PLATAFORMA INTEGRADA DE DESARROLLO DE APLICACIONES PARA DISPOSITIVOS MÓVILES CON J2ME

Euler Adrián Trejo Narváez
Gabriel Alberto Vásquez Muños

Director: Ing. Esp. Oscar Mauricio Caicedo Rendón

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICAIONES
DEPARTAMENTO DE TELEMÁTICA
Popayán
Agosto de 2004

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

INTRODUCCIÓN

Teniendo en cuenta uno de los principios de lo que se ha denominado nueva economía o economía digital, el cual plantea que “todo lo que pueda ser móvil será móvil”, se ve la potencialidad que tienen los servicios y aplicaciones que se ejecuten en dispositivos móviles a través de redes de telecomunicaciones; desde este punto de vista se ha dado en el mundo un desarrollo acelerado en ese ámbito, pasando de las ya “obsoletas” comunicaciones analógicas que soportan bajas velocidades de transmisión de bits, a servicios de telecomunicaciones digitales que necesitan ancho de banda elevado. Las mejoras que continuamente se dan en los nuevos dispositivos y en las redes de telecomunicaciones hacen que día a día los usuarios requieran nuevas aplicaciones que tratando de aprovechar todas las capacidades existentes, con el pasar del tiempo y el avance de la tecnología se hacen cada vez más complejas. El desarrollo de estas aplicaciones requiere de las herramientas adecuadas, si se quiere ser competitivo en calidad y tiempo. Ya en otros tipos de aplicaciones como por ejemplo aplicaciones de escritorio o aplicaciones para Internet se ha tenido esto en cuenta y han aparecido entornos integrados de desarrollo de aplicaciones que aceleran bastante la implementación de las mismas; incluso en el área de los dispositivos móviles, muchos de estos entornos dan muy buen soporte, sin embargo, dadas sus características (están diseñados para soportar el desarrollo de aplicaciones no solo para dispositivos móviles sino también otros tipos de aplicaciones) requieren que su ejecución se realice en máquinas de muy buenas capacidades, tanto de almacenamiento como de procesamiento. Además de esto, el hecho de que no sean entornos de desarrollo especializados implica que si no se tiene la suficiente experiencia en ellos, puede resultar un poco compleja la configuración de alguna o varias de sus características para una aplicación particular. Estas consideraciones llevaron a pensar en la necesidad de tener una plataforma especializada en el desarrollo de aplicaciones para dispositivos móviles, que además hiciera uso de una de las tecnologías que industrialmente se está imponiendo en los últimos años, como los es JAVA y específicamente en su versión estándar (J2SE) y su versión para dispositivos móviles o Micro Edición (J2ME); es así

como surge PROMETEO, una plataforma integrada de desarrollo de aplicaciones para dispositivos móviles con J2ME.

Haciendo uso de los paradigmas de la programación orientada a objetos y el desarrollo de software basado en componentes, se logró la construcción de una plataforma útil y eficiente para el desarrollo aplicaciones J2ME, sin embargo, más allá de la construcción de una aplicación funcional, el trabajo realizado busca la definición de una arquitectura en la que se puedan enmarcar este tipo de plataformas, así como la caracterización, utilización y construcción de los componentes necesarios requeridos para llevar cabo su implementación y de cierta manera, la evaluación de las tecnologías empleadas en su desarrollo. De esta forma, teniendo como referencia metodológica el "Modelo para construcción de soluciones"[26], el cual se basa en un desarrollo iterativo e incremental con el fin de hacer más manejables los proyectos, se logró desarrollar la primera versión de una plataforma que brinda soporte a muchas de las necesidades de los programadores de aplicaciones para dispositivos móviles, que si bien no cuenta con todas las funcionalidades que se pudieran desear, posee características propias que la hacen muy útil a la hora de desarrollar.

Esta primera versión de PROMETEO, aunque cumple con el alcance de todos los objetivos planteados al comienzo del proyecto, presenta también ciertas limitaciones que dan pie a pensar en trabajos futuros que puedan llevar a mejorarla y mantenerla a medida que aumentan o que se hacen más complejas las necesidades de los desarrolladores y las aplicaciones requeridas.

El presente documento es una recopilación de los aspectos más importantes que cubrieron todo el desarrollo de Prometeo y está organizado de la siguiente forma:

Capitulo 1: Desarrollo de Software Basado en Componentes, contiene de manera general los aspectos relacionados con esta temática, haciendo especial énfasis en el modelo de Java Beans, sobre el cual se fundamentó el desarrollo basado en componentes.

Capitulo 2: Tecnología J2ME y el MIDP, contiene las características fundamentales de la plataforma J2ME y trata de manera especial los perfiles MID1.0 y MID2.0 haciendo una comparación de éstos a nivel de los paquetes que contienen.

Capitulo 3: PROMETEO, Plataforma Integrada de Desarrollo de Aplicaciones para Dispositivos Móviles con J2ME, contiene los aspectos relacionados con el desarrollo de la plataforma, cubriendo las etapas de análisis, diseño, implementación y validación, así como una descripción de la arquitectura funcional de Prometeo y las tecnologías que se emplearon en su construcción. Además, contiene una lista de las actividades que se consideraron importantes en todo el proceso de desarrollo.

Capitulo 4: Conclusiones, Recomendaciones y Trabajos futuros, contiene los aportes que cada integrante del grupo hace con relación a todo el trabajo realizado teniendo en cuenta el proceso de desarrollo, las tecnologías empleadas, los resultados obtenidos, etc.

Además, los siguientes anexos pretenden profundizar en algunos de los temas relacionados con cada capitulo o dar soporte a algunos de los planteamientos hechos a través de los mismos:

Anexo A: Comparación Entre MIDP1.0 y MIDP2.0, la comparación realizada en este anexo va a un nivel de profundidad mayor a la comparación hecha en el capitulo 2, llegando a comparar los atributos y métodos que fueron modificados o adicionados de un perfil a otro.

Anexo B: Diseño Detallado, contiene la descripción detallada de cada caso de uso implementado, así como su diagrama de secuencias y de clases.

Anexo C: Inconsistencias de Portabilidad, contiene ejemplos y descripciones de ciertos fragmentos de código de lenguaje JAVA cuyo comportamiento se presentaba diferente al ser ejecutados en distintas plataformas, particularmente WINDOWS y LINUX, afectando la portabilidad de las aplicaciones.

Anexo D: Pruebas de Integración, contiene las respuestas de la plataforma a diferentes pruebas realizadas, incluyendo por cada prueba el resultado esperado, el resultado obtenido y la respectiva discusión en donde haya lugar.

Anexo E: Código Fuente Documentado

Anexo F: Guía de Instalación

Anexo G: Ejecutables

Anexo H: Código Fuente Aplicación de Validación

Anexo I: Manual de Usuario, remitirse a la ayuda de PROMETEO.

I. DESARROLLO DE SOFTWARE BASADO EN COMPONENTES

1.1. GENERALIDADES

El desarrollo de software basado en componentes (CBD por sus siglas en inglés) es una de las áreas que ha recibido considerable atención como una de las estrategias que favorece el desarrollo de aplicaciones abiertas y distribuidas con el objetivo de soportar la gestión de las operaciones globales y dinámicas del negocio, bajo una premisa fundamental: la reutilización. "Reutilización de software es el proceso de crear sistemas de software a partir de software existente, en lugar de desarrollarlo desde el comienzo" [1].

Varios estudios han demostrado la efectividad de la reutilización de software[1]:

- 40-60% del código fuente es reutilizable de una aplicación a otra.
- Aproximadamente el 60% del diseño y del código de aplicaciones administrativas es reutilizable.
- Aproximadamente el 75% de las funciones son comunes a más de un programa.
- Sólo el 15% del código encontrado en muchos sistemas es único y novedoso a una aplicación específica.
- El rango general de reuso potencial está entre el 15% y el 85%.

Cabe destacar que la reutilización de software va más allá de la reutilización de componentes de software, ésta involucra uso de otros elementos denominados activos de software, tales como: algoritmos, diseños, arquitecturas de software, planes, documentación y especificaciones de requerimientos.

A pesar que se habla de desarrollo de componentes de software desde hace más de 30 años (como solución a la crisis del software), el desarrollo de software basado en componentes es relativamente nuevo y actualmente ha sido impulsado gracias a las

mejoras tecnológicas en cuanto a hardware e infraestructura de redes lo cual ha permitido el establecimiento de plataformas propicias para el desarrollo de aplicaciones de este tipo. Modelos de componentes como COM (Component Object Model), DCOM (Distributed COM) de Microsoft, JavaBeans de Sun Microsystem, SOM (System Object Model) de IBM, CORBA (Common Object Request Broker Architecture) del consorcio OMG (Object Management Group), luchan día a día por convertirse en estándares del mercado.

Un desarrollo de software basado en componentes debe integrar y equilibrar apropiadamente consideraciones tanto tecnológicas como metodológicas. Actualmente el enfoque de desarrollo basado en componentes se concentra en definir aspectos tecnológicos de infraestructura, restándole importancia a la parte metodológica que permite el establecimiento de métodos y técnicas adecuadas para el desarrollo de componentes software reutilizables, adaptables y fácilmente extensibles.

Desde el punto de vista del desarrollo de software basado en componentes, la ingeniería de software se percibe como un proceso de búsqueda, selección, adaptación y composición de componentes software almacenados en una biblioteca. Cuando se trata de componentes con una funcionalidad relevante en el entorno del problema, el análisis debe ser más riguroso, los aspectos de especificación, adaptación y evolución se tornan tan complejos que deben ser enfrentados desde el punto de vista metodológico y de manera independiente de la plataforma tecnológica en que éstos componentes van a ser implementados.

1.2. CARACTERÍSTICAS DEL SOFTWARE BASADO EN COMPONENTES

Las principales características del software basado en componentes se pueden resumir en las siguientes:

- **Distribución**: El software puede estar distribuido de tal manera que los componentes residan en diferentes equipos pertenecientes a una misma red.
- **Heterogeneidad**: El software se ejecuta en diferentes plataformas, sobre distintos sistemas operativos, además los componentes pueden estar desarrollados en

diferentes lenguajes de programación.

- **Independencia de la extensibilidad**: Es fácilmente modificable y ampliable, añadiendo nuevos componentes.
- **Dinamismo**: Es altamente dinámico, puede evolucionar fácilmente ya sea por ampliación, desaparición, sustitución de componentes o reconfiguración de las relaciones entre ellos.

1.3. INGENIERÍA DE DOMINIO

La reutilización de software dentro de un dominio de aplicación pasa por el descubrimiento de elementos comunes a los sistemas pertenecientes a dicho dominio. Cuando se utiliza este enfoque se está produciendo un cambio de un desarrollo orientado a un único producto software a un desarrollo centrado en varios productos que comparten unas características formando una familia. Esto provoca una reestructuración del proceso software de forma que surgen dos procesos distintos: la *ingeniería de dominio* y la *ingeniería de aplicación*.

La ingeniería de dominio se centra en el desarrollo de elementos reutilizables que formarán la familia de productos, mientras que la ingeniería de aplicación se orienta hacia la construcción o desarrollo de productos individuales, pertenecientes a la familia de productos, y que satisfacen un conjunto de requisitos y restricciones expresados por un usuario específico, reutilizando, adaptando e integrando los elementos reutilizables existentes y producidos en la ingeniería de dominio.

La ingeniería de dominio se puede definir como el proceso clave que se necesita para el diseño sistemático de una arquitectura y de un conjunto de elementos software reutilizables que pueden ser usados en la construcción de una familia de aplicaciones relacionadas o subsistemas. Es el proceso sistemático que incorpora criterios de negocio y produce un soporte racional, modelos y arquitecturas que permiten tomar mejores decisiones, llevar un registro del dominio, obtener nuevas versiones y mejorar el proceso de desarrollo gracias al conocimiento que se tiene del sistema [2], [3].

El objetivo fundamental de la ingeniería de dominio es la optimización del proceso de desarrollo del software en un espectro de múltiples aplicaciones que representan un negocio común o problema de dominio [4].

El propósito de la ingeniería de dominio es el desarrollo de elementos software dentro de un dominio para que puedan ser usados y obviamente reutilizados en la construcción de aplicaciones para un dominio concreto.

Así, la ingeniería de dominio ofrece un conjunto de modelos de referencia que describen el dominio, identificando las arquitecturas y los componentes que permitirán el desarrollo de aplicaciones dentro de él. Además, ofrece la base de conocimiento adecuada para comprender el espacio del problema definido por el software presente en el dominio. Es, por tanto, una parte clave en la consecución de una arquitectura y un conjunto de elementos reutilizables que reúnan la calidad suficiente para confiar en ellos.

La ingeniería de dominio se originó alrededor de la década de los ochenta y desde entonces se han publicado varias aproximaciones que incluyen un amplio rango de métodos y procesos, tanto formales como informales, para llevar a cabo las diferentes actividades en que se descompone la ingeniería de dominio, para capturar y representar información sobre sistemas que comparten un conjunto común de características y datos.

Entre las diversas propuestas existentes en la ingeniería de dominios cabe destacar como las más representativas a FODA (*Feature-Oriented Domain Analysis*), OODA (*Object Oriented Domain Analysis*), ODM (*Organization Domain Modeling*), DAGAR (*Domain Architecture-based Generation for Ada Reuse*), FORM (*Feature-Oriented Reuse Method*), FeatureRSEB (*Feature Reuse- Driven Software Engineering Business*) y FODAcom.

La diferencia entre estas propuestas reside en la manera en la que identifican el dominio de forma efectiva y hacen uso de la máxima experiencia sobre él, la arquitectura y los sistemas existentes. Algunos métodos se centran en cómo conseguir ejemplares existentes para hacer un análisis detallado de los mismos, mientras que otros están orientados a cómo encontrar, representar y agrupar conjuntos de características.

Diferentes aplicaciones en la misma familia de aplicaciones o dominio de problema se comparan o se distinguen gracias a sus *características*. Cuando se desarrolla una arquitectura y un conjunto de componentes para su reutilización es importante comprender cuáles de ellas se incluyen como parte de los elementos reutilizables, cómo se relacionan y cuáles son obligatorias u opcionales para las diferentes aplicaciones.

Una característica es una abstracción funcional distintiva e identificable que puede ser empaquetada, implementada, probada, distribuida y mantenida. Son, por tanto, objetos de primera clase en el desarrollo de software orientado a un dominio [5]. Existen diferentes tipos de características que son tenidas en cuenta dependiendo del interés que se tenga puesto en el desarrollo del sistema. Tanto los usuarios, como los analistas y los desarrolladores están involucrados en el desarrollo pero con diferentes perspectivas.

Los usuarios están más preocupados por los servicios o funciones que debe ofrecer el sistema; los analistas y diseñadores se centran en las tecnologías del dominio; y los desarrolladores se interesan especialmente por las técnicas de implementación. Las aplicaciones no podrán llevarse a cabo sin un consenso de todas las partes que permita conseguir un conjunto de características consistente.

La diversidad en las propuestas trae consigo una diversidad en la terminología, existiendo una falta de consenso en la misma. Normalmente, por ingeniería de dominio se hace referencia a todo el proceso de creación de los elementos reutilizables propios del dominio. En lo que ya no existe uniformidad es en el nombre de las fases que componen el proceso. Típicamente existen dos fases muy diferenciadas: una primera de análisis del dominio, donde debe adquirirse el conocimiento sobre el dominio (búsqueda de elementos comunes y diferencias en la familia de sistemas que conforman el dominio) y modelarse, y una segunda fase de ingeniería de componentes de dominio, en la que los elementos software reutilizables propios del dominio son creados. [6].

1.4. PROCESO DE REUTILIZACION DE SOFTWARE

Dentro del proceso de desarrollo de software reutilizable se hace necesario distinguir dos conceptos fundamentales, el desarrollo de componentes software reutilizables y el desarrollo de software basado en componentes reutilizables.

1.4.1 Desarrollo de componentes software reutilizables

Es la adaptación o desarrollo de componentes software con el propósito expreso de ser reutilizados en futuras aplicaciones. Su objetivo es producir repositorios de activos o componentes de software reutilizable que puedan ser utilizados en el desarrollo de software**.**

El desarrollo de componentes se lleva a cabo a través de la adaptación de componentes existentes o la creación de repositorios aplicando los procesos de la Ingeniería de Dominio.

En el proceso de desarrollo de componentes software, es necesario tener en cuenta que un componente debe ser genérico, es por esto que se deben generalizar ciertos aspectos básicos como:

- El nombre del componente, debe referirse a entidades genéricas del dominio y no a aspectos específicos de una aplicación.
- Las operaciones del componente, no deben pertenecer a una aplicación determinada, de tal manera que la funcionalidad del componente pueda ser extendida fácilmente agregando nuevas operaciones, por esto las operaciones específicas deben ser removidas del componente.
- Las excepciones manejadas por el componente, no se debe tener en cuenta excepciones específicas de una aplicación. Los mecanismos de manejo de excepciones se agregan al componente sólo para incrementar su robustez.
- La Certificación del componente, el componente debe estar certificado como un componente reutilizable con el fin de asegurar su calidad y confiabilidad.

1.4.2. Desarrollo de software basado en componentes reutilizables

Se refiere al desarrollo de una nueva aplicación, la cual se construye a partir de un conjunto de componentes ya existentes. Es un enfoque de desarrollo de software que maximiza la reutilización de componentes software y reduce el número de componentes que requieren ser desarrollados desde cero.

Como condiciones mínimas para el desarrollo de software a partir de la reutilización de componentes, se tiene que deben existir previamente repositorios o bases de componentes reutilizables, que estos componentes sean confiables y actúen de acuerdo a sus especificaciones y que los componentes estén debidamente documentados.

1.5. COMPONENTES SOFTWARE REUTILIZABLES

Con respecto al concepto de componente software se tienen muchas definiciones de autores respetables, pero para evitar confusiones se tomará la definición dada por el CBDi Forum: "*Un componente es una pieza de software que describe y/o libera un conjunto de servicios que son usados sólo a través de interfaces bien definidas*"

1.5.1. Características de un Componente Software Reutilizable

- Es identificable. Debe tener una identificación clara y consistente que facilite su catalogación y acceso.
- Es auto contenido. Un componente no debe requerir, en lo posible, la reutilización de otros componentes para cumplir su función, si el componente requiere de otros, el conjunto de componentes o funciones, visto como un todo, debe ser considerado como un componente de más alto nivel.
- Es rastreable. Debe retener su identidad y ser rastreable durante el ciclo de desarrollo
- Es reemplazable. Puede ser fácilmente reemplazado por una nueva versión o por otro componente que proporcione los mismos servicios.
- Es accesible sólo a través de su interfaz. Es accedido a través de una interfaz claramente definida, la interfaz debe ser independiente de la implementación física y

debe ocultar los detalles de su diseño interno.

- Sus servicios son inmutables. Los servicios que ofrece un componente a través de su interfaz no deben variar, la implementación física de estos servicios pueden ser modificadas, pero no deben afectar a la interfaz.
- No posee un estado persistente. A excepción de los atributos no funcionales como el número de serie, un componente no se distingue de otras copias de sí mismo, por lo tanto en un proceso se puede decir si hay o no un componente pero no varias instancias del mismo.
- Es documentado. Debe contar con una documentación clara que facilite la recuperación del componente desde el repositorio, la evaluación del componente, su aplicación y adaptación al nuevo ambiente y su integración con otros componentes del sistema en que se reutiliza.
- Es mantenido. Debe ser mantenido para facilitar un reuso sistemático, su estado de reuso debe contener información acerca de quien es el propietario del componente, quien lo mantiene, a quien acudir cuando surjan problemas y cual es el estado de la calidad del componente.
- Es genérico. No debe implementarse con características particulares de una aplicación, debe contener los aspectos generales de un conjunto de aplicaciones.
- Su implementación física está oculta. Sólo se tiene acceso a su interfaz, pero no se tiene acceso a los detalles internos de su implementación.
- Es independiente de otros componentes. Debe ser autocontenido y no depender de otros componentes.
- Es encapsulado. Presenta alta cohesión manteniendo su implementación interna oculta.
- Es independiente del lenguaje o a las herramientas usadas en su desarrollo. Debe prestar servicios concretos, sin importar el lenguaje de programación en que fue implementado o las herramientas utilizadas para su desarrollo.
- Es independiente de la plataforma de ejecución.
- Puede ser reutilizado dinámicamente.
- Es certificado. Ha sido previamente verificado y cumple todas las características de un componente.

1.5.2. Interfaz y Contratos de un Componente

Una interfaz determina la manera en que un componente se reutiliza y como puede conectarse a otros componentes. La interfaz permite a los clientes acceder a los servicios proporcionados por el componente. La interfaz define una operación o un conjunto de operaciones llamadas servicios o responsabilidades. Por todas estas razones las interfaces son un aspecto fundamental en la composición de componentes.

