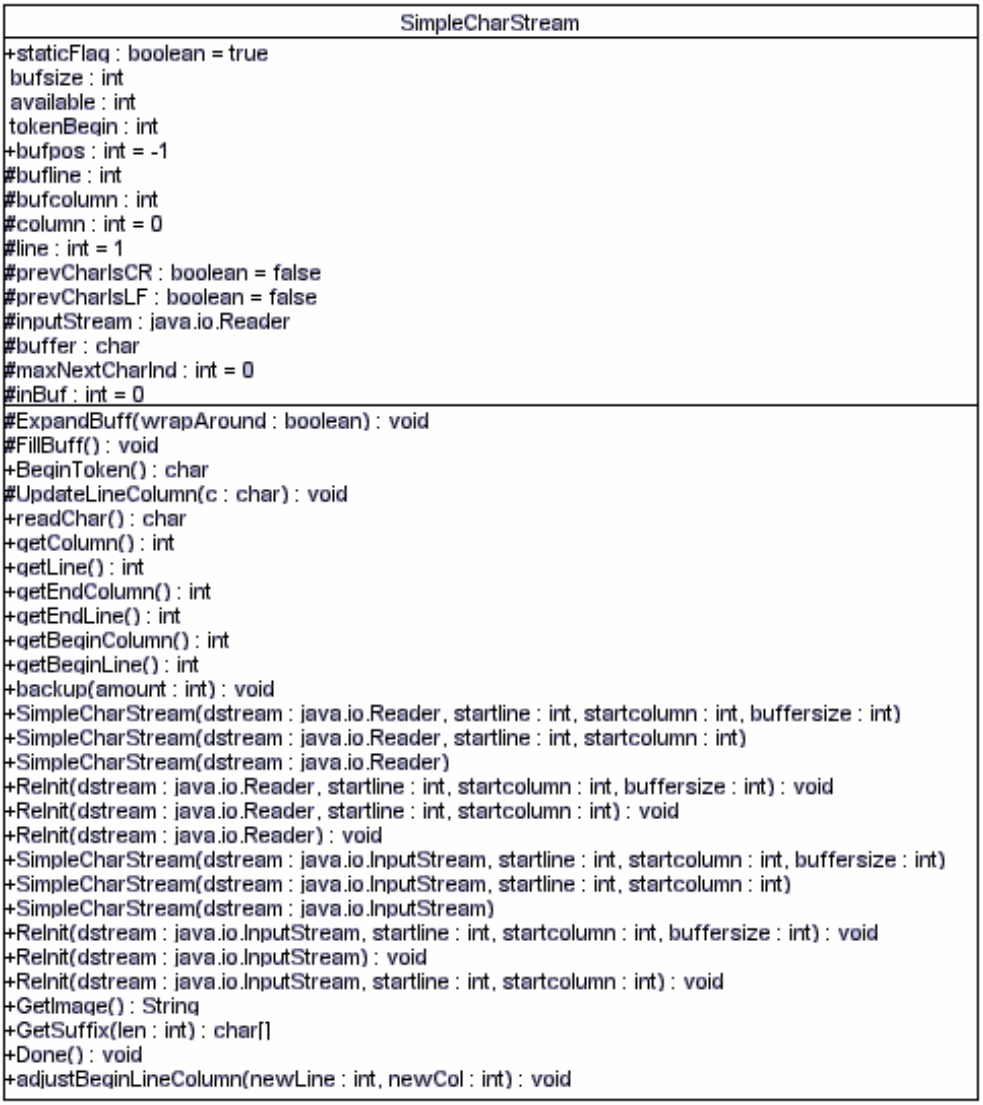
WMLParserConstants

```
  int EOF = 0;                          String[] tokenImage = {               "\" \"",
  int CHARS = 1;                          "<EOF>",                            "\"\\t\"",
  int ENTIDAD = 2;                        "<CHARS>",                          "\"\\n\"",
  int FIN = 3;                            "<ENTIDAD>",                        "\"\\r\"",
  int BOOLEANO = 4;                       "\">\"",                            "\"\\n\\r\"",
  int NUMBER = 5;                         "<BOOLEANO>",                       "\"\\\"\"",
  int NUMBER_ATR = 6;                     "<NUMBER>",                         "\"\\b\"",
  int SIMBOL = 7;                         "<NUMBER_ATR>",                     "<COMENTARIO>",
  int LENGTH = 8;                         "<SIMBOL>",                         "<QUOTEDSTRA>",
  int CDATA = 9;                          "<LENGTH>",                         "\"<wml\"",
  int NMTOKEN = 10;                       "<CDATA>",                          "\"</wml>\"",
  int TITLE = 11;                         "<NMTOKEN>",                        "\"<head>\"",
  int LABEL = 12;                         "<TITLE>",                          "\"</head>\"",
  int VAR = 13;                           "<LABEL>",                          "\"<access\"",
  int VAR_REF = 14;                       "<VAR>",                            "\"<meta\"",
  int VDATA = 15;                         "<VAR_REF>",                        "\"/>\"",
  int VAR_DEF = 16;                       "<VDATA>",                          "\"<template\"",
  int ID_LAB = 17;                        "<VAR_DEF>",                        "\"</template>\"",
  int ALT_LAB = 18;                       "<ID_LAB>",                         "\"<do\"",
  int SRC_LAB = 19;                       "<ALT_LAB>",                        "\"</do>\"",
  int NAME_LAB = 20;                      "<SRC_LAB>",                        "\"<go\"",
  int VALUE_LAB = 21;                     "<NAME_LAB>",                       "\"</go>\"",
  int DOMAIN_LAB = 22;                    "<VALUE_LAB>",                      "\"<prev/>\"",
  int SIZE_LAB = 23;                      "<DOMAIN_LAB>",                     "\"<noop/>\"",
  int TYPE_DEFS = 24;                     "<SIZE_LAB>",                       "\"<refresh\"",