Un componente puede tener varias interfaces, una para cada punto de acceso: uso, administración, configuración y demás, pero no es aconsejable tener interfaces redundantes.

Por lo general un componente produce (exporta) un resultado que es consumido (importado) por otro componente.

Un contrato específica las relaciones recíprocas entre un componente y un cliente, que puede ser otro componente, estas relaciones están dadas por las interfaces que define cada componente, en este sentido, un contrato es una relación de co-dependencia entre componentes, por ejemplo, un cliente puede depender de un componente para proveer un servicio e igualmente, el componente puede depender del cliente para que provea los argumentos dentro de ciertas restricciones o para iniciar apropiadamente el servicio del componente.

Un contrato puede estar definido en dos sentidos, uno relacionado directamente con el componente y el otro relacionado con las interacciones. Para el primer caso se tiene un contrato de componente, en el cual se especifica un patrón de interacción propio de ese componente, los servicios que provee el componente y las obligaciones de los clientes y el ambiente necesario para que el componente pueda proveer esos servicios. Para el segundo caso se tiene un contrato de interacción el cual especifica un patrón de interacción entre diferentes roles y las obligaciones reciprocas entre componentes que satisfacen esos roles.

Dado que los componentes pueden ser implementados en distintos lenguajes de

programación, la naturaleza de la interfaz varia dependiendo del lenguaje empleado para su implementación, esto es, para lenguajes OO (Orientados a Objetos) las clases se diseñan de tal manera que la interfaz sea visible y separada de la implementación de las operaciones (métodos); mientras que en los lenguajes procedimentales la interfaz se define a través de la declaración de la función o procedimiento o el uso de variables globales.

Existen varios tipos de interfaces:

- Interfaces de datos.
- Interfaces gráficas.
- Interfaces de programación.

Interfaces de Datos
Empleadas en componentes E-P-S, estos leen datos de Entrada, ejecutan un Proceso de manipulación de datos o cálculo y escriben datos de Salida.

Entre los formatos más comunes que se manipulan en este tipo de interfaces tenemos el formato ASCII y el formato de las Bases de datos relaciónales.

Interfaces de Programación
El mecanismo empleado en este tipo de interfaces es el API (Application Programming Interface), mediante el cual se establece la interacción entre dos aplicaciones con plataformas iguales o diferentes.

Un API es "un conjunto de subrutinas de interfaz o llamadas a librerías que definen los métodos empleados para que un programa pueda invocar servicios externos".

Un ejemplo claro de este tipo de interfaces son las empleadas en la librería de clases del sistema en Java, la cual provee diferentes paquetes (java.lang, java.util, java.io) que contienen clases de uso común. Cada paquete contiene una interfaz y una colección de clases.

1.6. COMPOSICIÓN SOFTWARE

Es el término utilizado en el CBD para referirse a cómo los sistemas software se ensamblan. La composición de software se puede entender como "el proceso de construir aplicaciones mediante la interconexión de componentes de software a través de sus interfaces de composición". Se trata de aplicar la filosofía plug and play, asemejando la construcción de aplicaciones software a la construcción de sistemas hardware, en donde los componentes pueden ser reemplazados con otros de igual funcionalidad y con ciertas características estandarizadas; haciendo con esto que el mantenimiento y evolución del sistema sea relativamente fácil. Puesto que uno de los objetivos de la programación orientada a componentes es crear un mercado global de componentes; se persigue con esto, que el desarrollo de aplicaciones se realice de manera mucha más rápida y eficiente aprovechando al máximo los componentes ya desarrollados por otras personas.

1.6.1. Contenedores

Los contenedores son entidades software que permiten contener a otras entidades proporcionando un entorno compartido de interacción. Normalmente objetos y componentes visuales que a su vez pueden contener otros componentes visuales. La relación entre los objetos y/o componentes se establece mediante eventos. [7].

1.6.2. Entornos de Desarrollo Integrado

Un entorno de Desarrollo Integrado (IDE) es una aplicación (visual) que permite la construcción de aplicaciones a través de componentes. Incluye editores, Browsers, ayudas, directorios de componentes, editores de propiedades, personalizadores, compiladores, depuradores, herramientas de control de configuración, etc.

1.6.3. Framework de componentes

Los Frameworks son la piedra angular de la moderna ingeniería del software. El desarrollo de Frameworks está ganando aceptación rápidamente, debido a su capacidad para

promover la reutilización del código del diseño y del código fuente. Los frameworks son los Generadores de Aplicación que se relacionan directamente con un dominio específico, es decir, con una familia de problemas relacionados. [8].

Una analogía ampliamente utilizada para comprender el concepto de Framework de componentes es verlo como un mini sistema operativo, siendo los componentes al Framework lo que los procesos al sistema operativo.

Como ejemplos de Frameworks se tiene la especificación de EJB que define un framework de servidores y contenedores que soportan el modelo de componentes de EJB, o el framework de Microsoft Visual Basic, especializado en la composición de componentes visuales, donde el framework es el intérprete para el lenguaje de script y la composición, todo ello acoplado con el modelo de componentes COM y los servicios de comunicación ofrecidos por el sistema operativo.

1.6.4. Plataforma de componentes

Una plataforma de componentes es un entorno de desarrollo y de ejecución de componentes que permiten aislar la mayor parte de las dificultades conceptuales y técnicas que conlleva la construcción de aplicaciones basadas en los componentes de un modelo de componentes concreto. [7].

Una plataforma de desarrollo, es un conjunto de componentes que pueden involucrar diferentes tecnologías en lo referente a herramientas y lenguajes de programación. Las plataformas de desarrollo facilitan la implementación de aplicaciones, teniendo como objetivo la disminución del tiempo de desarrollo mediante el ocultamiento de ciertos detalles de implementación a los desarrolladores, ofreciendo una funcionalidad completa dentro de un ambiente integrado, tanto de desarrollo como de ejecución.

La plataforma gestiona los recursos que comparten los componentes y ofrece los mecanismos subyacentes que permiten la comunicación e interacción entre los distintos componentes que la conforman.

En el desarrollo de una plataforma de componentes se debe tener en cuenta, no sólo el domino de la plataforma como aplicación sino el conjunto de dominios sobre los cuales va a operar la plataforma. Los componentes deben ser lo suficientemente genéricos para que puedan responder a las necesidades y requerimientos funcionales, de rendimiento, seguridad y calidad de los dominios de las distintas aplicaciones soportadas por la plataforma de desarrollo.

Los frameworks y las plataformas de componentes poseen muchas características similares e igualmente comparten muchas propiedades, pero las plataformas se desenvuelven en un entorno de trabajo más genérico y en cierta forma son más estáticas, mientras que los frameworks son orientados a una lógica de negocio más específica y poseen un mayor dinamismo.

1.7. BENEFICIOS Y RIESGOS

Para aprovechar al máximo todos los beneficios de los componentes software, éstos deberían formar parte de un framework o de una plataforma de desarrollo, para que mediante ellos, se pueda explotar toda su potencialidad de reuso.

El desarrollo de componentes, al igual que muchas otras actividades del desarrollo de software, representa una inversión que debe ser cuidadosamente evaluada. Los beneficios pueden observarse en la medida en que el framework o la plataforma de desarrollo de componentes faciliten llenar (por lo menos parcialmente) las necesidades de un conjunto de diversas aplicaciones, es decir el conjunto de dominios de las diferentes aplicaciones que encajen en la plataforma y además que permitan particularizar una aplicación dada efectuando los cambios necesarios. Si estos requerimientos se satisfacen en suficiente grado, se puede considerar aceptable la inversión en el desarrollo de una plataforma de componentes.

De hecho, el uso de plataformas de desarrollo presenta muchas ventajas y por ende es muy conveniente utilizarlas en el desarrollo de software basado en componentes. Es muy común que las aplicaciones con cierto grado de complejidad, durante el tiempo de

desarrollo experimenten cambios en los requerimientos que pueden ser fácilmente resueltos sí se trabaja con una plataforma de desarrollo, y para las aplicaciones de menor complejidad que por lo general son desarrolladas con restricciones de tiempo, su desarrollo puede indudablemente llevarse a cabo más rápido con el uso de una plataforma de desarrollo.

A pesar que el desarrollo de software basado en componentes presenta grandes ventajas, también se cuenta con algunos riesgos, que deben ser considerados:

- Los desarrolladores deben invertir tiempo y esfuerzo en el aprendizaje del manejo de la plataforma de desarrollo antes de poder aprovechar sus beneficios.
- El desarrollo de nuevos frameworks y plataformas de desarrollo es una actividad costosa y a largo plazo. Los beneficios a largo plazo deben ser justificados en términos de oportunidades para recobrar la inversión.
- Por lo general, los proyectos pequeños tienen metas a corto plazo que presentan conflictos con las metas a largo plazo de la ingeniería de componentes. La gestión debe ser confiable para desarrollar una infraestructura orientada al servicio para soportar el uso de una plataforma de desarrollo en el proyecto. Si el uso de plataformas de desarrollo resulta ser demasiado costoso los proyectos simplemente no lo adoptan.
- El tratar de desarrollar rápidamente una plataforma de desarrollo ocasiona que se produzcan rediseños antes de que esta pueda ser totalmente estable. El costo de los ajustes que se deben realizar puede ser extremadamente alto, aunque a largo plazo los beneficios pueden ser muy significativos.

Es de notar que la ingeniería de componentes no solamente está relacionada con el desarrollo de componentes software, ésta involucra todos los aspectos del desarrollo de software, desde la captura de requerimientos y especificaciones hasta el diseño e implementación. Es por esto que los mayores beneficios del reuso no son los componentes en si mismos, sino el dominio del conocimiento y el desarrollo de diseños genéricos. El reuso del software es más ventajoso si se cuenta con un plan de desarrollo. Si se espera hasta después de los requerimientos sean especificados y el sistema sea diseñado, muchas oportunidades de reuso pueden perderse.

1.8. MODELOS DE COMPONENTES

Un Modelo de Componentes es una especificación en la que se define la estructura de los componentes, la comunicación entre componentes y la manipulación por parte de los ambientes de desarrolladores.

Entre los principales modelos de componentes se tiene:

- COM/DCOM
- CORBA
- JavaBeans.

1.8.1. Component Object Model (COM) / Distributed COM (DCOM)[1]

COM empezó a desarrollarse entre DEC y Microsoft. Posteriormente DEC abandonó el proyecto, quedando tan sólo en manos de Microsoft.

COM tiene sus raíces en OLE (Object Linking and Embedding) versión 1, creada en 1991 como marco de gestión e integración de documentos para la familia de programas Microsoft Office. Más adelante, la compañía se dio cuenta que la integración de documentos no es sino un caso especial de integración de componentes, y a partir de la versión 2 de OLE, presentada en 1995 y denominada desde entonces como COM, proporciona un mecanismo de propósito general para la integración de componentes en plataformas Windows.

Con el término COM, se conoce tanto a la especificación como la implementación de Microsoft que proporciona un marco para la integración de componentes. Este marco soporta la interoperabilidad y la reutilización de componentes distribuidos, con lo que los desarrolladores pueden construir sistemas a base de ensamblar componentes de diferentes proveedores, que se comunican unos con otros vía COM. Otros beneficios que se pretenden alcanzar con este esquema de construcción de sistemas son la adaptabilidad y la mantenibilidad.

[1] Puede encontrar mayor información en DCOM TECHNICAL OVERVIEW
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomtec.asp

COM define un interfaz de programación (API) para la creación de componentes que van a usarse para la construcción de aplicaciones a medida o para permitir que diversos componentes interactúen. Sin embargo, para que esta interacción sea posible, los componentes deben adoptar una estructura binaria especificada por Microsoft. Si lo hacen, aunque los componentes estén escritos en diferentes lenguajes, éstos pueden interoperar.

DCOM es una extensión de COM el cual soporta objetos distribuidos en una red LAN, WAN o Internet. DCOM también fue desarrollado por Microsoft y se ha sometido a la IETF (Internet Engineering Task Force) como estándar. Desde 1996, ha formado parte del sistema Operativo Windows NT 4.0, y está también disponible en Windows 95. Existen implementaciones DCOM en plataformas como UNIX, además de versiones para Solaris, Linux y HP/UX.

DCOM permite la interacción de componentes en una red. Mientras que los procesos COM pueden funcionar en la misma máquina aunque en diferentes espacios de direcciones, con DCOM estos procesos pueden estar dispersos en diferentes lugares de una red. Con DCOM es posible la interacción de componentes disponibles en diferentes plataformas, siempre que exista DCOM para esos entornos.

Resulta más conveniente considerar COM y DCOM como una sola tecnología que proporciona una serie de servicios para la interacción de componentes, desde la integración de objetos en una sola plataforma hasta la interactividad de componentes a través de redes heterogéneas. De hecho, las extensiones COM y DCOM están mezcladas en un solo 'runtime' que proporciona tanto acceso local como remoto.

COM y DCOM están especialmente maduras para los entornos propiedad de Microsoft. Existe un amplio mercado de componentes conformes con las especificaciones COM para Windows. Una iniciativa de importancia fuera del campo Windows es el desarrollo EntireX de la compañía Software AG, que extiende las posibilidades de DCOM a diversos sistemas Unix (Solaris, Digital Unix, HP-UX, AIX, Linux) así como al entorno de grandes ordenadores OS/390.

Uno de los puntos débiles de DCOM es la seguridad, demostrándose su vulnerabilidad con la reciente proliferación de gusanos a través de Internet que se aprovechan de sus falencias en este aspecto para atacar a los sistemas Microsoft Windows NT, 2000 y XP.

El paradigma de las aplicaciones distribuidas está moviéndose muy rápidamente, debido en parte a la relativa inmadurez de la tecnología y a los avances recientes de la informática basada en la Web. La industria que desarrolla esta informática ha empezado a alinearse en dos campos tecnológicos rivales, uno de ellos centrado alrededor de las tecnologías de Microsoft, COM/DCOM, Internet Explorer, y ActiveX, y el otro en soluciones de Netscape, CORBA y Java, sin que todavía pueda decirse que haya un claro vencedor. Afortunadamente ambos campos trabajan en diversos mecanismos para interconectar las dos bases tecnológicas.

1.8.2. Common Object Request Broker Architecture (CORBA)[2]

CORBA define la infraestructura para la arquitectura OMA (Object Management Architecture) de OMG (Object Management Group), especificando los estándares necesarios para la invocación de métodos sobre objetos en entornos heterogéneos.

Los entornos heterogéneos son aquellos en los que las arquitecturas que constituyen el entorno pueden ser sistemas Microsoft Windows, máquinas Unix de diferentes fabricantes (GNU/Linux entre otros) e inclusos sistemas como MacOS o OS/2. Y es más, dentro de la heterogeneidad también se incluyen los sistemas de comunicaciones (protocolos de comunicación como TCP/IP, IPX) o los lenguajes de programación utilizados en las diferentes arquitecturas.

CORBA es la respuesta a la necesidad de interoperabilidad entre la rápida proliferación de hardware y productos de software. Permite la comunicación entre aplicaciones sin importar donde estén localizadas o quien las diseñó o implementó. Es una arquitectura

[2] Puede encontrar mayor información en PI FIET 0682 - CORBA Y JAVA como soporte de aplicaciones y servicios de ingeniería distribuida. Martínez Ortega, José Fernán. Puede encontrar mayor información en PI FIET 0682 - CORBA Y JAVA como soporte de aplicaciones y servicios de ingeniería distribuida. Martínez Ortega, José Fernán

que implementa programas, o llamadas a objetos los comunica indiferentemente del lenguaje de programación en que se escribieron o en que sistema operativo corran.

CORBA define su propio modelo de objetos, basado en la definición de las interfaces de los objetos mediante el lenguaje IDL (Interface Definition Languaje). De esta forma se logra una abstracción de la heterogeneidad que permite que el uso de CORBA no sea nada complejo. CORBA sigue una metodología concreta y fácil de seguir.

CORBA ha logrado parte de su éxito gracias a la clara separación entre la interfaz de los objetos y la implementación de los mismos. Las interfaces se definen utilizando el lenguaje IDL, cuya principal característica es su alto nivel de abstracción, lo que le separa de cualquier entorno de desarrollo específico. Para la implementación de los objetos se puede utilizar cualquier lenguaje de programación que proporcione enlaces con el lenguaje IDL. Para que un lenguaje de programación se pueda utilizar desde CORBA, debe tener definida la forma de enlazarse con IDL.
De esta forma, y a partir de una especificación de las interfaces en IDL, se generan unos cabos (proxies) en el lenguaje elegido que permiten el acceso a la implementación de los objetos desde la arquitectura CORBA.

Un Object Request Broker (ORB) es el middleware que establece las relaciones entre objetos cliente-servidor. Un cliente que utiliza un ORB puede de manera transparente invocar un método de un objeto del servidor, el cual puede estar en la misma máquina o en otro equipo perteneciente a la red. El ORB provee la interoperabilidad entre aplicaciones de diferentes máquinas, en ambientes heterogéneos distribuidos e interconecta sistemas de múltiples objetos.

El cliente es la entidad que desea ejecutar una operación en un objeto del servidor. La implementación del objeto es el código y los datos que forman el objeto. Las interfaces pueden ser definidas estáticamente en el IDL como un stub IDL (el cual es dependiente del objeto).

Las interfaces del cliente son completamente independientes de la localización del objeto, del lenguaje de programación, o de cualquier otro aspecto que no se refleje en la

interface del objeto.

El ORB es el responsable de encontrar la implementación del objeto requerido. También prepara la implementación del objeto y recibe el requerimiento, enviando los datos para resolver la demanda. El cliente no tiene que estar informado de donde se localiza el objeto, de su lenguaje de programación, ni de su sistema operativo, o de cualquier otro aspecto del sistema que no sea parte de la interfaz del objeto.

Entre los servicios del ORB se incluyen: generación e interpretación de las referencias del objeto, invocación del método, activación de la implementación, desactivación y registro de implementaciones.

CORBA es un estándar creado con la idea de una distribución de los sistemas basada en objetos. Con CORBA se pretende definir una arquitectura que especifique cómo se crean los objetos y cómo se accede a sus funcionalidades.

1.8.3. Java Beans

Un JavaBean también conocido como Bean es un componente software que se puede reutilizar y que puede ser manipulado visualmente por una herramienta de programación en lenguaje Java. Para ello, se define una interfaz para el momento del diseño que permite a la herramienta de programación o IDE, interrogar al componente y conocer las propiedades que define y los tipos de eventos que puede generar en respuesta a diversas acciones.

En general, un Bean es una clase que obedece ciertas reglas:

- Un Bean debe tener un constructor por defecto (sin argumentos).
- Un Bean debe tener persistencia, es decir, implementar la interface Serializable.
- Un Bean debe tener introspección. Los IDE reconocen ciertas pautas de diseño, nombres de las funciones miembros o métodos y definiciones de las clases, que permiten a la herramienta de programación mirar dentro del Bean y conocer sus propiedades y su conducta.

Aunque los Beans individuales pueden variar ampliamente en funcionalidad desde los más simples a los más complejos, todos ellos comparten las siguientes características:

Introspección: permite analizar a la herramienta de programación o IDE como trabaja el Bean. Los Beans soportan la introspección de dos formas, la primera de ellas, adhiriéndose a las convenciones específicas de nombres conocidas como patrones de nombrado, cuando se nombran las características del Bean. La clase java.beans.Introspector examina el Bean buscando esos patrones de diseño para descubrir las características del Bean. La segunda forma de realizar la introspección es proporcionando explícitamente información sobre la propiedad, el método o el evento con una clase Bean Information relacionada. Esta clase implementa el interface BeanInfo. Una clase BeanInfo lista explícitamente aquellas características del Bean que están expuestas a la herramienta de desarrollo.

Personalización: la apariencia y el comportamiento de un Bean pueden ser personalizados en el momento del diseño dentro de las herramientas de desarrollo compatibles con los Beans. Un Bean se puede personalizar de dos formas, utilizando un editor de propiedades, esto es, cada propiedad del Bean tiene su propio editor; normalmente las herramientas de desarrollo muestran los editores de propiedades de un Bean en una hoja de propiedades. Un editor de propiedades está asociado y edita un tipo de propiedad particular o utilizando personalizadores, los cuales ofrecen un completo control GUI sobre la personalización del Bean. Estos personalizadores se utilizan allí donde los editores de propiedades no son prácticos o no se pueden aplicar. Al contrario que un editor de propiedad, que está asociado con una propiedad, un personalizador está asociado con un Bean.

Editores de Propiedades

Un editor de propiedad es una herramienta para personalizar un tipo de propiedad particular. Los editores de propiedades se muestran, o se activan desde la hoja de propiedades. Una hoja de propiedades determinará el tipo de la propiedad, buscará un editor de propiedad adecuado, y mostrará el valor actual de la propiedad de una forma adecuada.