  int TYPEOE_DEF = 25;                    "<TYPE_DEFS>",                      "\"</refresh>\"",
  int TYPEINPUT_DEF = 26;                 "<TYPEOE_DEF>",                     "\"<setvar\"",
  int CLASS_LAB = 27;                     "<TYPEINPUT_DEF>",                  "\"<postfield\"",
  int HREF_LAB = 28;                      "<CLASS_LAB>",                      "\"<card\"",
  int PATH_LAB = 29;                      "<HREF_LAB>",                       "\"</card>\"",
  int HTTP_LAB = 30;                      "<PATH_LAB>",                       "\"<onevent\"",
  int FORUA_LAB = 31;                     "<HTTP_LAB>",                       "\"</onevent>\"",
  int ACCEPT_LAB = 32;                    "<FORUA_LAB>",                      "\"<timer\"",
  int SCHEME_LAB = 33;                    "<ACCEPT_LAB>",                     "\"<p\"",
  int OPTIONAL_LAB = 34;                  "<SCHEME_LAB>",                     "\"</p>\"",
  int METHOD_LAB = 35;                    "<OPTIONAL_LAB>",                   "\"<a\"",
  int SENDREFERER_LAB = 36;               "<METHOD_LAB>",                     "\"</a>\"",
  int ONENTERFORWARD_LAB = 37;            "<SENDREFERER_LAB>",                "\"<anchor\"",
  int ONENTERBACKWARD_LAB =               "<ONENTERFORWARD_LAB>",             "\"</anchor>\"",
38;                                       "<ONENTERBACKWARD_LAB>",            "\"<fieldset\"",
  int ONTIMER_LAB = 39;                   "<ONTIMER_LAB>",                    "\"</fieldset>\"",
  int XML_LAB = 40;                       "<XML_LAB>",                        "\"<img\"",
  int NEWCONTEXT_LAB = 41;                "<NEWCONTEXT_LAB>",                 "\"localsrc=\"",
  int CONTENT_LAB = 42;                   "<CONTENT_LAB>",                    "\"vspace=\"",
  int ALIGN_ATR = 43;                     "<ALIGN_ATR>",                      "\"hspace=\"",
  int PALIGN_ATR = 44;                    "<PALIGN_ATR>",                     "\"height=\"",
  int TALIGN_ATR = 45;                    "<TALIGN_ATR>",                     "\"width=\"",
  int ONPICK_LAB = 46;                    "<ONPICK_LAB>",                     "\"<input\"",
  int ORDERER_LAB = 47;                   "<ORDERER_LAB>",                    "\"format=\"",
  int MODE_ATR = 48;                      "<MODE_ATR>",                       "\"emptyok=\"",
  int EM_TAG = 49;                        "<EM_TAG>",                         "\"maxlength=\"",
```