Los editores de propiedades deben implementar el interface PropertyEditor. Este interface proporciona métodos que especifican cuando una propiedad debe ser mostrada en una hoja de propiedades.

Personalizadores

Cuando se utiliza un personalizador de Bean, se obtiene el control completo sobre la configuración o edición del Bean. Un personalizador es como una aplicación que específicamente contiene la personalización de un Bean. Algunas veces las propiedades son insuficientes para representar los atributos configurables de un Bean. Los Personalizadores se usan cuando se necesita utilizar instrucciones sofisticadas para modificar un Bean, y cuando los editores de propiedades son demasiados primitivos para conseguir la personalización del Bean.

Todos los personalizadores deben:

- Descender de java.awt.Component o de una de sus subclases.
- Implementar el interface java.beans.Customizer. Esto significa la implementación de métodos para registrar objetos PropertyChangeListener, y lanzar eventos de cambio de propiedad a dichos oyentes cuando ocurre un cambio en el Bean fuente.
- Implementar un constructor por defecto.
- Asociar el personalizador con la clase fuente mediante BeanInfo.getBeanDescriptor().

Eventos

Los Beans utilizan los eventos para comunicarse con otros Beans. Un Bean que quiere recibir eventos (un Bean oyente) registra su interés con un Bean que lanza eventos (un Bean fuente). Las herramientas de desarrollo pueden examinar un Bean para determinar que eventos puede disparar (enviar) y cuales puede manejar (recibir).

Persistencia

Un Bean persiste cuando tiene sus propiedades, campos e información de estado almacenadas y restauradas desde un fichero. El mecanismo que hace posible la persistencia se llama serialización. Cuando un ejemplar de Bean es serializado se convierte en una canal de datos para ser escritos. Cualquier Applet, aplicación o herramienta que utilice el Bean puede "recontistuirlo" mediante la deserialización. Los

JavaBeans utilizan el API Object Serialization del JDK para sus necesidades de serialización. Siempre que una clase en el árbol de herencia implemente los interfaces Serializable o Externalizable, esa clase será serializable.

Todos los Bean deben persistir. Para persistir, los Beans deben soportar la serialización implementando los interfaces java.io.Serializable o java.io.Externalizable. Estos interfaces ofrecen la elección entre serialización automática o “manual”.

Propiedades

Una propiedad es un atributo del JavaBean que afecta a su apariencia o a su conducta. Por ejemplo, un botón puede tener las siguientes propiedades: el tamaño, la posición, el título, el color de fondo, el color del texto, si está o no habilitado, etc.

Los Beans exponen sus propiedades para poder ser personalizados durante el diseño. Las propiedades se exponen a las herramientas de desarrollo mediante los patrones de diseño o una clase BeanInfo.

Las propiedades de un Bean pueden examinarse y modificarse mediante métodos o funciones miembro, que acceden a dicha propiedad, y pueden ser de dos tipos:

- getter method: lee el valor de la propiedad
- setter method: cambia el valor de la propiedad.

Un IDE que cumpla con las especificaciones de los JavaBeans sabe como analizar un Bean y conocer sus propiedades. Además, crea una representación visual para cada uno de los tipos de propiedades, denominada editor de propiedades, para que el programador pueda modificarlas fácilmente en el momento del diseño.

Cuando un programador, toma un Bean de la paleta de componentes y lo deposita en un panel, el IDE muestra el Bean sobre el panel. Cuando seleccionamos el Bean aparece una hoja de propiedades, que es una lista de las propiedades del Bean, con sus editores asociados para cada una de ellas.

El IDE llama a los métodos o funciones miembro que empiezan por get, para mostrar en los editores los valores de las propiedades. Si el programador cambia el valor de una propiedad se llama a un método cuyo nombre empieza por set, para actualizar el valor de dicha propiedad y que puede o no afectar al aspecto visual del Bean en el momento del diseño.

Las especificaciones JavaBeans definen un conjunto de convenciones (design patterns) que el IDE usa para inferir qué métodos corresponden a propiedades, estas son:

- public void setNombrePropiedad(TipoPropiedad valor)
- public TipoPropiedad getNombrePropiedad( )

Cuando el IDE carga un Bean, usa el mecanismo denominado reflexión para examinar todos los métodos, fijándose en aquellos que empiezan por set y get. El IDE añade las propiedades que encuentra a la hoja de propiedades para que el programador personalice el Bean.

Las propiedades se pueden clasificar en varios tipos:

- Propiedades Simples: Representan un único valor.
- Propiedades Indexadas: Representan un arreglo de propiedades.
- Propiedades Ligadas (Bound): Los objetos de una clase que tiene una propiedad ligada notifican a otros objetos (listeners) interesados, cuando el valor de dicha propiedad cambia, permitiendo a estos objetos realizar alguna acción. Cuando la propiedad cambia, se crea un objeto (event) que contiene información acerca de la propiedad (su nombre, el valor previo y el nuevo valor), y lo pasa a los otros objetos (listeners) interesados en el cambio.
- Propiedades restringidas (constrained): Una propiedad restringida es similar a una propiedad ligada salvo que los objetos (listeners) a los que se les notifica el cambio del valor de la propiedad tienen la opción de vetar cualquier cambio en el valor de dicha propiedad.

Las herramientas de desarrollo están en capacidad de examinar a un Bean para exponer sus características (propiedades, eventos y métodos) en un hoja de propiedades, para esto utilizan la clase java.beans.Introspector. Esta clase utiliza el corazón de reflexión del

API del JDK para descubrir los métodos del Bean, y luego aplica los patrones de diseño de los JavaBeans para descubrir sus características. Como se mencionó anteriormente este proceso de descubrimiento se llama introspección.

De forma alternativa, se pueden exponer explícitamente las características del Bean en una clase asociada separada que implemente la interface BeanInfo. Asociando una clase BeanInfo con un Bean se puede:

- Exponer solo aquellas características que queremos exponer.
- Relegar en BeanInfo la exposición de algunas características del Bean, mientras se deja el resto para la reflexión de bajo nivel.
- Asociar un icono con el Bean fuente.
- Especificar una clase personalizada.
- Segregar las características entre normales y expertas.
- Proporcionar un nombre más descriptivo, información adicional sobre la característica del Bean.

Además BeanInfo define métodos que devuelven descriptores para cada propiedad, método o evento que se quiere exponer.

II. JAVA 2 MICROEDITION Y EL MIDP

2.1. TECNOLOGÍA JAVA 2 MICRO EDITION

Sun Microsystems define J2ME como “un ambiente de ejecución altamente optimizado de java que apunta a una amplia gama de productos de consumo que incluye buscapersonas, teléfonos celulares, PDAs, set-top boxes digitales y sistemas de navegación automovilísticos”[17]. Anunciado en Junio de 1999 en la Conferencia de Desarrolladores JavaOne, J2ME trae la plataforma de funcionalidad del lenguaje Java para dispositivos móviles que tienen una potencia de cálculo baja (velocidad de procesamiento y cantidad de memoria) e interfaces de usuario con capacidades restringidas.

2.1.1. Arquitectura J2ME

J2ME define un ambiente completo de ejecución de Java (Java Runtime Environment), abarca una máquina virtual, configuraciones, perfiles y un conjunto de paquetes adicionales u opcionales que dependen de la aplicación específica. J2ME utiliza las configuraciones y perfiles para personalizar el ambiente de ejecución de Java (JRE). La siguiente figura muestra la arquitectura de la tecnología en consideración.

Figura 2.1. Arquitectura de J2ME

2.1.2. Maquina virtual

Una máquina virtual de Java (JVM – Java Virtual Machine) es un programa encargado de interpretar código intermedio (bytecode) de los programas Java precompilados a código máquina ejecutable por la plataforma, efectuar las llamadas pertinentes al sistema operativo subyacente y observar las reglas de seguridad y corrección de código definidas para el lenguaje Java. De esta forma, la JVM proporciona al programa Java independencia de la plataforma con respecto al hardware y al sistema operativo. Las implementaciones tradicionales de JVM son, en general, muy pesadas en cuanto a memoria ocupada y requerimientos computacionales. J2ME define varias JVMs de referencia adecuadas al ámbito de los dispositivos electrónicos que, en algunos casos, suprimen algunas características con el fin de obtener una implementación menos exigente.

Como se verá más adelante, existen dos configuraciones la CLDC(Connected Limited Device Configuration) y CDC(Connected Device Configuration), cada una con unas características propias. Como consecuencia, cada una requiere su propia máquina virtual. La VM (Virtual Machine) de la configuración CLDC se denomina KVM(Kilobyte Virtual Machine) y la de la configuración CDC se denomina CVM(Compact Virtual Machine); a continuación se muestran sus principales características:

La KVM

Se trata de una máquina virtual java altamente portable y compacta, pensada y diseñada para funcionar en dispositivos con recursos limitados y con conexión a red. El objetivo de la tecnología KVM es el de crear la máquina virtual de java más pequeña y completa que mantuviese los aspectos más importantes del lenguaje de programación Java y que funcionase en dispositivos con procesadores de 16 o 32 bits y capacidades de unos pocos cientos de KiloBytes (originalmente 128 Kbytes) de memoria. Entre las características mas importantes de la KVM se tienen:

- Pequeña, con una carga de memoria entre los 40Kb y los 80Kb, dependiendo de la plataforma y las opciones de compilación.
- Alta portabilidad.

- Diseñada para ser lo más completa y rápida posible.

Debido a su reducido tamaño en memoria, la KVM presenta ciertas limitaciones en comparación con la clásica JVM, a saber:

- No existe soporte para JNI (*Java Native Interface*) debido a los recursos limitados de memoria.
- No existen cargadores de clases (*class loaders*) definidos por el usuario. Sólo existen los predefinidos.
- No se permiten los grupos de hilos o hilos *daemon* . Cuándo se quiera utilizar grupos de hilos se deben utilizar los objetos *Colección* para almacenar cada hilo en el ámbito de la aplicación.
- No existe la finalización de instancias de clases. No existe el método Object.finalize().
- No hay referencias débiles.
- Limitada capacidad para el manejo de excepciones debido a que el manejo de éstas depende en gran parte de las APIs de cada dispositivo por lo que son éstos los que controlan la mayoría de las excepciones.

La CVM

La CVM ha sido tomada como Máquina Virtual Java de referencia para la configuración CDC y soporta las mismas características que la Máquina Virtual de J2SE. Está orientada a dispositivos electrónicos con procesadores de 32 bits de gama alta y entornos de 2Mb o más de memoria RAM. Las características que presenta esta máquina virtual son:

- Sistema de memoria avanzado.
- Tiempo de espera bajo para el recolector de basura.
- Separación completa de la VM del sistema de memoria.
- Recolector de basura modularizado.
- Portabilidad.
- Rápida sincronización.
- Ejecución de las clases Java fuera de la memoria de sólo lectura (ROM).
- Soporte nativo de hilos.

- Baja ocupación en memoria de las clases.
- Proporciona soporte e interfaces para servicios en Sistemas Operativos de Tiempo Real.
- Conversión de hilos Java a hilos nativos.
- Soporte para todas las características de Java2 v1.3 y librerías de seguridad, referencias débiles, JNI, invocación remota de métodos (RMI – *Remote Method Invocation*), Interfaz de depuración de la Máquina Virtual (JVMDI - *Java Virtual Machine Depuration Interface*) .

2.1.3. Configuraciones

Este nivel es menos visible a los usuarios, pero es muy importante para los desarrolladores de perfiles. Define el conjunto mínimo de características de la KVM, las características soportadas del lenguaje de programación Java y las librerías de clases básicas y APIs (Application Programing Interface) disponibles en una categoría de dispositivos. Representaría una franja horizontal de dispositivos con un mínimo denominador común que los desarrolladores asumen que tienen todos los dispositivos con esta configuración.

Así mismo, una configuración define el ambiente de ejecución básico, es decir un conjunto de clases principales y una máquina virtual especifica que están orientadas a conformar el corazón de las implementaciones para dispositivos con características específicas.

Actualmente existen dos configuraciones para J2ME, la CLDC, orientada a dispositivos con limitaciones tanto a nivel de procesamiento como de interfaz gráfica y la CDC orientada a dispositivos con menos limitaciones.

A. Configuración para Dispositivos con Conexión (CDC)

La CDC esta orientada a dispositivos tales como decodificadores de televisión digital, televisores con Internet, algunos electrodomésticos y sistemas de navegación en

automóviles. La CDC usa una Máquina Virtual Java similar en sus características a una de J2SE, pero con limitaciones en el apartado gráfico y de memoria del dispositivo. Ésta Máquina Virtual es la que se conoce como CVM. La CDC está enfocada a dispositivos con las siguientes capacidades:

- Procesador de 32 bits.
- 2Mb o más de memoria total, incluyendo memoria RAM y ROM.
- Funcionalidad completa de la Máquina Virtual Java2.
- Conectividad a algún tipo de red.

La CDC está basada en J2SE v1.3 e incluye varios paquetes Java de la edición estándar. Las peculiaridades de la CDC están contenidas principalmente en el paquete javax.microedition.io, que incluye soporte para comunicaciones HTTP y basadas en datagramas.

B. Configuración para Dispositivos Limitados con Conexión (CLDC)

La CLDC está orientada a dispositivos dotados de conexión y con limitaciones en cuanto a capacidad gráfica, procesamiento y memoria. Un ejemplo de éstos dispositivos son: teléfonos móviles, buscapersonas(pagers), PDAs, organizadores personales, etc. La mayoría de las restricciones vienen dadas por el uso de la KVM, necesaria al trabajar con la CLDC debido a su pequeño tamaño. Los dispositivos que usan CLDC deben cumplir los siguientes requisitos:

- 160Kb y 512Kb de memoria total disponible. Como mínimo se debe disponer de 128Kb de memoria no volátil para la Máquina Virtual Java y las bibliotecas CLDC, y 32Kb de memoria volátil para la Máquina Virtual en tiempo de ejecución.
- Procesador de 16 o 32 bits con al menos 25Mhz de velocidad.
- Bajo consumo, debido a que éstos dispositivos trabajan con suministro de energía limitado, normalmente baterías.
- Conexión a algún tipo de red, normalmente sin cable, con conexión intermitente y ancho de banda limitado (unos 9600bps).

La CLDC aporta las siguientes funcionalidades a los dispositivos:

- Un subconjunto del lenguaje Java y todas las restricciones de su Máquina Virtual(KVM).
- Un subconjunto de las bibliotecas Java del núcleo.
- Soporte para Entrada/Salida básica.
- Soporte para acceso a redes.
- Seguridad.

2.1.4. Perfiles

Este nivel es el más visible a los usuarios y desarrolladores de aplicaciones. Las aplicaciones se desarrollan sobre un determinado perfil que a su vez está implementado sobre una determinada configuración. Un perfil define un conjunto de APIs y características comunes para una franja vertical de dispositivos. Las clases de un perfil permiten el acceso a funcionalidades específicas de los dispositivos como la interfaz gráfica, funcionalidades de red, almacenamiento persistente, etc.

Las aplicaciones desarrolladas sobre un determinado perfil van a ser portables a cualquier dispositivo que soporte ese perfil; cabe destacar que un dispositivo puede soportar varios perfiles y que sobre una configuración pueden existir diversos perfiles. De igual forma, un perfil define las APIs que controlan el ciclo de vida de la aplicación, interfaz de usuario, etc. Más concretamente, un perfil es un conjunto de APIs orientado a un ámbito de aplicación determinado. Los perfiles identifican un grupo de dispositivos por la funcionalidad que proporcionan (electrodomésticos, teléfonos móviles, etc.) y el tipo de aplicaciones que se ejecutarán en ellos. Las librerías de la interfaz gráfica son un componente muy importante en la definición de un perfil. Aquí se pueden encontrar grandes diferencias entre interfaces, desde el menú textual de los teléfonos móviles hasta los táctiles de los PDAs.

El perfil establece unas APIs que definen las características de un dispositivo, mientras que la configuración hace lo propio con una familia de ellos. Esto hace que a la hora de construir una aplicación se cuente tanto con las APIs del perfil como de la configuración. Se debe tener en cuenta que un perfil siempre se construye sobre una configuración

determinada, de este modo, podemos pensar en un perfil como un conjunto de APIs que dotan a una configuración de funcionalidad específica.
En J2ME hasta el momento se han definido los siguientes perfiles:

Foundation Profile: Este perfil define una serie de APIs sobre la CDC orientadas a dispositivos que carecen de interfaz gráfica como, por ejemplo, decodificadores de televisión digital. Este perfil incluye gran parte de los paquetes de J2SE, pero excluye totalmente los paquetes "java.awt" *Abstract Windows Toolkit* y "java.swing" que conforman la interfaz gráfica de usuario (GUI) de J2SE. Si una aplicación requiriera una GUI, entonces sería necesario un perfil adicional.

Personal Profile: El *Personal Profile* está construido sobre la CDC y es un subconjunto de la plataforma J2SE v1.3, que proporciona un entorno con un completo soporte gráfico AWT. El objetivo es el de dotar a la configuración CDC de una interfaz gráfica completa, con capacidades web y soporte de applets Java. Este perfil requiere una implementación del *Foundation Profile* .

RMI Profile : Este perfil también está construido sobre la CDC y requiere que una implementación del *Foundation Profile* se construya debajo de él . El perfil RMI soporta un subconjunto de las APIs J2SE v1.3 RMI, debido a que algunas características de estas APIs se han eliminado del perfil RMI como consecuencia de las limitaciones de cómputo y memoria de los dispositivos.

PDA Profile : El PDA Profile está construido sobre CLDC. Pretende abarcar PDAs de gama baja, tipo Palm, con una pantalla y algún tipo de puntero (ratón o lápiz) y una resolución de al menos 20000 pixels (al menos 200x 100 pixels) con un factor 2:1. No es posible dar mucha más información porque en este momento este perfil se encuentra en fase de definición.

MID Profile : Está construido sobre la CLDC y es el perfil más usado actualmente para el desarrollo de aplicaciones. Se ampliará la información sobre este perfil en la sección 2.2.

2.1.5. Proceso de desarrollo de una aplicación J2ME

A continuación se describe de manera general el proceso que se realiza cuando se desarrolla una aplicación para dispositivos móviles haciendo uso de la tecnología J2ME.

Escritura de los archivos Java: esta fase corresponde a la escritura del código fuente de una aplicación, este código puede estar organizado en paquetes, lo cual es recomendable para preservar el orden de la aplicación.

Compilación: en esta fase se genera el código objeto de los archivos java (archivos *class*). Se debe notar que para la versión micro de la plataforma Java2 también se usa el compilador estándar de java, pero con el conjunto de bibliotecas definidas para la configuración y el perfil utilizados.

Preverificación: al contrario de lo que pasa con las otras versiones de la plataforma Java, en las cuales con los archivos *class* es suficiente para ejecutar una aplicación, en J2ME se hace necesario el proceso de preverificación, el cual consiste en hacer un examen del código objeto generado con el fin de garantizar que no viola ninguna de las características de seguridad de la plataforma J2ME y que los archivos *class* generados son completamente compatibles con las bibliotecas de la configuración y el perfil utilizados.

Empaquetado: para la ejecución de aplicaciones J2ME en ambientes reales (como teléfonos móviles que soporten el ambiente de ejecución de J2ME) se hace necesario que las aplicaciones desarrolladas se encuentren empaquetadas dentro de un archivo JAR (Java ARchives), el cual típicamente contiene los ficheros de clases y los recursos auxiliares asociados con la aplicación. Estos recursos auxiliares podrían incluir, por ejemplo, ficheros de imagen y sonido. Los archivos JAR requieren de un archivo especial llamado *Manifest.mf*, que sirve para describir su contenido. Para el caso de los asistentes digitales personales (de sus siglas en inglés PDA), se requiere que el contenido de las aplicaciones se encuentre empaquetado en archivos PRC (Palm ResourCe).

Descripción de la aplicación: al igual que un archivo JAR, para la ejecución de aplicaciones J2ME en un ambiente de ejecución real, se requiere también de un archivo

JAD (Java Descriptor), el cual es básicamente una descripción de las características de la aplicación desarrollada y características especificas del JAR como tamaño en bytes y dirección de su ubicación.

2.2. EL MIDP (Mobile Information Device Profile)

El primer perfil desarrollado se denomina MIDP, que en conjunto está diseñado para operar con la CDLC (figura 2.1) y permitir la ejecución de aplicaciones java en dispositivos móviles MID(Mobile Information Devices). En el desarrollo de este perfil participan empresas tales como Ericcson, Motorola, Nokia, NTT DoCoMo, Palm Computing, Sony, Siemens y Sun Microsystems, entre otros e incluye teléfonos celulares y PDAs (Personal Digital Assistants).

Las aplicaciones que se desarrollan con J2ME sobre el MIDP reciben el nombre de MIDlets. Los MIDlest deben agruparse en archivos JAR para que sea posible su distibución y en un archivo JAR se pueden incluir varios MIDlets y a este conjunto se le denomina MIDlet Suite.

Hasta el momento se han definido dos perfiles, el MIDP 1.0 y el MIDP 2.0, cuyas especificaciones finales salieron en Septiembre de 2000 y Noviembre de 2002 respectivamente.

2.2.1 MIDP 1.0

Este perfil está construido sobre la configuración CLDC. Al igual que CLDC fue la primera configuración definida para J2ME, MIDP fue el primer perfil definido para esta plataforma.

A. Hardware

Este perfil está orientado para dispositivos con las siguientes características:

- Reducida capacidad computacional y de memoria.
- Conectividad limitada (en torno a 9600bps).

- Capacidad gráfica muy reducida (mínimo un display de 96x54 píxeles monocromo).
- Entrada de datos alfa numérica reducida.
- 128Kb de memoria no volátil para componentes MIDP.
- 8Kb de memoria no volátil para datos persistentes de aplicaciones.
- 32Kb de memoria volátil en tiempo de ejecución para la pila Java.

B. Capacidades de las APIs

El MIDP 1.0 incluyó solo las APIs que eran consideradas como requerimiento absoluto para alcanzar una amplia portabilidad, estas APIs están relacionadas con:

- Aplicación (Definición de la semántica y control de la aplicación MIDP)
- Interfaz de usuario (manejo de entradas y Despliegues)
- Almacenamiento persistente
- Comunicaciones en red.
- Temporizadores(Timers)

2.2.2 MIDP 2.0

Desde su introducción, MIDP 1,0 ha prometido ser la plataforma de programación universal más importante para los dispositivos móviles. Esta versión del Mobile Information Device Profile provee APIs estándar para el desarrollo de aplicaciones, incluyendo APIs para el manejo del ciclo de vida de las aplicaciones, conectividad con redes HTTP, interfaces de usuario y almacenamiento persistente. La versión 2.0 del MIDP incluye muchas mejoras y adiciones a la versión anterior, principalmente en los paquetes de interfaces graficas de usuario, conectividad a redes y seguridad.

La evolución tecnológica que se ha dado en los dispositivos móviles modernos, los cuales proporcionan más memoria, pantallas a color mejoradas, soporte para multimedia (video, MP3) y cámara fotográfica incorporada también se ve reflejada en la versión 2.0 del MIDP, la cual mejora las capacidades de la plataforma de Java para los dispositivos móviles.

A. Hardware

Para poder ejecutar aplicaciones MIDP 2.0, los dispositivos deberían tener las siguientes características mínimas:

- Despliegue
  - ☀ Tamaño de Pantalla: 96x54 píxeles
  - ☀ Profundidad de despliegue: 1-bit
- Entradas:
  - ☀ Teclado
  - ☀ Pantalla de tacto
- Memoria:
  - ☀ 256 kilobytes de memoria no volátil para la aplicación MIDP, más de la que requiere CLDC.
  - ☀ 8 kilobytes de memoria no volátil para datos persistentes creados en la aplicación.
  - ☀ 128 kilobytes de memoria volátil para el ambiente de ejecución Java.
- Conexión a redes:
  - ☀ Dos vías, acceso inalámbrico posiblemente intermitente con ancho de banda limitado.
- Sonido:
  - ☀ Capacidad para reproducir tonos, vía Hardware o software.

B. Capacidades de las APIs

Al igual que MIDP 1.0, la versión 2.0 de este perfil solo incluyó las APIs consideradas como requerimiento indispensable para asegurar la portabilidad de las aplicaciones.

El MIDP 2.0 proporciona APIs para:

- Manejo del ciclo de vida de las Aplicaciones (Definición de la semántica de una aplicación MIDP y cómo ésta es controlada).
- Firmado de aplicaciones y seguridad para dominios privilegiados.
- Transacciones seguras entre usuarios finales (HTTPS)
- Registro de aplicaciones push (modelo de servidor push)
- Interconexión a redes.
- Almacenamiento persistente
- Sonido
- Temporizadores
- Interfaces de usuario mejoradas (que incluye despliegue, entradas y soporte para juegos).

2.3. COMPARACIÓN ENTRE LOS PERFILES MID1.0 Y MID2.0

En los apartados anteriores se vieron las características generales relacionadas con el hardware de los dispositivos y capacidades de las APIs al ejecutar aplicaciones en las versiones 1.0 y 2.0 del MIDP, a continuación se tratarán las diferencias software, tomando como punto de comparación los tipos de paquetes que define cada perfil (para una comparación mas detallada, ver Anexo A: Comparación entre MIDP 1.0 y MIDP2.0).

La especificación del MIDP 2.0 ha determinado la existencia de ocho tipos de paquetes, incluyendo los paquetes del núcleo, mientras que MIDP 1.0 define la existencia de cuatro tipos de paquetes, incluyendo también los paquetes del núcleo. En la siguiente tabla se

muestran los paquetes incluidos en cada uno de los perfiles indicando el tipo de paquete al que hacen referencia.

Tipo Del Paquete	Paquete MID2.0	Paquetes MID1.0
Interfaz de Usuario	javax.microedition.lcdui	javax.microedition.lcdui
Juegos	javax.microedition.lcdui.game	-----
Ciclo de vida	javax.microedition.midlet	javax.microedition.midlet
Interconexión a Redes	javax.microedition.io	javax.microedition.io
Seguridad	javax.microedition.pki	-----
Sonido	javax.microedition.media	-----
	javax.microedition.media.control	-----
Persistencia de datos	javax.microedition.rms	javax.microedition.rms
Básicos o del Núcleo	java.io	Java.io
	java.lang	java.lang
	java.util	java.util

Tabla. 2.1 Paquetes y tipos de paquetes en MIDP 1.0 y MIDP2.0

2.3.1. Paquetes para el manejo de Interfaces gráficas de usuario

Tanto MIDP1.0 como MIDP2.0 definen el mismo paquete para la creación de interfaces de usuario (Tabla 2.1) en el dispositivo móvil, no obstante, MIDP 2.0 incluye soporte para interfaces de usuario mejoradas que permiten a los desarrolladores la creación de aplicaciones más atractivas, a través de nuevas clases e interfaces o la adición de métodos y atributos a las clases existentes. MIDP2.0 proporciona al desarrollador la capacidad para la creación de sus propios objetos gráficos que podrá desplegar en pantalla.

2.3.2. Paquetes para la creación de Juegos

El perfil MID2.0, ha incluido el paquete ***javax.microedition.lcdui.game***, el cual provee una serie de clases que posibilitan el desarrollo de juegos con rico contenido gráfico en dispositivos inalámbricos.

Los dispositivos inalámbricos tienen mínimas capacidades de procesamiento, por lo cual la mayor parte de esta API intenta mejorar su funcionamiento reduciendo al mínimo la cantidad de trabajo hecho en java; esta característica trae además el beneficio adicional de reducir el tamaño de la aplicación. El API está estructurado para proveer considerable libertad cuando se implemente sobre él, permitiendo el uso extensivo de código nativo, aceleración de hardware y formatos de datos de imagen específicos del dispositivo cuando sea necesario.

El API usa las clases para gráficos estándar de bajo nivel de MIDP (*Graphics*, *Image*,etc) para poder utilizar las clases del API de juegos de alto nivel en conjunto con primitivas gráficas, por ejemplo es posible un fondo complejo utilizando el API de juegos y dibujar algo encima de él utilizando primitivas gráficas como *drawLine*, etc.

2.3.3. Paquetes para el manejo del ciclo de vida de las aplicaciones

Define las aplicaciones MIDP y las interacciones entre la aplicación y el entorno sobre el cual se está ejecutando.

MIDP define un modelo de aplicación para permitir que los recursos limitados de los dispositivos sean compartidos por múltiples aplicaciones MIDP. Este modelo define que es un MIDlet, como es empaquetado, que ambiente de ejecución está disponible para ese MIDlet y cómo debe comportarse de tal forma que puedan ser administrados los recursos del dispositivo.

En el paquete javax.microedition.midlet solo se encuentra la clase *MIDlet*. La aplicación debe extender esta clase para permitir al software de administración de aplicaciones controlar al *MIDlet* y permitirle recuperar características del descriptor de la aplicación así como notificar y solicitar cambios de estado. Los métodos de esta clase permiten crear, iniciar, pausar y destruir una aplicación.

El MIDP 2.0 ha adicionado el concepto de “Trusted Applications” o aplicaciones confiables, a través del cual permite a aplicaciones marcadas con este calificativo acceder

a recursos y APIs que en el MIDP1.0 eran consideradas como restringidas por la seguridad del dispositivo.

2.3.4. Paquete para persistencia de datos

El MIDP provee un mecanismo para que los MIDlets almacenen datos en forma persistente y posteriormente estos se puedan recuperar. Este mecanismo de almacenamiento es modelado como una pequeña base de datos orientada a registro y se conoce con el nombre de Record Managment System (RMS).

Un record store consiste de una colección de registros que permanecen persistentes durante múltiples invocaciones de un MIDlet. La plataforma es la responsable de mantener la integridad de los RecordStores de un MIDlet, durante el uso normal de la plataforma, incluyendo reinicio, cambios de batería, etc.

Los MIDlets dentro de un MIDlet suite pueden crear múltiples RecordStores, mientras cada uno de ellos tenga distinto nombre. Cuando un MIDlet suite es removido de una plataforma, todos los RecordStores asociados con estos MIDlets deben también ser removidos. Los MIDlets dentro de un MIDlet suite pueden acceder a los RecordStores directamente. Nuevas APIs en MIDP 2.0 permiten compartir explícitamente los RecordStores si el MIDlet crea el RecordStore seleccionando tal permiso. Los controles de acceso se definen cuando un RecordStores que se compartirá es creado. Los controles de acceso se hacen cumplir cuando los RecordStores son abiertos. Los modos de acceso permiten uso privado o compartido con cualquiera de los MIDlets suite. Los nombres de los RecordStores son sensibles a las mayúsculas y minúsculas y pueden estar formados por combinaciones de hasta 32 caracteres Unicode. Los nombres de los RecordStores deben ser únicos dentro del alcance de un MIDlet suite; en otras palabras, MIDlets dentro de un MIDlet suite no deben permitir crear más de un RecordStore con el mismo nombre; sin embargo, un MIDlet en un MIDlet suite puede tener un RecordStore con el mismo nombre que un MIDlet en otro MIDlet suite. En este caso, los RecordStores son considerados distintos y separados.

La implementación de un registro asegura que todas las operaciones individuales de un RecordStore son atómicas, síncronas y serializables de tal manera que no exista corrupción con múltiples accesos. Sin embargo, si un MIDlet usa múltiples hilos para acceder a un record store, este MIDlet es el responsable de coordinar estos accesos, o se pueden tener consecuencias no deseadas.

2.3.5. Paquetes para interconexión con redes

MIDP 2.0 ha adicionado soporte para conexiones seguras a través del protocolo HTTPS, el cual es básicamente HTTP sobre la capa de sockets seguros (SSL). SSL es un protocolo que cifra los datos para ser enviados a través de la red y provee autenticación para sockets finales.

Aunque HTTP y HTTPS son las únicas interfaces requeridas que una implementación MIDP debería soportar, varias interfaces adicionales de red han sido definidas en el MIDP2.0, el cual incluye soporte para comunicaciones a través de sockets (TCP y UDP), y comunicación a través de puertos seriales.

Adicionalmente una de las nuevas características más potentes del MIDP2.0 es lo que se ha denominado “Push Registry”, que activa aplicaciones dormidas cuando está disponible cierta información. Un MIDlet puede registrar un *Listener* de tal forma que cuando el MIDlet no está activo, el software de gestión de aplicaciones de la plataforma MIDP (AMS) escucha peticiones entrantes. Cuando una de estas peticiones es recibida, el AMS lanza al MIDlet asociado usando el método startApp(). A través de este sistema, por ejemplo, un proceso servidor puede activar una aplicación en el dispositivo para notificar un nuevo evento, aunque la aplicaciones no esté activa en ese preciso momento, esto se hace a través de la clase *PushRegistry*.

2.3.6. Paquetes para seguridad

MIDP 2.0 mejora de manera considerable el modelo de seguridad del MIDP 1.0 a través del paquete ***javax.microedition.pki***. Este paquete provee el manejo de certificados que son usados para la autenticación de la información en conexiones seguras.

En MIDP 2.0 es posible realizar firmado de aplicaciones y manejar dominios privilegiados. Con el firmado de aplicaciones podemos comprobar la identidad del proveedor de la aplicación y por lo tanto confiar en la misma. A través de los privilegios de dominio los proveedores de aplicaciones y los vendedores de dispositivos, pueden definir qué APIs son consideradas como restringidas. Estas nuevas características protegen al dispositivo frente accesos no autorizados por parte de las aplicaciones. Los protocolos seguros estándar como HTTPS, TLS/SSL y WTLS permiten la transmisión segura de datos a través de la encriptación de los mismos.

2.3.7. Paquetes para el manejo de audio

El rango de dispositivos J2ME va desde teléfonos celulares con una simple generación de tonos hasta PDAs con avanzadas capacidades de audio y video. Para trabajar con las diversas configuraciones y capacidades de procesamiento de multimedia, se necesita una API con un alto grado de abstracción. La meta del trabajo de MMAPI(Mobile Media API) Expert Group ha sido encaminada a un amplio rango de áreas de aplicaciones, y el resultado del trabajo ha sido ofrecer dos conjuntos de APIs: *Mobile Media API (JSR 135)* y *MIDP2.0 Media API.*

La primera API está enfocada para dispositivos J2ME con avanzadas capacidades de audio y multimedia. La última API es un subconjunto directamente compatible con el API de multimedia y esta enfocada para dispositivos con mayor cantidad de restricciones.

La MIDP2.0 Media API, al ser un subconjunto de la MMAPI, difiere respecto a esta de la siguiente forma: es únicamente para el manejo de audio, es decir no incluye ninguno de los controles específicos para proveer el manejo de video o gráficos y, no incluye el paquete javax.microedition.media.protocol, utilizado para el manejo de protocolos personalizados. Cuenta con una sola clase, ***Manager***, la cual sirve como punto de acceso para obtener los diferentes recursos del sistema. Las demás funcionalidades son provistas a través de interfaces de los paquetes ***javax.microedition.media*** y ***javax.microedition.media.control***.

III. PROMETEO, PLATAFORMA INTEGRADA DE DESARROLLO DE APLICACIONES PARA DISPOSITIVOS MÓVILES CON J2ME

3.1. Descripción General

PROMETEO es una plataforma de desarrollo implementada en el lenguaje de programación java, haciendo uso de los paradigmas de la orientación a objetos y la programación basada en componentes, que busca reducir el tiempo de desarrollo y emulación de aplicaciones para dispositivos móviles que utilicen tecnología Java 2 Microedition, dando soporte a los perfiles MID1.0 y MID2.0 y a las configuraciones CLCD 1.0 y CLDC 1.1. Como plataforma integrada de desarrollo, provee un ambiente de construcción y un ambiente de ejecución para este tipo de aplicaciones.

Adicionalmente, cuenta con generadores automáticos de código(tanto del tipo *wizard*, como del tipo *drag and drop*) que pueden ser configurados por el desarrollador para una aplicación especifica y que buscan acelerar el desarrollo.

El ambiente para la construcción de las aplicaciones proporciona un editor de código que identifica las palabras propias del lenguaje de programación, de igual forma proporciona un modelo de la aplicación que se está desarrollando (paquetes, clases, recursos) y un modelo de la clase que esté activa en el editor (atributos y métodos). El ambiente de ejecución corresponde al ambiente provisto por el J2ME *Wireless Toolkit* (WTK) de Sun Microsystems, que se integra a la plataforma de manera muy sencilla y una vez integrado, su utilización es transparente para el desarrollador.

Por estar implementado en Java, es independiente del sistema operativo utilizado, particularmente, hasta el momento ha sido probado exitosamente en Windows XP y Windows 2000 professional, así como Linux Mandrake9.1, Linux Slakeware 9.1 y Linux Suse 8.1. Sin Embargo, durante su desarrollo, se encontraron características propias del

lenguaje de programación que comprometían de cierta manera la portabilidad del código y que son detalladas en al anexo C (Inconsistencias de Portabilidad).

Una visión de la interfaz grafica de usuario principal de la plataforma construida se muestra en la figura 3.1.

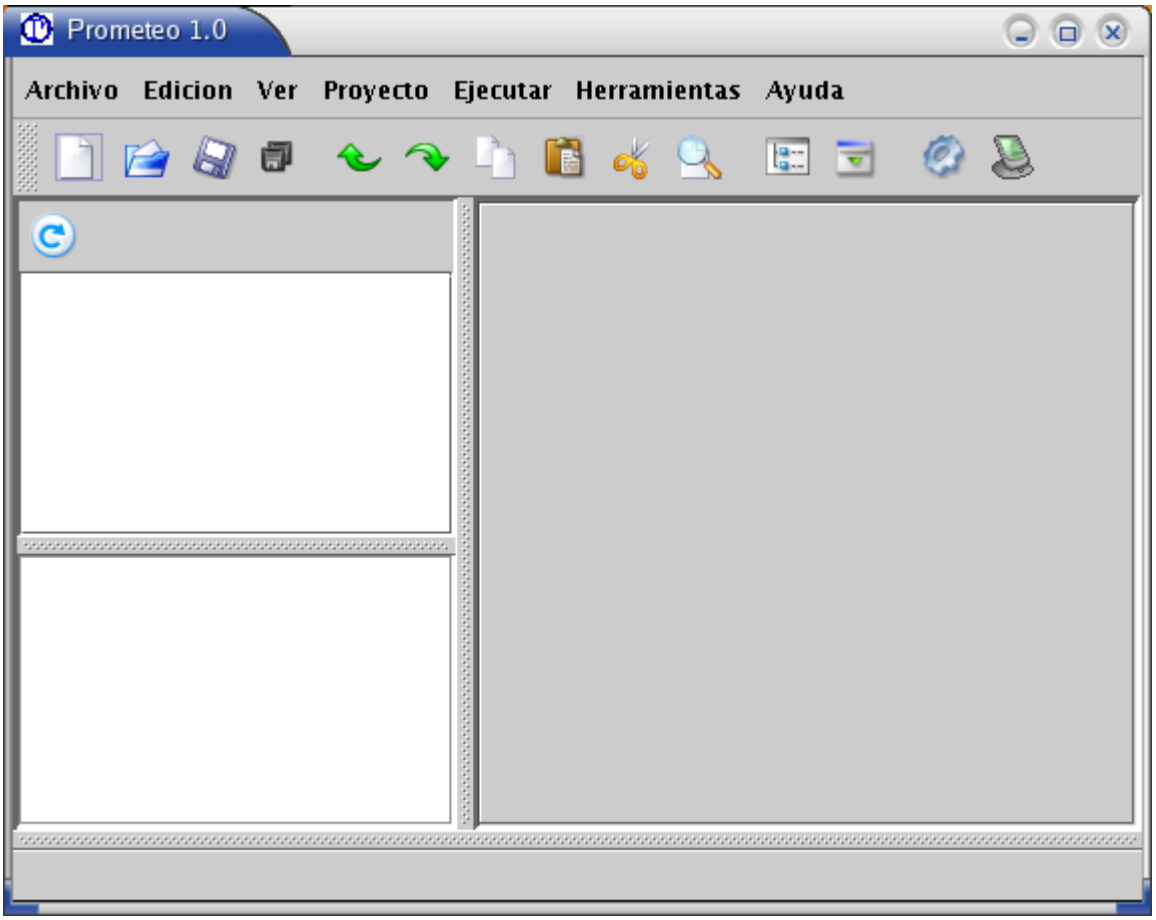

Figura 3.1. GUI principal de Prometeo

3.2. Proceso de desarrollo

La construcción de PROMETEO como plataforma de desarrollo de aplicaciones para dispositivos móviles en su primer ciclo (versión 1.0), cubrió las etapas de desarrollo de un producto software: análisis, diseño, implementación y pruebas. Sin embargo, por tratarse de una plataforma que sirve para construir otras aplicaciones software, la etapa de análisis cobró una importancia adicional a la del desarrollo de una aplicación convencional

ya que no se hizo el análisis de un dominio específico de aplicación (el caso de una aplicación software convencional), sino que se hizo necesario analizar varios dominios en los que pudiera tener cabida una aplicación para un dispositivo móvil y con mayor razón si se pretendía dotar a la plataforma de características adicionales como los generadores de código.

A continuación se muestran los resultados que se consideran más importantes en cada una de las etapas del proceso de desarrollo.

3.2.1. Análisis

El análisis de la plataforma a construir se realizó en su mayor parte mientras el equipo de desarrollo cursaba la materia “Desarrollo de aplicaciones para dispositivos móviles”; las clases, las prácticas, los talleres, el trabajo final y la utilización de otras plataformas de desarrollo permitieron tener una idea clara de las características con las que debería contar una plataforma para el desarrollo de aplicaciones para dispositivos móviles, al menos en su versión inicial. Y permitió tener una idea de los generadores de código más generales que pudieran ser útiles para el desarrollo de cualquier aplicación.