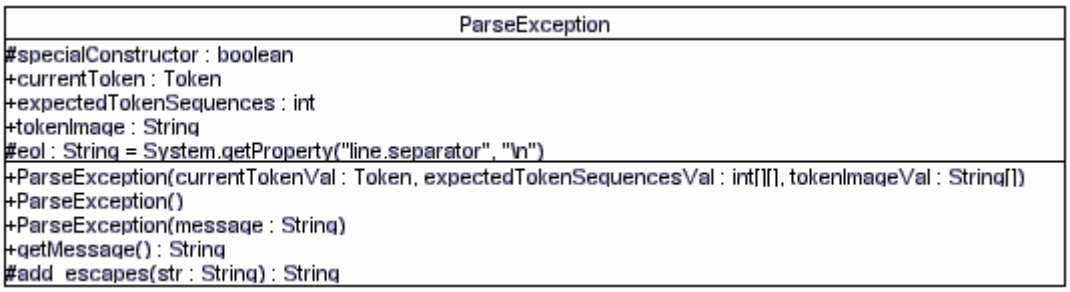
```
int STRONG_TAG = 50;                "<STRONG_TAG>",          "\"tabindex=\"",
int I_TAG = 51;                     "<I_TAG>",               "\"<select\"",
int B_TAG = 52;                     "<B_TAG>",               "\"multiple=\"",
int U_TAG = 53;                     "<U_TAG>",               "\"iname=\"",
int BIG_TAG = 54;                   "<BIG_TAG>",             "\"ivalue=\"",
int SMALL_TAG = 55;                 "<SMALL_TAG>",           "\"</select>\"",
int BR_TAG = 56;                    "\"<br/>\"",             "\"<table\"",
int ID_CLASS = 57;                  "<ID_CLASS>",            "\"align=\"",
int FILE = 58;                      "<FILE>",                "\"columns=\"",
int DOMAIN = 59;                    "<DOMAIN>",              "\"</table>\"",
int FILEPATH = 60;                  "<FILEPATH>",            "\"<tr\"",
int LINK = 61;                      "<LINK>",                "\"</tr>\"",
int FILE_HEADER = 62;               "<FILE_HEADER>",         "\"<td\"",
int COMENTARIO = 70;                                         "\"</td>\"",
int QUOTEDSTRA = 71;                                         "\"<option\"",
                                                             "\"</option>\"",
int DEFAULT = 0;                                             "\"<optgroup\"",
                                                             "\"</optgroup>\"",
                                                            }
```