Producto del análisis se identificó un único actor interactuando con el sistema:

Desarrollador: es la persona que hace uso de las funcionalidades que provee la plataforma integrada de desarrollo para implementar las aplicaciones para dispositivos móviles sobre la plataforma java 2 MicroEdition.

Adicionalmente se identificaron nueve requerimientos generales que debería cumplir la plataforma de desarrollo a implementar:

- Gestionar un proyecto: entendiendo por proyecto una carpeta con un nombre específico que contiene todos los archivos necesarios para desarrollar una aplicación.
- Gestionar Archivos: capacidad para abrir, cerrar y eliminar tipos específicos de archivos.
- Editar Código

- Generar Código
- Compilar una Aplicación
- Generar Archivos: archivos específicos para la ejecución de las aplicaciones.
- Ejecutar una aplicación.
- Configurar el entorno.
- Obtener Ayuda

La figura 3.2 muestra el diagrama de casos de uso del análisis obtenido a partir de los requerimientos generales.

Se hace notar que los requerimientos generales y su respectivo diagrama de casos de uso se encuentran a un nivel de abstracción bastante alto y pueden considerarse como requerimientos y casos de uso comunes a cualquier plataforma de desarrollo.

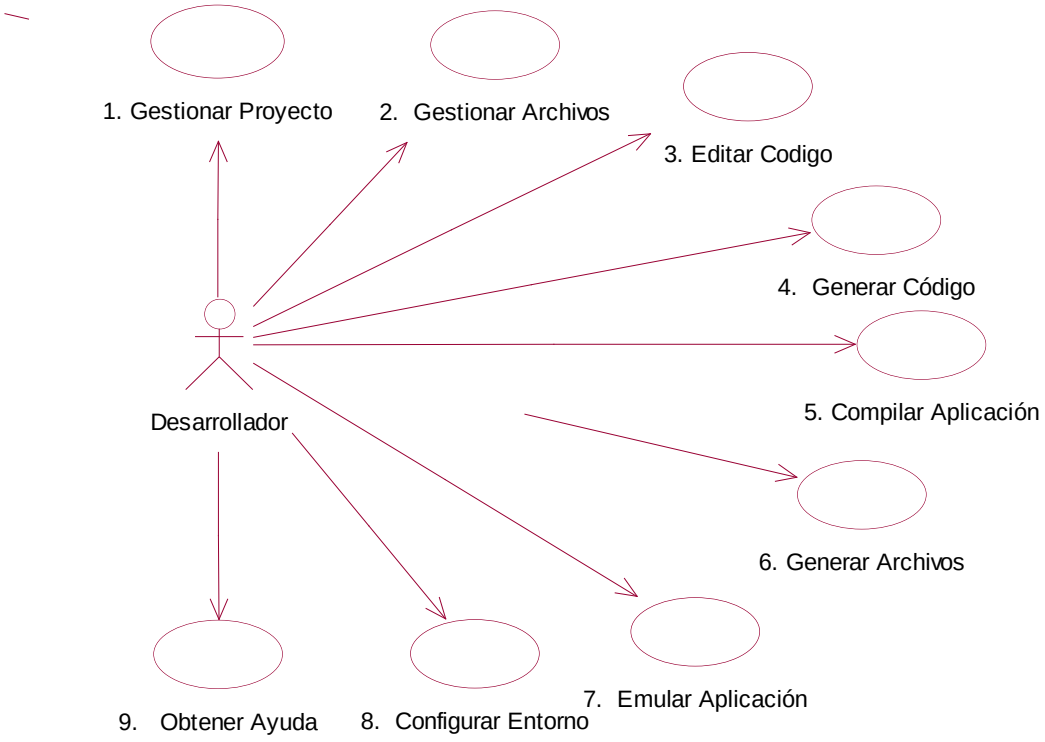

Figura 3.2. Diagrama de casos de uso del Análisis

Una vez identificados los requerimientos generales de una plataforma de desarrollo, se hizo necesaria la particularización de estos para determinar los requerimientos propios de una plataforma que permitiera desarrollar aplicaciones para dispositivos móviles utilizando la tecnología Java 2 MicroEdition. Producto de esto, se identificó que la plataforma a implementar debe:

- Permitir al desarrollador administrar adecuadamente los proyectos que cree dentro del entorno de desarrollo (crear, eliminar, guardar, abrir, cerrar los proyectos).
- Permitir al desarrollador administrar adecuadamente los paquetes y clases que cree dentro del ambiente de desarrollo (crear, eliminar, guardar, abrir, cerrar paquetes y clases).
- Permitir al desarrollador la creación y edición del código generado dentro del ambiente de desarrollo (copiar, cortar, pegar código).
- Permitir al desarrollador la creación de mecanismos para manejar almacenamiento persistente de datos en el dispositivo móvil.
- Permitir al desarrollador la creación de mecanismos para la interconexión con redes de datos desde el dispositivo móvil.
- Permitir al desarrollador la creación rápida de Interfaces Gráficas de Usuario (GUI).
- Permitir al desarrollador la compilación de las aplicaciones creadas dentro del ambiente de desarrollo.
- Permitir al desarrollador la emulación de las aplicaciones creadas dentro del ambiente de desarrollo.
- Permitir al desarrollador la creación de archivos necesarios para la utilización de las aplicaciones en entornos reales (archivos .jad y .jar).
- Permitir al desarrollador la configuración de bibliotecas de clases, tanto en la plataforma como en una aplicación específica (crear, modificar, eliminar bibliotecas de clases).
- Permitir al desarrollador la integración de diferentes emuladores al ambiente de desarrollo.
- Permitir al desarrollador escoger el emulador en el que va a probar su aplicación.
- Permitir al desarrollador la adición y manejo de recursos en sus aplicaciones (imágenes, sonido, texto).

- Brindar ayuda al desarrollador, con respecto a la plataforma y con respecto al lenguaje de programación.

3.2.2. Diseño

Esta etapa del proceso de desarrollo comenzó con llevar a un modelo de casos de uso los requerimientos específicos obtenidos de la etapa anterior, producto de esto se obtuvo el diagrama de casos de uso de diseño y la descripción de cada caso de uso.

A. Diagrama de casos de uso

Dado su tamaño, el diagrama de casos de uso se muestra en este documento en dos figuras diferentes (Figuras 3.3A y Figura 3.3B).

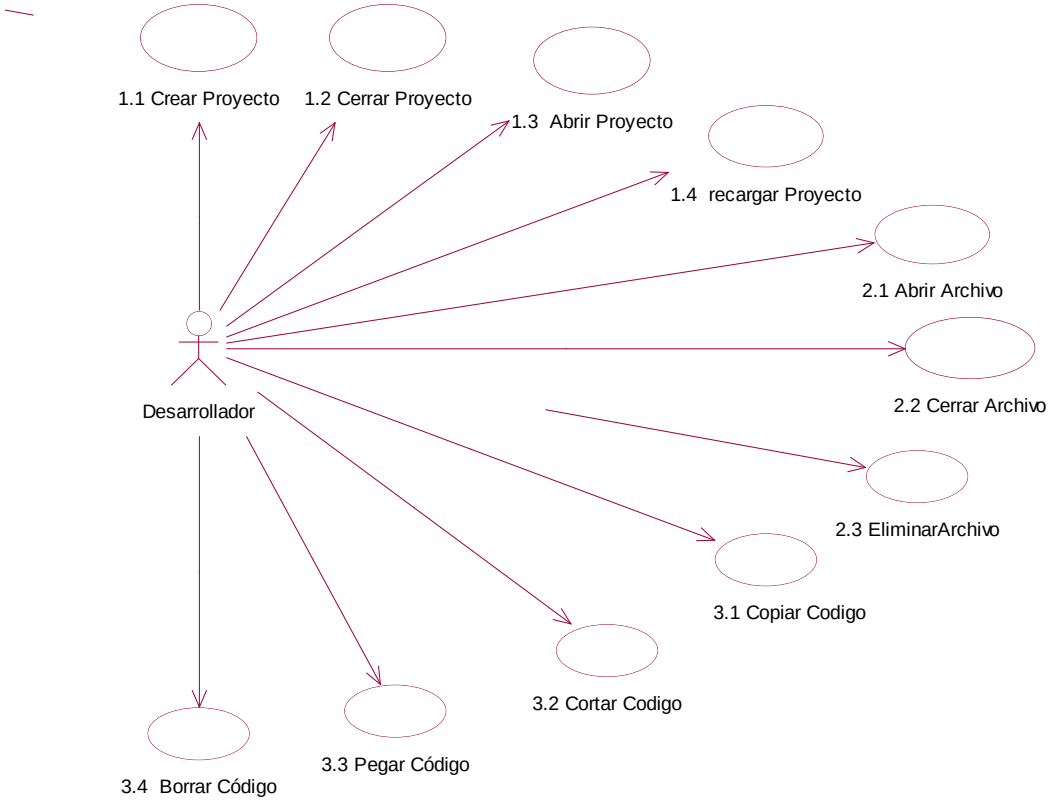

Figura 3.3A Casos de uso de Diseño

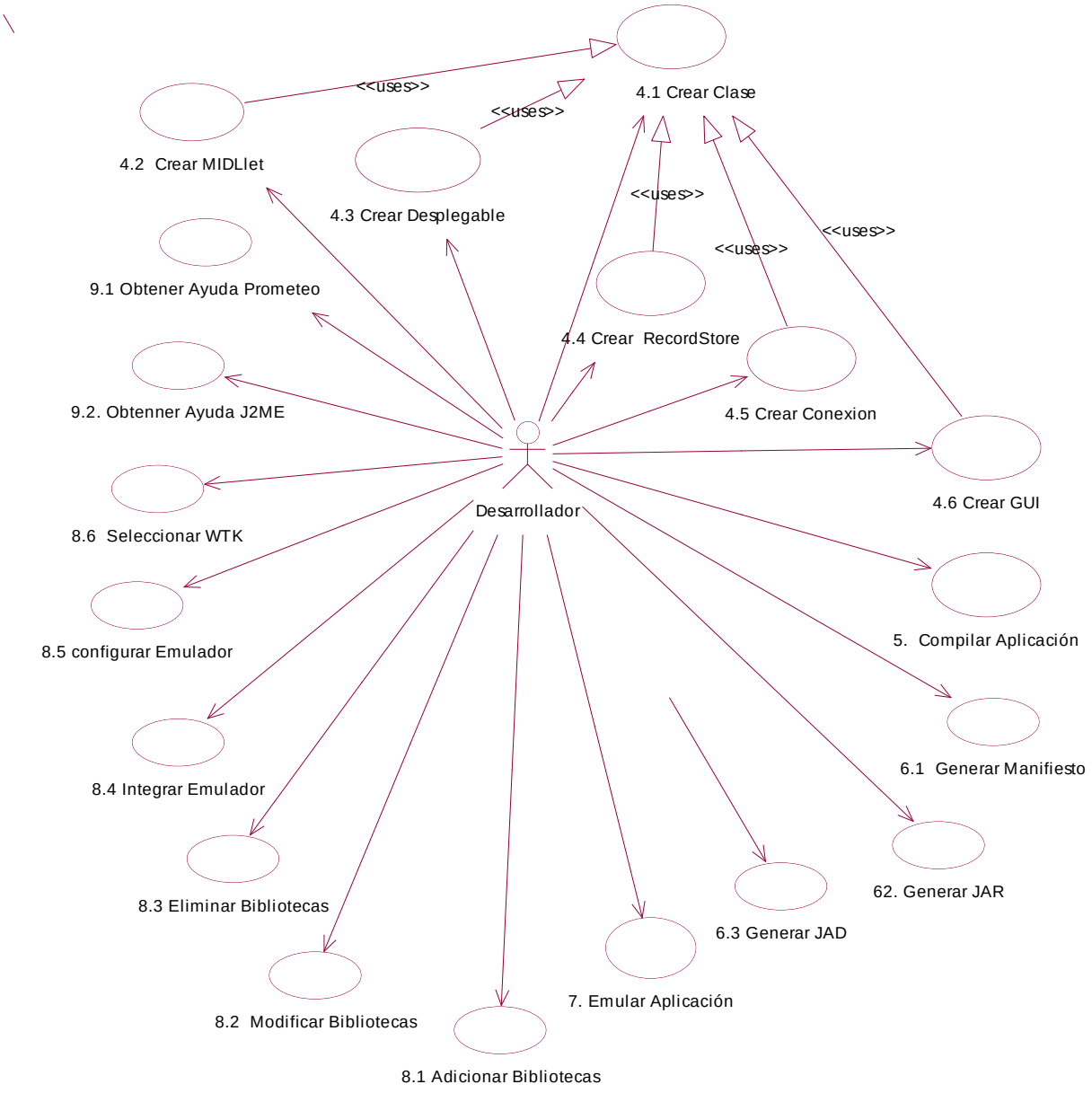

Figura 3.3B. Casos de uso de diseño

B. Descripción de los casos de uso

Caso de uso No. 1: Crear Proyecto

Actores: Desarrollador (iniciador)

Propósito: Permitir al desarrollador la creación de un directorio en donde se almacenen todos los recursos necesarios para su aplicación (código fuente, código objeto, archivos XML, JAR, JAD, imágenes, etc).

Resumen: el desarrollador proporciona el nombre y la ubicación del proyecto en el sistema de archivos. El sistema crea una carpeta con el nombre del proyecto y le muestra al desarrollador el nombre del nuevo proyecto en la GUI principal.

Flujo principal:

- Este caso de uso empieza cuando el desarrollador hace clic sobre el icono Nuevo o cuando selecciona en el menú la opción Archivo / Nuevo.
- El sistema responde presentando la GUI mostrada en la figura 3.4.
- El desarrollador hace clic sobre la opción Proyecto.
- El sistema responde mostrando la GUI mostrada en la figura 3.5.
- El desarrollador proporciona el nombre y la ubicación del proyecto.
- El desarrollador selecciona Aceptar.
- El sistema crea una carpeta con el nombre del proyecto, dentro de esta carpeta se crea un archivo con extensión “ppr” (proyecto de prometeo) y el nombre dado por el desarrollador para su proyecto (este archivo es un documento XML que sirve para almacenar de manera persistente características de configuración de cada proyecto creado), finalmente el sistema muestra en la GUI principal el nombre del proyecto seguido de la extensión “.ppr”, concluyendo de esta forma el caso de uso. La estructura del archivo ppr es la mostrada en la Figura 3.6.

Figura 3.4. GUI Nuevo

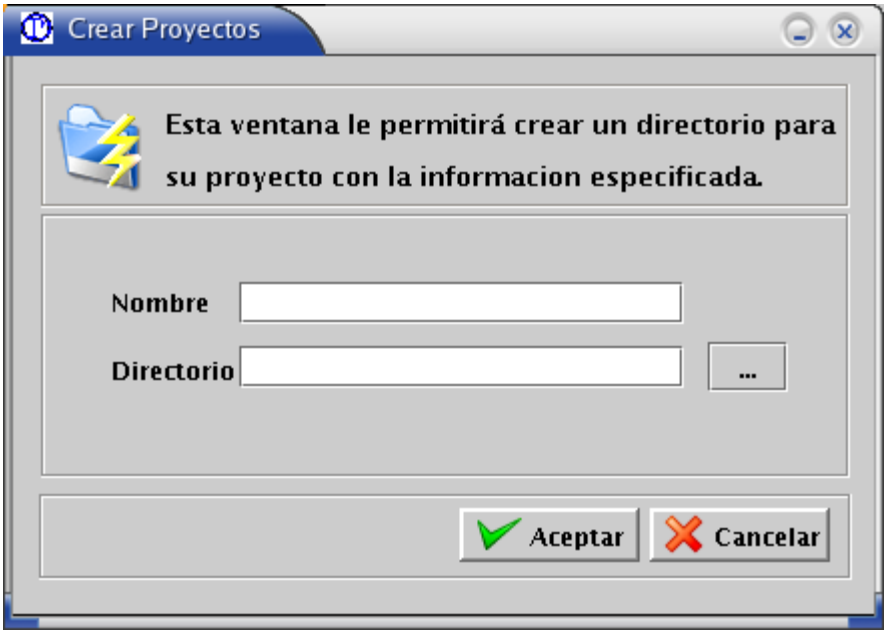

Figura 3.5. GUI Crear Proyectos

Archivo ppr

```
<?xml version="1.0" encoding="UTF-8" ?>
 <proyecto>
     <nombre />
     <path />
     <bibliotecas />
     <emulador />
 </proyecto>
```

Figura 3.6. Estructura de un archivo ppr.

Caso de uso No. 2: Cerrar proyecto

Actores: Desarrollador (iniciador)

Propósito: permitir al desarrollador cerrar el proyecto abierto en el entorno.

Resumen: El desarrollador selecciona la opción "cerrar nombre_proyecto.ppr" y confirma la acción.

Precondiciones:

Debe haber un proyecto abierto en el entorno.

Flujo Principal:

- Este caso de uso se inicia cuando el desarrollador hace clic derecho sobre el nombre de su proyecto en el árbol que muestra la estructura de archivos de su aplicación y selecciona en el menú emergente la opción "cerrar nombre_proyecto.ppr".
- El sistema responde presentando una ventana de confirmación en la que se le pregunta al desarrollador si desea cerrar el proyecto guardando o no los cambios.
- Si el desarrollador selecciona si, el sistema responde guardando todos los archivos de la aplicación y reiniciando todo el entorno, terminando así el caso de uso.
- Si el desarrollador selecciona no, el sistema responde reiniciando todo el entorno, terminando así el caso de uso.

Caso de uso No. 3: Abrir Proyecto

Actores: Desarrollador (iniciador)

Propósito: permitir al desarrollador abrir un proyecto creado para continuar trabajando en el desarrollo de su aplicación.

Resumen: El desarrollador selecciona abrir en la GUI principal y proporciona la dirección del archivo .ppr del proyecto que desea abrir.

Precondiciones:

Debe existir el archivo .ppr del proyecto que el desarrollador desee abrir en el entorno.

Flujo Principal:

- Este caso de uso inicia cuando el desarrollador hace clic sobre el icono “abrir” o selecciona en el menu la opción Archivo / Abrir.
- El sistema responde presentando una GUI en la que el desarrollador puede buscar y seleccionar el archivo ppr del proyecto que desee abrir (Figura 3.7).
- El usuario pulsa el botón “Abrir”.
- El sistema responde mostrando en la GUI principal la estructura de archivos del proyecto abierto, finalizando así el caso de uso.

Caso de uso No. 4: Actualizar Proyecto

Actores: Desarrollador (iniciador)

Propósito: permitir al desarrollador visualizar la estructura de archivos actual de su aplicación cuando haga algún cambio en alguno de los directorios.

Resumen: El desarrollador selecciona “actualizar” en la GUI.

Precondiciones:

- Debe haber un proyecto abierto en el entorno.
- Puede iniciarse desde el caso de uso Eliminar Archivo.

Flujo Principal:

- Este caso de uso se inicia cuando el desarrollador hace clic sobre el icono “actualizar” o selecciona en el menú la opción Proyecto / Actualizar.
- El sistema responde mostrando la estructura de archivos actualizada del proyecto sobre el cual el desarrollador está trabajando, finalizado así el caso de uso.

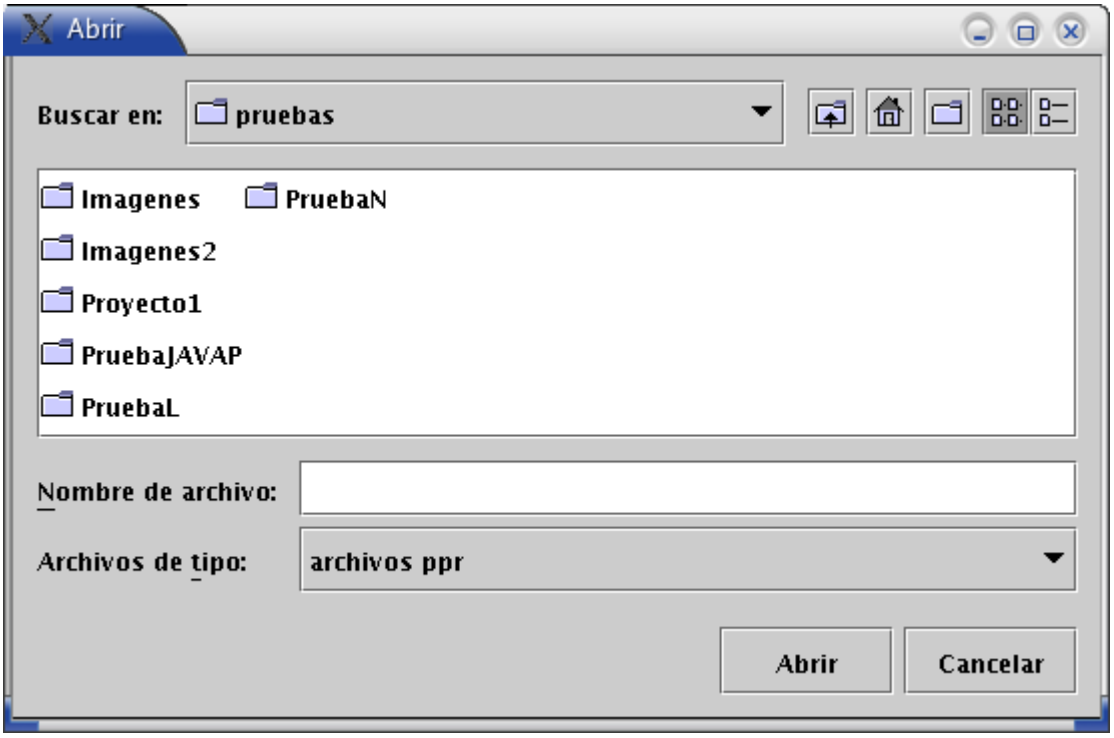

Figura 3.7. GUI Abrir Proyecto

Caso de uso No. 5: Abrir Archivo

Actores: Desarrollador (iniciador)

Propósito: permitir al desarrollador abrir diferentes tipos de archivos en el entorno de desarrollo, particularmente archivos java, xml, txt y wsdl.

Resumen: El desarrollador selecciona el archivo que desea abrir y elige la opción “abrir” o da doble clic sobre el archivo en el árbol que muestra la estructura de archivos de su proyecto.

Precondiciones:

- El archivo que el desarrollador desee abrir en el entorno debe existir y debe tener una de las siguientes extensiones: java, xml, txt o wsdl.
- El archivo debe verse en el árbol que muestra la estructura de archivos.

Flujo principal:

- Este caso de uso se inicia cuando el desarrollador hace doble clic sobre uno de los archivos en el árbol que muestra la estructura de archivos de su proyecto o cuando hace clic derecho sobre uno de estos archivos y en el menú emergente selecciona *Abrir nombre_archivo*.

- El sistema responde mostrando en el editor de código una pestaña con el nombre del archivo abierto y el contenido del mismo, finalizando así el caso de uso.

Caso de uso No. 6: Cerrar Archivo

Actores: Desarrollador (iniciador)

Propósito: permitir al desarrollador finalizar la visualización de un archivo que este abierto en el entorno.

Resumen: el desarrollador selecciona el archivo que desea cerrar y elige la opción “cerrar nombre_archivo”.

Precondiciones:

El archivo a cerrar debe estar abierto en el editor de código.

Flujo principal:

- Este caso de uso se inicia cuando el desarrollador hace clic derecho sobre la pestaña que representa el archivo que desea cerrar.
- El sistema responde mostrando un menú emergente con las opciones de cerrar el archivo seleccionado, cerrar todos los demás archivos abiertos o cerrar todos los archivos.
- Si el desarrollador selecciona “cerrar nombre_archivo” el sistema responde cerrando esa pestaña en el editor, finalizando así el caso de uso.
- Si el desarrollador selecciona “cerrar otros archivos” el sistema responde cerrando todas las pestañas que se encuentren abiertas en el editor, con excepción de la que seleccionó el desarrollador, finalizando de esta forma este caso de uso.
- Si el desarrollador selecciona “cerrar todos los archivos” el sistema responde cerrando todas las pestañas que se encuentren abiertas en el editor, finalizando así el caso de uso.

Caso de uso No. 7: Eliminar Archivo

Actores: Desarrollador (iniciador)

Propósito: brindar al desarrollador la capacidad para eliminar desde el entorno los archivos que no sean requeridos para el desarrollo de su aplicación.

Resumen: El desarrollador selecciona el archivo que desea eliminar, elige la opción *Eliminar* y confirma al sistema que desea eliminar el archivo seleccionado.

Precondiciones:

- Debe estar abierto un proyecto en el entorno.

- El archivo que se desea eliminar debe ser visible en el árbol de la estructura de archivos de la aplicación.

Flujo principal:

- Este caso de uso inicia cuando el desarrollador hace clic derecho sobre el nodo en el árbol que representa al archivo que desea eliminar y en el menú emergente hace clic en la opción “Eliminar nombre_archivo”.
- El sistema responde mostrando una ventana de confirmación en la que se pregunta al desarrollador si en realidad desea eliminar el archivo seleccionado (Figura 3.8).
- Si el desarrollador selecciona si el sistema responde eliminando el archivo seleccionado y actualizando el árbol de la estructura de archivos del proyecto, finalizando así el caso de uso.
- Si el desarrollador selecciona no, el sistema responde ocultando la ventana de confirmación, finalizando así el caso de uso.

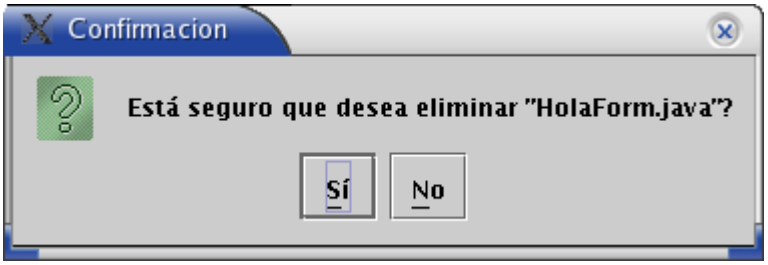

Figura 3.8. GUI confirmar Eliminación

Caso de uso No. 8: Copiar Código

Actores: Desarrollador (iniciador)

Propósito: permitir al desarrollador copiar código y en general texto entre archivos abiertos en el entorno y desde archivos abiertos en el entorno hacia otros archivos ajenos al entorno y viceversa.

Resumen: el desarrollador selecciona el código o texto que desee copiar, elige la opción copiar.

Precondiciones:

Debe haber un archivo abierto en el editor de código del entorno.

Flujo principal:

- Este caso de uso inicia cuando el desarrollador selecciona el código o texto que desea copiar, pulsa clic derecho y en el menú emergente selecciona la opción *copiar* o

cuando el desarrollador selecciona el código o texto que desea copiar y pulsa las teclas CTRL + C. O cuando el desarrollador hace clic en el icono *copiar*.

- El sistema responde almacenando en el clipboard sistema operativo la selección hecha por el usuario finalizando así el caso de uso.

Nota: Como el contenido seleccionado está en el portapapeles del sistema operativo, este contenido puede copiarse a cualquier parte permitida por el sistema operativo.

Caso de uso No. 9: Cortar Código

Actores: Desarrollador (iniciador)

Propósito: permitir al desarrollador cortar código o en general texto desde un archivo abierto en el entorno.

Resumen: el desarrollador selecciona el código o texto que desea cortar y hace clic en la opción cortar.

Precondiciones:

El archivo del que se desee cortar código debe estar abierto en el entorno.

Flujo principal:

- Este caso de uso inicia cuando el desarrollador selecciona el código o texto que desea cortar, hace clic derecho sobre el editor y en el menú emergente selecciona la opción *cortar*. O cuando habiendo seleccionado el código o texto pulsa las teclas CTRL + X o hace clic en el icono *cortar*.
- El sistema responde eliminado el código o texto seleccionado del editor y almacenándolo en el portapapeles del sistema operativo, finalizando así el caso de uso.

Caso de uso No. 10: Pegar Código

Actores: Desarrollador (iniciador)

Propósito: permitir al desarrollador pegar el código o texto que esté almacenado en la memoria del sistema operativo.

Resumen: El desarrollador se ubica en el archivo en el que desee pegar el código o texto y selecciona la opción *pegar*.

Precondiciones:

- Debe haber un archivo abierto en el entorno.
- Debe haber código o texto en el clipboard del sistema operativo.

Flujo principal:

- Este caso de uso inicia cuando el desarrollador hace clic derecho en el editor y en el menú emergente elige la opción *pegar*. O cuando pulsa el icono *cortar* o pulsa las teclas CTRL. + V.
- El sistema responde pegando el código o texto que se encuentra en el clipboard del sistema operativo en la ubicación actual del cursor en el editor, finalizando así el caso de uso.

Caso de uso No. 11: Crear Clase

Actores: Desarrollador (iniciador)

Propósito: brindar al desarrollador un generador de código que le permita crear la plantilla de una nueva clase.

Resumen: El desarrollador selecciona la opción crear clase, proporciona la información necesaria (nombre de la clase, paquete) y si desea la información opcional (autor, extiende de, implementa a) y pulsa aceptar.

Precondiciones:

- Debe haber un proyecto abierto.
- El sistema debe contar con información obligatoria provista por el desarrollador (Nombre de la clase y paquete).

Flujo principal:

- Este caso de uso inicia cuando el desarrollador hace clic sobre el icono Nuevo o escoge la opción Archivo / Nuevo en el menú principal y hace clic sobre el botón Clase (Figura 3.4).
- El sistema responde mostrando una GUI como la de la figura 3.9.
- El desarrollador suministra la información necesaria y pulsa el botón Aceptar.
- El sistema responde creando un archivo .java con el nombre y en el paquete dados por el usuario, el cual es mostrado en el editor de código, finalizando así el caso de uso.

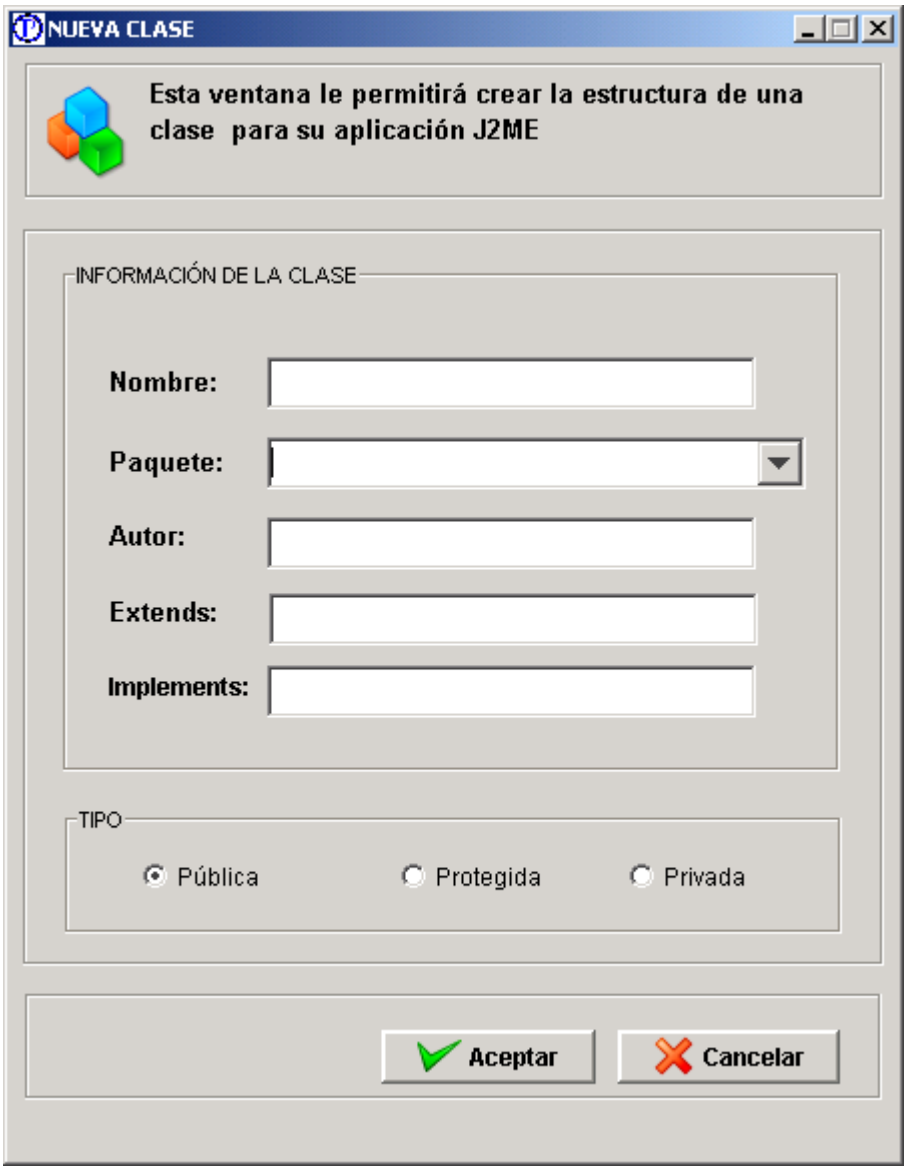

Figura 3.9. GUI Nueva Clase

Caso de uso No. 12: Crear MIDlet

Actores: Desarrollador (iniciador)

Propósito: brindar al desarrollador un generador de código que le permita crear la plantilla de un nuevo MIDlet con los métodos y atributos requeridos para que se pueda ejecutar en un emulador.

Resumen: El desarrollador selecciona la opción crear MIDlet, proporciona la información necesaria y pulsa aceptar.

Precondiciones:

- Debe haber un proyecto abierto.

- El sistema debe contar con información obligatoria provista por el desarrollador (Nombre de la clase y paquete).

Flujo principal:

- Este caso de uso inicia cuando el desarrollador hace clic sobre el icono Nuevo o escoge la opción Archivo / Nuevo en el menú principal y hace clic sobre el botón MIDlet (Figura 3.4).
- El sistema responde mostrando una GUI como la de la figura 3.10.
- El desarrollador suministra la información necesaria y pulsa el botón Aceptar.
- Si el desarrollador marcó la opción crear Desplegable, el sistema responde creando dos archivos .java, uno para el MIDlet y otro para el Desplegable, con los nombres y en el paquete especificado por el desarrollador y mostrando en el editor el archivo correspondiente al Desplegable, finalizando así el caso de uso.
- Si el desarrollador no marcó la opción crear Desplegable, el sistema responde creando un archivo .java para el nuevo MIDlet con el nombre de clase y en el paquete dados por el usuario, dicho archivo es mostrado en el editor de código, finalizando de esta forma este caso de uso.

Caso de uso No. 13: Crear Desplegable

Actores: Desarrollador (iniciador)

Propósito: brindar la posibilidad de utilizar generadores de código que le permitan crear diferentes tipos de clases para presentar las GUI en el dispositivo móvil (Form, Canvas, List, TextBox).

Resumen: El desarrollador selecciona la opción nuevo desplegable, ingresa la información requerida, escoge el tipo de desplegable que desea crear y pulsa el botón aceptar.

Precondiciones:

- Debe haber un proyecto abierto.
- El sistema debe contar con información obligatoria provista por el desarrollador (Nombre de la clase y paquete).

Flujo principal:

- Este caso de uso inicia cuando el desarrollador hace clic sobre el icono Nuevo o escoge la opción Archivo / Nuevo en el menú principal y hace clic sobre el botón Desplegable (Figura 3.4).
- El sistema responde mostrando una GUI como la de la figura 3.11.

- El desarrollador suministra la información necesaria y pulsa el botón Aceptar.
- El sistema responde creando una clase con el nombre, paquete y tipo de desplegable dados por el desarrollador y mostrando en el editor el archivo correspondiente al desplegable creado, finalizando así el caso de uso.

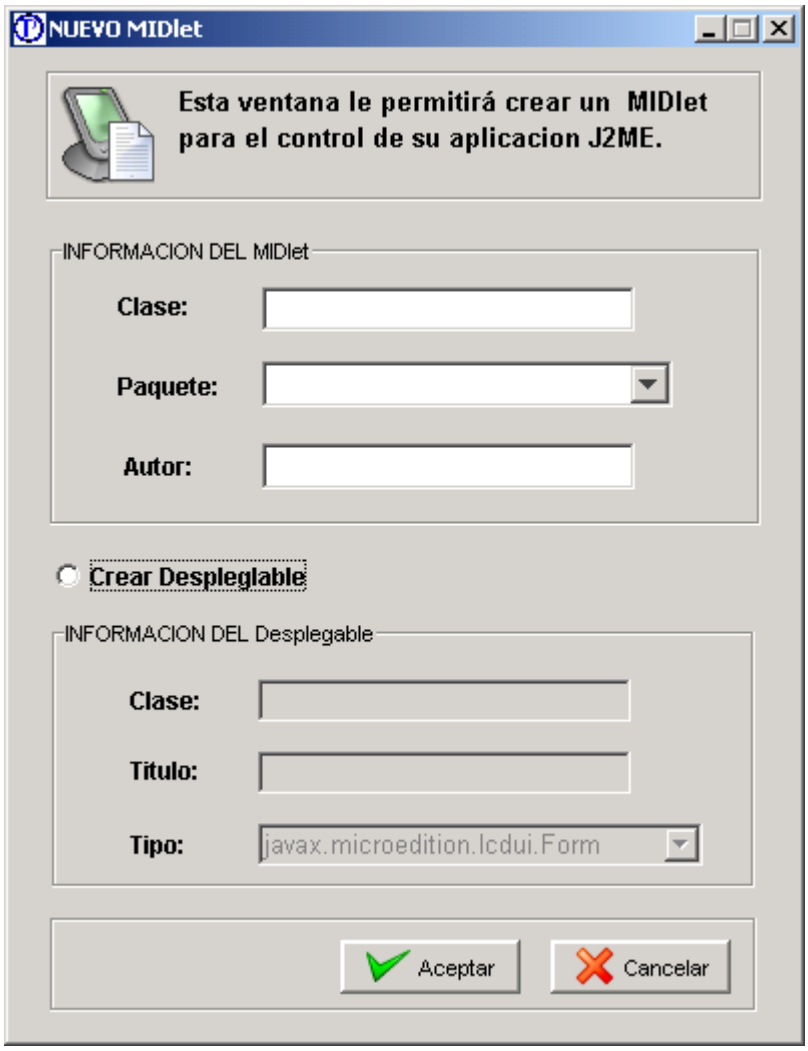

Figura 3.10. GUI Nuevo MIDlet

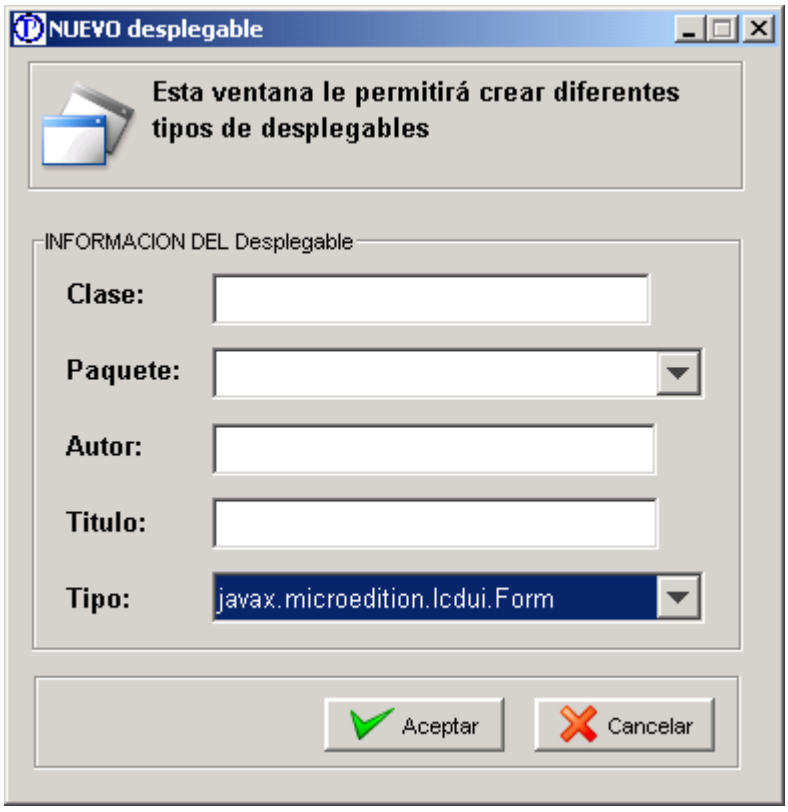

Figura 3.11. GUI Nuevo Desplegable

Nota: Si el desarrollador no selecciona ningún tipo de desplegable, la opción por defecto es un desplegable de tipo javax.microedition.lcdui.Form.

Caso de uso No. 14: Crear Conexión

Actores: Desarrollador (iniciador)

Propósito: Permitir al desarrollador la rápida creación de un par de clases que le permitan comunicarse con una red de datos a través de una conexión HTTP.

Resumen: El desarrollador selecciona la opción nueva conexión, suministra la información requerida y pulsa el botón aceptar.

Precondiciones:

- Debe haber un proyecto abierto.
- El sistema debe contar con información obligatoria provista por el desarrollador (Nombre del Form y paquete).