WMLParserTokenManager

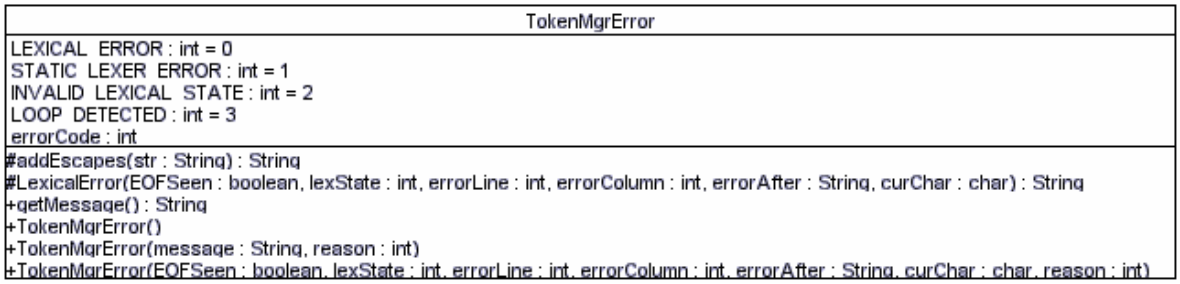
SimpleCharStream



ParseException



TokenMgrError



APÉNDICE B
Componentes de prueba

pruebatabla.wml

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="card1" title="Formato de Texto">
<p>
<table columns="3">
<tr>
<td>Formato</td><td>Tag</td><td>Muestra</td>
</tr>
<tr>
<td>Negrita</td><td>&lt; b &#62; </td><td><b>Texto en Negrita</b></td>
</tr>
<tr>
<td>Italica</td><td>&lt; i &#62; </td><td><i>Texto en Italica</i></td>
</tr>
<tr>
<td>Subrayado</td><td>&lt; u &gt; </td><td><u>Texto subrayadito</u></td>
</tr>
<tr>
<td>Enfatizado</td><td>&lt; em &gt; </td> <td><em>Texto enfatizado</em></td>
</tr>
<tr>
<td>Fuerte enfatizado</td><td>&lt; strong &gt; </td><td><strong>Enfatizado
fuerte</strong></td>
</tr>
<tr>
<td>Fuente grande</td><td>&lt; big &gt; </td><td><big>Texto grande</big></td>
</tr>
<tr>
<td>Fuente chica</td><td>&lt; small &gt;</td><td><small>Texto chico</small></td>
</tr>
</table>
</p>
</card>

</wml>
```

pfinal.wml

```
<!--pfinal.wml -->
<!-- archivo inicial -->
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card  title="Primer Carta" id="card1">
        <do type="accept" label="Siguiente">
                <go href="#card2"/>
        </do>
        <p align = "left">
                Por favor introduce tu nombre <br/>
                <input type="text" name = "nomvar"/>
        </p>
</card>
<card id="card2" title= "Segunda Carta">
        <onevent type="onenterforward">
                <refresh>
                        <setvar name="nombre" value="$nomvar"/>
                </refresh>
        </onevent>
        <do type="accept" label="Siguiente">
                <go href="#card3"/>
        </do>
        <p>
                Bienvenido <i>$(nombre)</i><br/>
                La tesis emplea las tecnologias <br/>
                <a href="#card3">uml</a><br/>
                <a href="C:\foo.wml">java</a> y
                <a href="C:\foo.wml">javacc</a><br/>
        </p>
</card>
<card id="card3" title="Tercera Carta">
        <do type="accept" label="Siguiente">
                <go href="C:\foo.wml"/>
        </do>
        <p>
                Con <b>UML</b> desarrollamos la <br/>
                solucion <br/>Con <a href="C:\foo.wml">java ...</a>
        </p>
</card>
</wml>
```

foo.wml