Flujo principal:

- Este caso de uso inicia cuando el desarrollador hace clic sobre el icono *Nuevo* o escoge la opción *Archivo / Nuevo* en el menú principal y hace clic sobre el botón

Conexión.

- El sistema responde mostrando una GUI como la de la figura 3.12.
- El desarrollador suministra la información necesaria y pulsa el botón Aceptar.
- El sistema responde creando dos archivos .java, el primero una clase con los atributos y métodos necesarios para realizar la conexión HTTP y el segundo un Form con los métodos, atributos e interfaces requeridos para manejar la conexión, mostrando en el editor de código el segundo archivo, finalizando así el caso de uso.

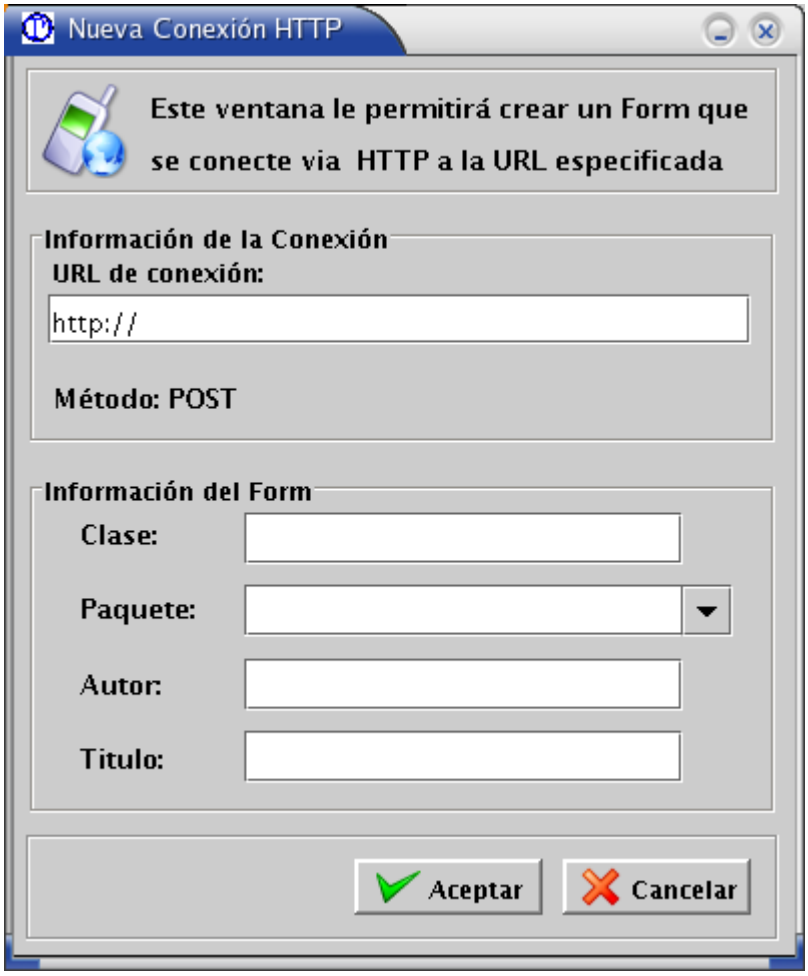

Figura 3.12. GUI Nueva Conexión

Caso de uso No. 15: Crear GUI

Actores: Desarrollador (iniciador)

Propósito: Permitir al desarrollador la implementación de interfaces gráficas de usuario de una manera más fácil y rápida mediante un diseñador gráfico.

Resumen: El desarrollador selecciona en la barra de ítems en el diseñador grafico(figura 3.13) aquel que desea adicionar a su GUI y lo arrastra hasta donde desea ubicarlo en la pantalla del emulador.

Precondiciones:

- Debe estar abierto un proyecto.
- Debe estar abierto el editor gráfico.

Flujo principal:

- Este caso de uso se inicia cuando el desarrollador selecciona la pestaña Diseño en la interfaz principal.
- El sistema responde mostrando una GUI como la de la figura 3.13.
- El Desarrollador hace clic sobre el ítem que desea ubicar en la GUI.
- El sistema responde fijando el Java Bean que se dibujará al hacer clic sobre el área de dibujo.
- El desarrollador hace clic sobre el área de dibujo
- El sistema responde mostrando en el área de dibujo el ítem seleccionado y asigna un conjunto de propiedades por defecto al java Bean que representa el ítem dibujado
- El desarrollador hace clic sobre la pestaña fuente
- El sistema responde adicionando el código correspondiente al archivo java modificado y recargando en el editor dicho archivo, finalizando así el caso de uso.

Caso de uso No. 16: Compilar Aplicación

Actores: Desarrollador (iniciador)

Propósito: permitir al desarrollador la compilación de las aplicaciones creadas en el entorno.

Resumen: El desarrollador selecciona la opción compilar.

Precondiciones:

- Debe estar instalado un WTK versión 1.4 o posterior.
- Debe haber un proyecto abierto en el entorno.

Flujo principal:

- Este caso de uso inicia cuando el desarrollador hace clic sobre el botón Compilar o selecciona Proyecto / Compilar proyecto en el menú principal.
- El sistema responde mostrando una barra de progreso que indica el estado de la compilación por paquetes, permitiendo visualizar el nombre del archivo que está siendo compilado.

- Si no hay errores en la compilación el sistema muestra el mensaje →Compilación completa, finalizando así el caso de uso.
- Si hay errores en la compilación el sistema muestra los errores producidos y al final el mensaje →Compilación completa, finalizando así el caso de uso.

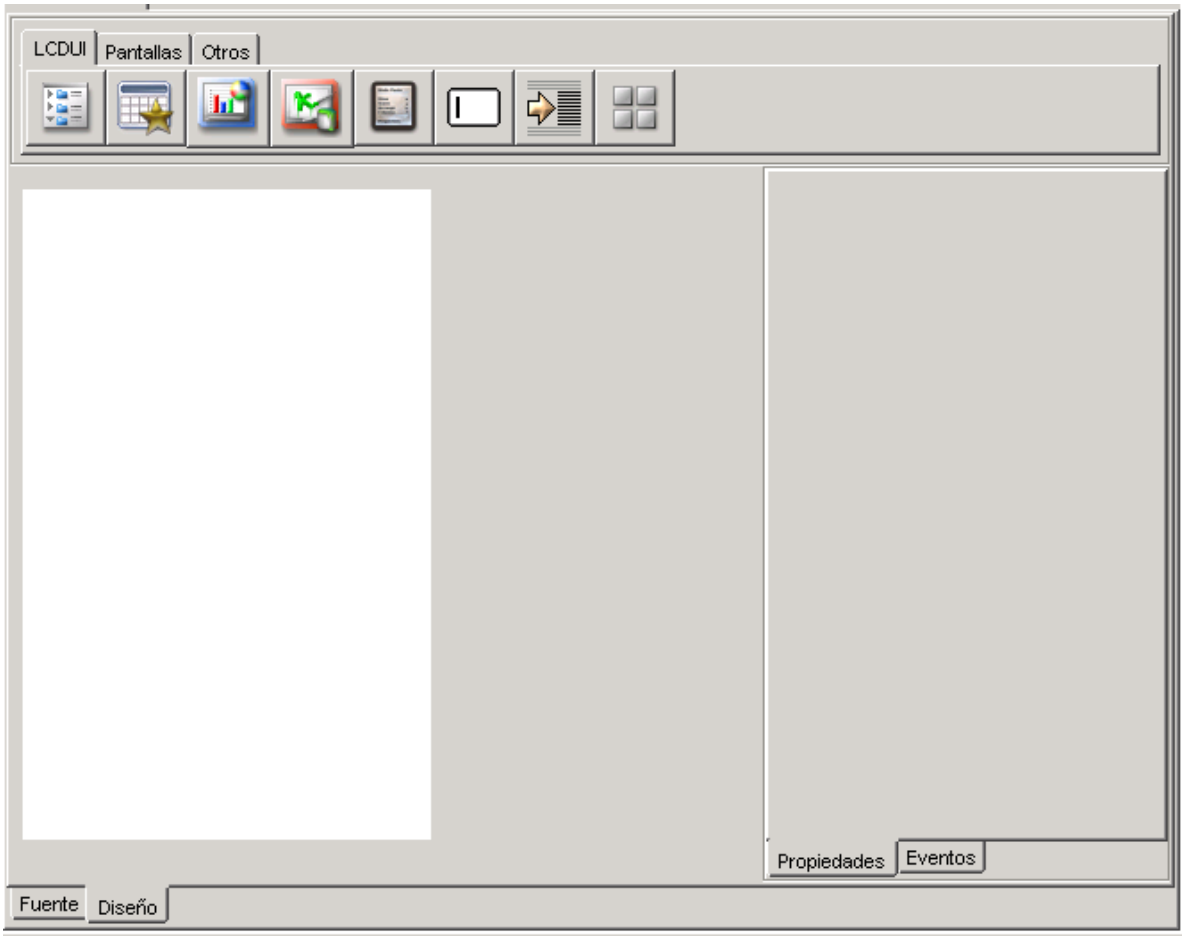

Figura 3.13. Editor gráfico

Caso de uso No. 17: Crear JAR

Actores: Desarrollador (iniciador)

Propósito: Crear el archivo que contiene todos los recursos de la aplicación.

Resumen: el desarrollador selecciona la opción *Crear JAR* y en la GUI emergente pulsa el botón *Aceptar.*

Precondiciones:

- No debe haber errores de compilación
- Debe haber un proyecto abierto.

Flujo principal:

- Este caso de uso inicia cuando el desarrollador selecciona Herramientas / Crear JAR,
- El sistema responde mostrando una GUI como la de la figura 3.14.
- El desarrollador pulsa el botón Aceptar.
- El sistema responde llamando al caso de uso Crear Manifiesto.
- Si el manifiesto es creado correctamente, el sistema crea un archivo nombre_proyecto.jar que contiene todas las clases (código objeto) de la aplicación desarrollada y todos los recursos (imágenes y otros archivos) que el desarrollador haya puesto en su proyecto.
- Si el archivo JAR fue creado correctamente, el sistema llama al caso de uso Crear JAD, finalizando así el caso de uso.

Caso de uso No. 18: Generar Manifiesto

Actores: Caso de uso Crear JAR (iniciador).

Propósito: crear el archivo manifiesto que servirá para crear el archivo JAR que contendrá todos los recursos de la aplicación desarrollada.

Resumen: el desarrollador selecciona en el menú la opción *Crear* Jar, modifica o no el archivo manifiesto y pulsa el botón *Aceptar*.

Precondiciones:

- No debe haber errores de compilación en la aplicación.
- Debe haber un proyecto abierto

Flujo principal:

- Este caso de uso inicia cuando el desarrollador selecciona en el menú Herramientas / Crear JAR.
- El sistema responde mostrando una GUI como la de la figura 3.14.
- Si el desarrollador modifica los valores del archivo manifiesto, el sistema responde creando un archivo de nombre MANIFIESTO.MF (Figura 3.15) en la carpeta bin dentro de la carpeta del proyecto con los valores ingresados por el desarollador finalizado así el caso de uso.
- Si el desarrollador no modifica los valores del archivo manifiesto, el sistema responde creando un archivo de nombre MANIFIESTO.MF (Figura 3.15) en la carpeta bin dentro de la carpeta del proyecto con los valores por defecto, finalizado así el caso de uso.

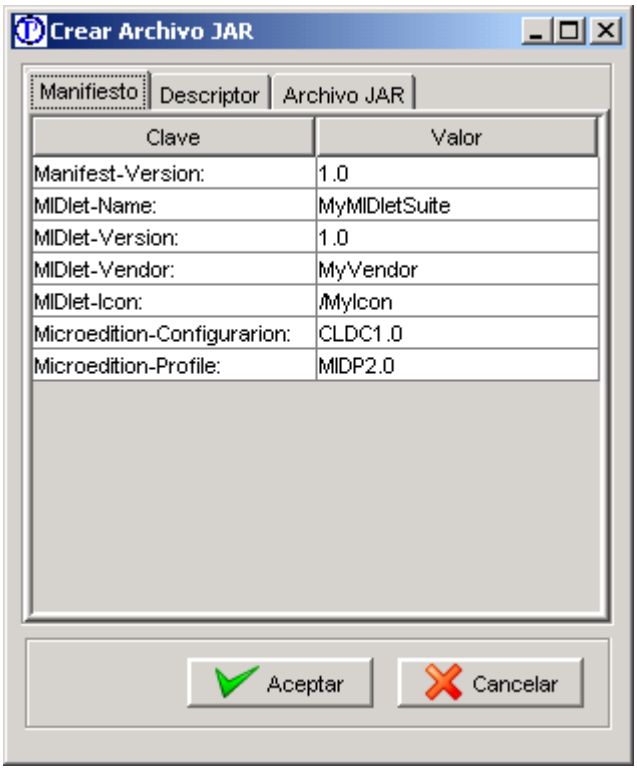

Figura 3.14. GUI Crear Archivo JAR

Archivo MANIFIESTO.MF
Manifest-Version: 1.0
MIDlet-Version: 1.0
MIDlet-1: Nombre_MIDlet_1, , paquete.ClaseMIDlet_1
:
MIDlet-n: Nombre_MIDlet_n, , paquete.Clase_MIDlet_n
MIDlet-Name: MiMIDletSuite
MIDlet-Vendor: My Vendor
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-2.0

Figura 3.15. Estructura del archivo MANIFIESTO.MF

Caso de uso No. 19: Generar JAD

Actores: Caso de uso Crear JAR

Propósito: crear el archivo descriptor para la aplicación desarrollada.

Resumen: el desarrollador selecciona en el menú la opción *Crear* Jar, modifica o no el archivo Manifiesto y pulsa el botón *Aceptar*.

Precondiciones:

- Debe existir el archivo JAR.
- Debe haber un proyecto abierto.

Flujo principal:

- Este caso de uso se inicia cuando en el desarrollado selecciona la opción *Herramientas / Crear JAR*,
- El sistema responde mostrando una GUI como la de la figura 3.14.
- Si el desarrollador modifica los valores del archivo JAD, el sistema responde creando un archivo nombre_proyecto.jad (Figura 3.16) en una carpeta bin dentro de la carpeta del proyecto, con los valores ingresados por el desarrollador, finalizando así el caso de uso.
- Si el desarrollador no modifica los valores del archivo JAD, el sistema responde creando un archivo nombre_proyecto.jad (Figura 3.16) en una carpeta bin dentro de la carpeta del proyecto, con los valores por defecto, finalizando así el caso de uso.

Archivo .jad
MIDlet-1: nombre_MIDlet_1, ,paquete.ClaseMIDlet_1
:
MIDlet-n: nombre_MIDlet_n, ,paquete.ClaseMIDlet_n
MIDlet-Name: MiMIDletSuite
MIDlet-Vendor: My Vendor
MIDlet-Version: 1.0
MIDlet-Jar-Size: tamaño_archivo_JAR
MIDlet-Jar-URL: ubicación_archivo_JAR

Figura 3.16. Estructura del archivo .jad

Caso de uso No. 20: Emular Aplicación

Actores: Desarrollador (iniciador)

Propósito: permitir al desarrollador observar el comportamiento de la aplicación desarrollada en un emulador.

Resumen: El desarrollador selecciona la opción “ejecutar”

Precondiciones:

- La aplicación no debe presentar errores de compilación.
- Debe existir el archivo .jad.

Flujo principal:

- Este caso de uso se inicia cuando el desarrollador hace clic sobre el botón Ejecutar o escoge las opciones Ejecutar / Ejecutar en el menú principal.
- El sistema responde mostrando el emulador que haya sido configurado para el proyecto y en él, el nombre de todos los MIDlets que contenga la aplicación.
- El desarrollador selecciona y ejecuta el MIDlet deseado.
- El desarrollador cierra el emulador.
- El sistema responde mostrando las características de ejecución de la aplicación en el panel de mensajes de la plataforma, finalizando así este caso de uso.

Caso de uso No. 21: Adicionar Bibliotecas

Actores: Desarrollador (iniciador)

Propósito: permitir al desarrollador adicionar bibliotecas de clases que puedan ser utilizadas en la plataforma de desarrollo.

Resumen: El desarrollador selecciona la opción C*onfigurar Bibliotecas*, pulsa el botón *Nueva*, ingresa el nombre, selecciona los paths de la nueva bibliotecas y pulsa el botón *Aceptar* en las ventanas emergentes.

Precondiciones:

Ninguna.

Flujo principal:

- Este caso de uso inicia cuando el desarrollador escoge la opción Herramientas / Configurar Bibliotecas y en la ventana emergente (Figura 3.18) pulsa el botón Nueva.
- El sistema responde mostrando una ventana en donde se pide al desarrollador el nombre de la nueva biblioteca y los paths de las clases que la conformarán (Figura 3.19).
- El desarrollador ingresa la información necesaria y pulsa la el botón Aceptar.
- El sistema responde mostrando las bibliotecas configuradas en el entorno.
- El desarrollador pulsa el botón Aceptar.
- El sistema responde grabando en el archivo de configuración de la plataforma (Figura 3.23) el nombre de la nueva biblioteca y los paths de las clases que la componen, finalizando así este caso de uso.

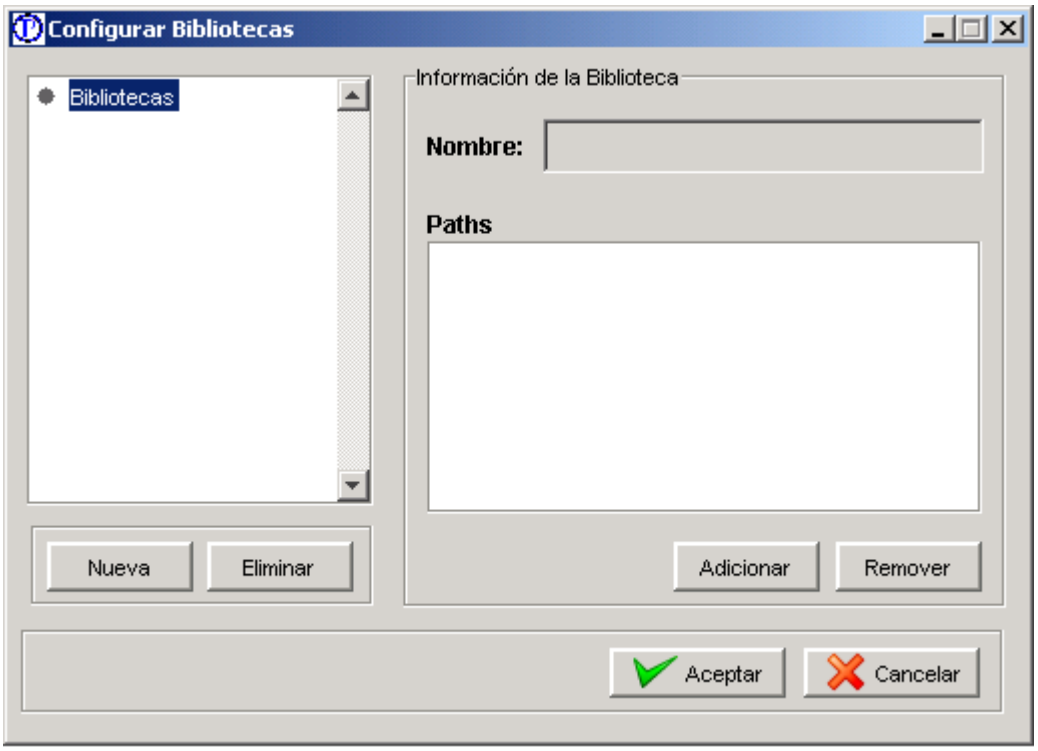

Figura 3.17. GUI configurar Bibliotecas

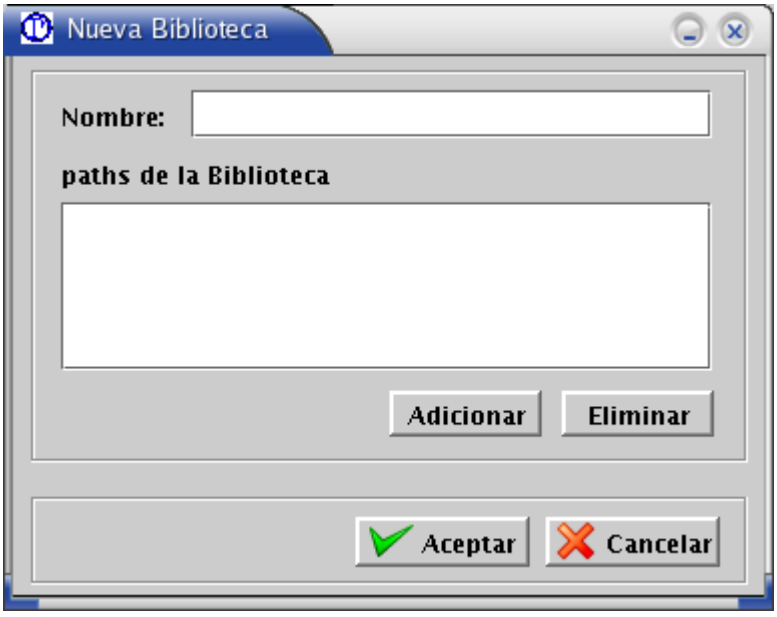

Figura 3.18. GUI Nueva Biblioteca

Caso de uso No. 22: Modificar Bibliotecas

Actores: Desarrollador (iniciador)

Propósito: permitir al desarrollador modificar los paths de las bibliotecas de clases que

estén configuradas en la plataforma de desarrollo.