```
<!--foo.wml -->
<!-- archivo final -->
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card  title="Primer Carta" id="card1">
        <do type="accept" label="Siguiente">
                <go href="#card2"/>
        </do>
        <p>
                Con <b>Java</b> Swing creamos el <br/>
                sistema<br/>Con <a href="#card2">JavaCC ...</a>
        </p>
</card>
<card id="card2" title= "Segunda Carta">
        <do type="accept" label="Siguiente">
                <go href="#card3"/>
        </do>
        <p>
                Con <b>JavaCC</b> creamos el <br/>
                parser y el analizador lexico
        </p>
</card>
<card id="card3" title="Tercera Carta">
        <p>
                <big>Gracias</big> por su atencion<br/>
                Regresar a la pagina <a href="C:\pfinal.wml">principal</a>
        </p>
</card>
</wml>
```

pruebavar.wml

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">

<wml>
<card id="card1" title="Primera Carta">
<onevent type="onenterforward">
<refresh>
        <setvar name="card1_var1" value="val_1"/>
</refresh>
</onevent>
<p>
        Carta 1 ... <br/>
        <!-- Las siguientes variables no estarán definidas
            hasta que se haya accesado a todas las cartas -->
        carta1 variable1 $(card1_var1) <br/>
        carta2 variable2 $(card2_var1) <br/>
        carta3 variable3 $(card3_var1) <br/>
        <do type="accept" label="Siguiente Carta">
                <go href="#card2"/>
        </do>
</p>
</card>
<card id="card2" title="Segunda Carta">
<onevent type="onenterbackward">
<refresh>
        <setvar name="card2_var1" value="val_2"/>
</refresh>
</onevent>
<p>
        Carta 2 ... <br/>
        carta1 variable1 $(card1_var1) <br/>
        carta2 variable2 $(card2_var1) <br/>
        carta3 variable3 $(card3_var1) <br/>
        <do type="accept" label="Tercera Carta">
                <go href="#card3"/>
        </do>
</p>
</card>
<card id="card3" title="Tercera Carta">
<onevent type="onenterforward">
<refresh>
        <setvar name="card3_var1" value="val_3"/>
</refresh>
</onevent>
```

```
<p>
        Carta 3 ... <br/>
        carta1 variable1 $(card1_var1) <br/>
        carta2 variable2 $(card2_var1) <br/>
        carta3 variable3 $(card3_var1) <br/>
        <do type="prev" label="Previo">
                <prev/>
        </do>
</p>
</card>
</wml>
```

Bibliografía y Sitios de Interés

CAPÍTULO I.

[Oberliesen, 1982] R. Oberliesen. *Information, Daten und Signale.* Rowohlt Taschenbuch Verlag Gmbh, 1982

[Bunch, 1993] B. Bunch y A. Hellemans. *The Timetables of Technology*. Simon & Schuster, 1993

[Oberliesen, 1982] R. Oberliesen. *Information, Daten und Signale.* Rowohlt Taschenbuch Verlag Gmbh, 1982

[Kuhlmann, 1996] Kuhlmann Federico y Alonso Antonio. *Información y Telecomunicaciones*. Fondo de Cultura Económica, 1996

[Shannon, 1949] C. E. Shannon y W. Weaver. *A Mathematical Theory of Communication*. University of Illinois Press, USA, 1949

[WMLClub] http://www.wmlclub.com

[WAP] http://www.wapforum.org

CAPÍTULO II.

[WAPARCH] "Wireless Application Protocol Architecture Specification"

[WAPWAEO] "Wireless Application Environment Overview"

[WAPWAES] "Wireless Application Environment Specification"

[WAPWDP] "Wireless Datagram Protocol Specification"

[WAPWML] "Wireless Markup Language Specification"

[WAPWSP] "Wireless Session Protocol Specification"

[WAPWTA] "Wireless Telephony Application Specification"

[WAPWTAI] "Wireless Telephony Application Interface Specification"

[WAPWTLS] "Wireless Transport Layer Security Specification"

[WAPWTP] "Wireless Transaction Protocol Specification"

[WAP] http://www.wapforum.org

CAPÍTULO III.

[WMLClub] http://www.wmlclub.com

[WMLReference] http://www.forum.nokia.com

[WMLSpec] Wireless Markup Languaje Specification
http://www.wapforum.org

CAPÍTULO IV.

[Holt *et al.*, 1987] Robert W. Holt, Deborah A. Boehm-Davis and Alan C. Schultz. "Mental Representations of Programs for Students and Professional Programmers" en *Empirical Studies of Programmers: Second Workshop,* Ablex Publishing, Norwood, New Jersey,1987

[Letovsky, 1986] Stanley Letovsky. "Cognitive Processes in Program Comprehension," en *Empirical Studies of Programmers,* Ablex Publishing, Norwood, New Jersey,1986

[Notkin 1985] David Notkin. “The GANDALF Project” en *Journal of Systems and Software,* 1985

[Reps *et al*, 1989] Thomas Reps and Tim Teitelbaum, *The Synthesizer Generator Reference Manual,* Springer Verlag, Berlin, 1989

[Van De Vanter *et al,* 1992] Michael L. Van De Vanter,Susan L. Graham and Robert A. Ballance, “Coherent User Interfaces for Language-Based Editing Systems,” en *International Journal of Man-Machine Studies,*1992

[Van De Vanter *et al,* 1995] Michael L. Van De Vanter, “Practical Languaje-Based Editing for Software Engineers”, en *Software Engineering and Human-Computer Interaction,* Spinger Verlag, Berlin,1995

[JAVACC] JavaCC – The Java Parser Generator: A product of Sun Microsystems http://javacc.dev.java.net

CAPÍTULO V.

[Blair et al, 1991] G. Blair, J. Gallagher, D. Hutchison, and D. Sheperd. *Object Oriented Languages Systems and Applications*. Halsted Press, New York, New York, 1991

[Booch, 1991] G. Booch. *Object-Oriented Design UIT Applications*. Benjamín/Cummins, Menlo Park, California,1991

[Booch, 1996] Grady Booch. *Análisis y diseño orientado a objetos con aplicaciones*. 2a edición, Addison Wesley Longman, México, 1996

[Booch, 1999] Grady Booch. James Rumbaugh, Ivar Jacobson. *El Lenguaje Unificado de Modelado*. Addison Wesley Iberoamericana, Madrid, 1999

[Budd,1986] T. Budd. *An Introduction to Object-Oriented Programming*. Addison-Wesley, Reading, Massachusetts, 1986

[ENSELING] Oliver Enseling, Build Your own languages with JavaCC

[IEEE,1983] IEEE, IEEE Standard Glossary of Software Engineering Terminology, The Institute of Electrical and Electronic Engineers, New York, New York, 1983

[Jacobson *et al*, 2000] Ivar Jacobson, James Rumbaugh, Grady Booch. *El proceso unificado de desarrollo de software*. Addison Wesley Iberoamericana, Madrid, 2000

[Rumbaugh *et al*, 2000] James Rumbaugh, Grady Booch, Ivar Jacobson. *El Lenguaje Unificado de Modeloado.Manual de Referencia*. Addison Wesley Iberoamericana, Madrid, 2000

[Wirfs-Brock *et al*,1990] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing Object-Oriented Software.* Prentice Hall, Englewood Cliffs, New Jersey,1990

[TOA] http://www.toa.com/pub/abstraction.html

CAPÍTULO VI.

[Booch, 1999] Grady Booch, James Rumbaugh, Ivar Jacobson. *El Lenguaje Unificado de Modeloado*. Addison Wesley Iberoamericana, Madrid, 1999

[Fowler, 1999] Martín Fowler. *UML gota a gota*. Addison Wesley Longman de Mexico, S.A. de C.V., Mexico, 1999

[Sinan, 1998] Sinan Si Alhir. *UML in a Nutshell: A Desktop Quick Reference*. O'Reilly & Associates, USA, 1998.

[Sinan, 2003] Sinan Si Alhir. *Learning UML*. O'Reilly & Associates, USA, 2003

CAPÍTULO VII.

[García de Jalón *et al*, 2000] Javier García de Jalón, José Ignacio Rodríguez, Iñigo Mingo, Aitor Imaz, Alfonso Brazález, Alberto Larzabal, Jesús Calleja, Jon García. *Aprenda Java como si estuviera en primero*. Escuela Superior de Ingenieros Industriales de San Sebastián, Universidad de Navarra, 2000.

[Loy et al, 2002] Marc Loy, Robert Eckstein, Dave Wood, James Elliott. *Java Swing*. 2a edición, O'Reilly & Associates, USA, 2002

[Schildt, 2001] Schildt, Herbert. *Java 2 Manual de Referencia*. McGraw-Hill, España, 2001.

CAPÍTULO VIII.

[Jacobson *et al*, 2000] Ivar Jacobson, James Rumbaugh , Grady Booch. *El proceso unificado de desarrollo de software*. Addison Wesley Iberoamericana, Madrid, 2000