Resumen: el desarrollador selecciona la opción *Herramientas / Configurar Bibliotecas*, selecciona la biblioteca que desea modificar, adiciona o elimina rutas a esa biblioteca y pulsa *Aceptar*.

Precondiciones:

Deben existir bibliotecas configuradas en la plataforma de desarrollo.

Flujo principal:

- Este caso de uso inicia cuando el desarrollador escoge la opción *Herramientas / Configurar Bibliotecas*.
- El sistema responde mostrando una ventana como la de la figura 3.17.
- El desarrollador selecciona en el árbol de bibliotecas aquella que desea modificar.
- El sistema responde mostrando el nombre de la biblioteca y las rutas de las clases que la conforman.
- Si el desarrollador pulsa el botón Adicionar (figura 3.17) el sistema responde mostrando una ventana en la que el desarrollador puede seleccionar la nueva ruta que hará parte de la biblioteca.
- Si el desarrollador selecciona una de las rutas que conforman la biblioteca y pulsa el botón Remover (figura 3.17), el sistema responde borrando de la interfaz gráfica la ruta seleccionada.
- El desarrollador pulsa el botón Aceptar.
- El sistema responde guardando en el archivo de configuración de la plataforma (archivo config.xml, figura 3.23) los cambios hechos a la biblioteca seleccionada, finalizando así este caso de uso.

Caso de uso No. 23: Eliminar Bibliotecas

Actores: Desarrollador (iniciador)

Propósito: permitir al desarrollador eliminar las bibliotecas de clases que haya adicionado a la plataforma de desarrollo.

Resumen: El desarrollador elige la opción *Herramientas / Configurar Bibliotecas*, selecciona en el árbol de bibliotecas aquella que desee eliminar y pulsa el botón *Eliminar*.

Precondiciones:

Deben existir bibliotecas configuradas en la plataforma de desarrollo.

Flujo principal:

- Este caso de uso inicia cuando el desarrollador escoge la opción *Herramientas / Configurar Bibliotecas*.
- El sistema responde mostrando una ventana como la de la figura 3.17.
- El desarrollador selecciona en el árbol de bibliotecas aquella que desea eliminar y pulsa el botón *Eliminar*.
- El sistema responde borrando la biblioteca seleccionada del archivo de configuración de la plataforma, finalizando así este caso de uso.

Caso de uso No. 24: Adicionar Emulador

Actores: Desarrollador (iniciador).

propósito: permitir al desarrollador la utilización no solo de los emuladores que vienen con el WTK de Sun Microsystems, sino también la utilización de emuladores de otras empresas que sean compatibles con este.

Resumen: El desarrollador ubica el nuevo emulador en el directorio *wtklib / devices* del WTK utilizado.

Precondiciones:

El nuevo emulador debe ser compatible con el WTK utilizado.

Flujo principal:

El desarrollador ubica el nuevo emulador en el directorio *wtklib / devices* del WTK utilizado.

Caso de uso No. 25: Seleccionar Emulador

Actores: Desarrollador (iniciador).

Propósito: permitir al desarrollador utilizar cualquiera de los emuladores disponibles en el WTK utilizado, para observar la ejecución de su aplicación.

Resumen: El desarrollador selecciona en el menú la opción *Herramientas / Configurar Emulador*, selecciona el emulador que desea utilizar y pulsa el botón *Aceptar*.

Precondiciones:

- Debe haber un proyecto abierto en el entorno.
- Debe haber un WTK 1.4 o posterior instalado

Flujo principal:

- Este caso de uso se inicia cuando el desarrollador selecciona en el menú la opción herramientas / Configurar Emulador.
- El sistema responde mostrando una GUI como la de la figura 3.19.

- El desarrollador selecciona de la lista de emuladores aquel que desea utilizar y pulsa el botón Aceptar.
- El sistema responde guardando en el archivo de configuración del proyecto (figura 3.6) el dispositivo que se utilizará como emulador, finalizando de esta forma el caso de uso.

Figura 3.19. GUI Seleccionar Emulador

Caso de uso No. 26: Configurar WTK

Actores: Desarrollador (iniciador)

Propósito: permitir al desarrollador escoger el WTK que desea utilizar en su aplicación de aquellos que tenga instalados en su sistema.

Resumen: el desarrollador elige la opción *Herramientas / Configurar WTK*, ingresa la ruta del directorio raíz del WTK que desea utilizar y pulsa *Aceptar*.

Precondiciones:

- Debe haber un proyecto abierto en el entorno.
- Debe haber un WTK 1.4 o posterior instalado

Flujo principal:

- Este caso de uso inicia cuando el desarrollador selecciona en el menú principal la opción Herramientas / Configurar WTK.
- El sistema responde desplegando una ventana en la que se le solicita al desarrollador la ruta del directorio raíz del WTK que desea utilizar para su aplicación (Figura 3.20).
- El usuario ingresa la ruta del WTK y pulsa la el botón Aceptar.

- El sistema responde guardando en el archivo de configuración del proyecto (Figura 3.6) la ruta del WTK configurado, finalizando así el caso de uso.

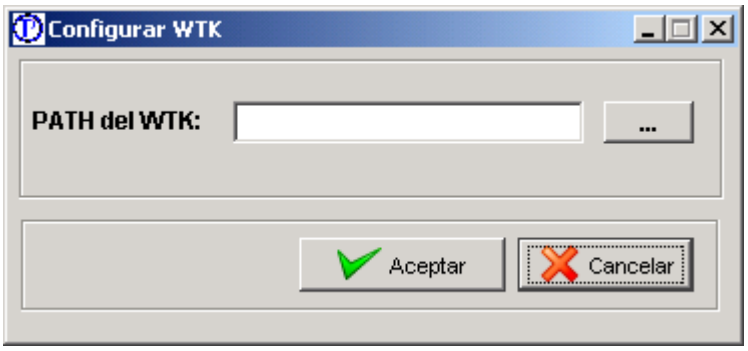

Figura 3.20. GUI Configurar WTK

Caso de uso No. 27: Obtener ayuda del Entorno

Actores: Desarrollador (iniciador)

Propósito: Permitir que el desarrollador obtenga ayuda acerca de cómo realizar alguna operación dentro de la plataforma de desarrollo.

Resumen: el desarrollador elige la opción *Ayuda / Ayuda del IDE* y en la ventana emergente selecciona el tema relacionado con la operación que desea realizar.

Precondiciones:

Ninguna.

Flujo principal:

- Este caso de uso se inicia cuando el desarrollador selecciona en el menú la opción *Ayuda / Ayuda del IDE.*
- El sistema responde mostrando una GUI como la de la figura 3.21.
- El desarrollador busca los temas relacionados con la ayuda que necesita.
- El desarrollador cierra la ventana finalizando así el caso de uso.

Caso de uso No. 28: Obtener ayuda de J2ME

Propósito: Permitir que el desarrollador obtenga ayuda acerca de las APIs de J2ME.

Resumen: el desarrollador elige la opción *Ayuda / Ayuda de J2ME* y en la ventana emergente selecciona el tema relacionado con la ayuda que necesita.

Precondiciones:

Debe haber un WTK instalado.

Flujo principal:

- Este caso de uso se inicia cuando el desarrollador selecciona en el menú la opción Ayuda / Ayuda del IDE.
- El sistema responde abriendo en un explorador (Internet Explorer para Windows o Mozilla para Linux) y desplegando la pagina Index.html de la documentación del WTK utilizado, finalizando así el caso de uso.

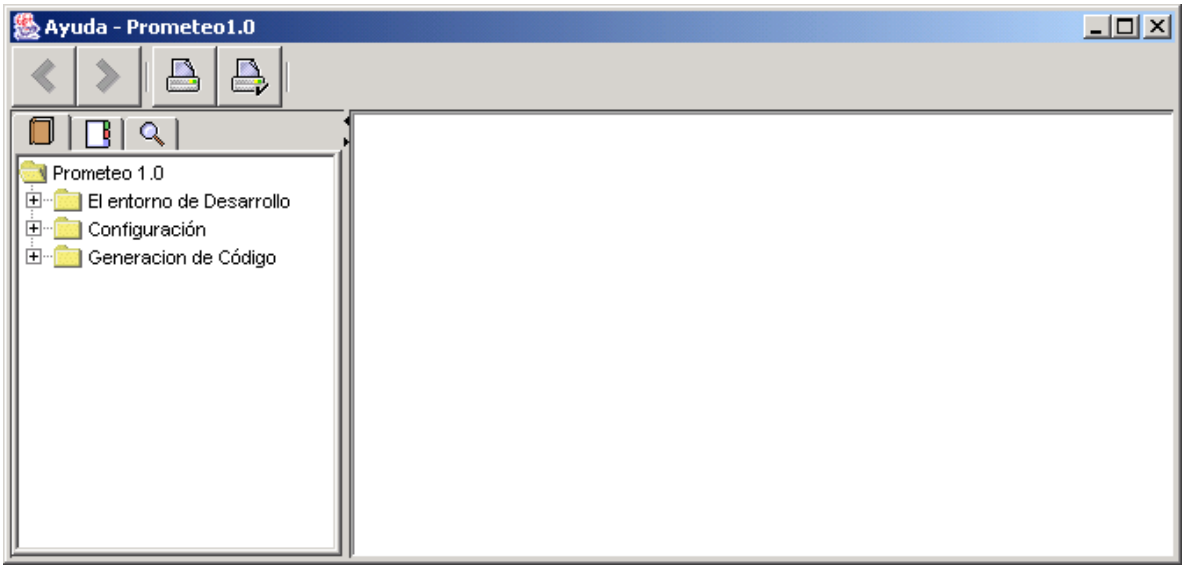

Figura 3.21. GUI Ayuda del IDE

Nota: Para una descripción más detallada de los casos de uso ver Anexo B (Diseño detallado).

C. Arquitectura propuesta

En esta etapa del proceso de desarrollo, se planteó una arquitectura que permitiera dar soporte a los casos de uso descritos. Esta arquitectura está basada en el paradigma vista-control-modelo, pero teniendo en cuenta que se deberían generar componentes software, que se necesitarían APIs externas a las del nucleo de java y se necesitarían archivos de configuración. Con estas consideraciones, la arquitectura propuesta es la mostrada en la figura 3.22.

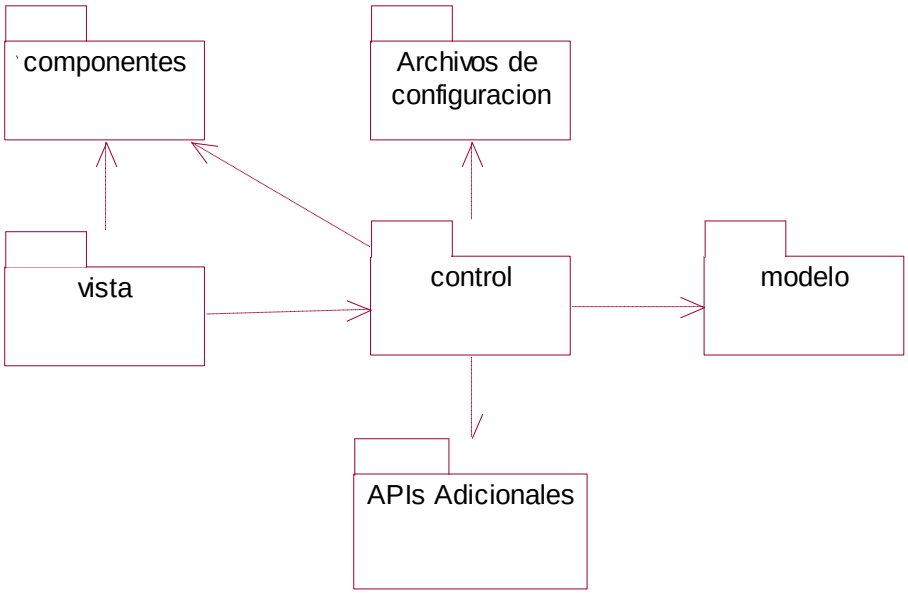

Figura 3.22. Arquitectura de diseño

La explicación de la arquitectura de diseño planteada se obvia dado que en la siguiente sección se hace la descripción de la arquitectura finalmente implementada. De igual forma, el diagrama de clases y la descripción de cada una de ellas se hará en la siguiente sección .

3.2.3. Implementación

En la Figura 3.23 se muestra una arquitectura con base en los paquetes que componen Prometeo. Es preciso anotar que no se tuvieron en cuenta los paquetes propios del nucleo del java 2 SDK. Se ha hecho una división de la arquitectura en cuatro niveles. En el primer nivel se encuentran los paquetes que corresponden a APIs que no se encuentran en el nucleo del J2SDK y que fue necesario incluirlos a manera de bibliotecas. En el segundo nivel se encuentra un paquete que ya existía (JEditSyntax) pero al cual se le hicieron ciertas modificaciones. En el tercer nivel están los paquetes que contienen las clases que fueron implementadas por completo y finalmente, en el cuarto nivel están los paquetes que se utilizan para guardar cierto tipo de archivos (principalmente XML y HTML).

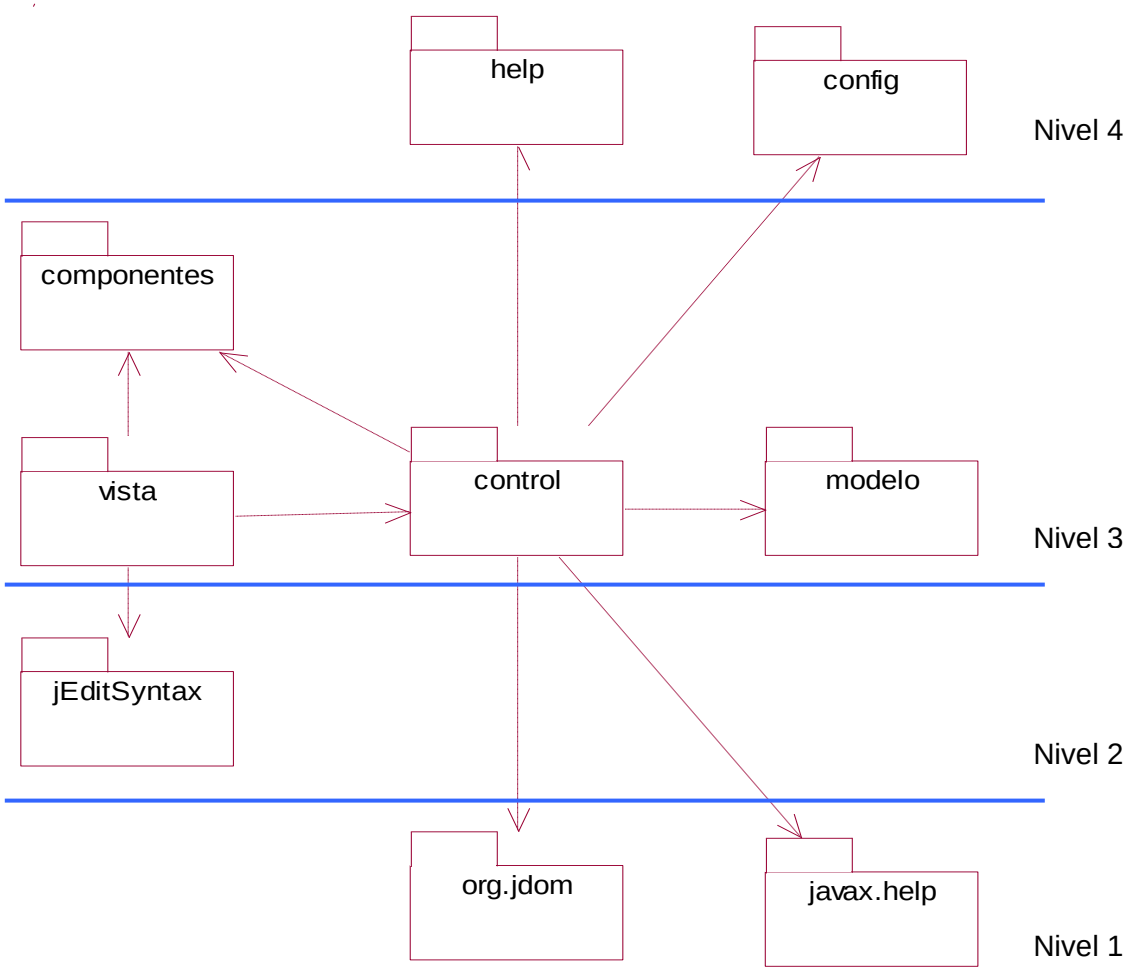

Figura 3.23. Arquitectura de paquetes de Prometeo.

Descripción de paquetes

config: es un paquete que hasta el momento (Prometeo versión 1.0) se utiliza para guardar un archivo especifico, el archivo config.xml, el cual es utilizado para almacenar de manera persistente ciertas características propias del entorno de desarrollo. El archivo config.xml presenta la siguiente estructura:

help: es un paquete que se utiliza para guardar archivos XML que configuran la ayuda de la plataforma de desarrollo y las páginas html que serán mostradas al desplegar esta ayuda.

Archivo config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<prometeo>
    <Bibliotecas />
    <ultimoproyecto />
</prometeo>
```

Figura 3.24. Estructura del Archivo config.xml

componentes: en este paquete se encuentran los Java Beans y las clases que proporcionan métodos para realizar operaciones que se podrían considerar como comunes a cualquier entorno de desarrollo, tales como compilación, copiado de archivos, etc. Así como las operaciones propias de una plataforma de desarrollo para J2ME tales como preverificación, emulación, etc. Adicionalmente en este paquete se encuentran los componentes que permiten al desarrollador utilizar el diseñador de interfaces graficas para su aplicación.

control: en este paquete se ubican las clases que sirven para hacer el control de la lógica de la plataforma de desarrollo y determinar cuando deben realizarse operaciones específicas de acuerdo con las peticiones y parámetros dados por el desarrollador. En el paquete control están las clases que permiten la manipulación de los archivos XML necesarios para almacenar persistentemente información relevante del entorno de desarrollo o de un proyecto en particular, estas clases se encargan de crear los archivos XML, modificarlos y extraer contenido de ellos haciendo el procesamiento necesario para recuperar los datos en formatos específicos. De igual forma, están las clases que proveen los métodos y atributos necesarios para la gestión de archivos y directorios (creación, modificación y eliminación).

modelo: en este paquete se encuentran las clases que contienen información que regularmente no cambia, específicamente en esta arquitectura, clases que definen la estructura del código que se genera a través de los wizards.

vista: en este paquete se encuentran las clases que sirven para construir interfaces gráficas de usuario, estás clases poseen características generales que las hacen útiles para heredar de ellas y crear más fácilmente las GUIs en mención. Internamente a este paquete se tienen otros dos paquetes:

vista.configuración: en el cual están las clases que muestran interfaces gráficas de usuario que permiten al desarrollador configurar ciertas características, tanto del entorno de desarrollo como de su proyecto particular(Bibliotecas, emuladores, etc).

vista.wizards: en el cual se encuentran las clases que muestran interfaces graficas de usuario que permiten introducir parámetros para la generación automática de código.

jEditSyntax: Este paquete es utilizado para contener las clases que permiten manejar el editor de código de la plataforma de desarrollo.

org.jdom: es el paquete que contiene las clases del API Jdom, el cual es utilizado para el trabajo con documentos XML(para mayor detalle ver sección 3.3.2).

javax.help: es el paquete que contiene las clases del API help, el cual es utilizado para menejar la ayuda de la plataforma de desarrollo.(para mayor detalle ver sección 3.3.3).

3.2.4. Pruebas y validación

Las pruebas realizadas a la plataforma desarrollo están detalladas en el anexo D (Pruebas de integración). A continuación se describe la aplicación desarrollada con el objetivo de validar la plataforma construida.

A. Generalidades

Para la validación de la plataforma desarrollada, se implementó una aplicación que permite ingresar datos desde el dispositivo móvil, almacenarlos localmente y a través de una conexión HTTP enviarlos hacia un servidor de aplicaciones para actualizar una base de datos. Se utilizó una aplicación de este tipo ya que permite utilizar la mayoría de las funcionalidades que provee la plataforma de desarrollo.

B. Arquitectura

La figura 3.25 muestra la arquitectura de la aplicación desarrollada con el objetivo de validar PROMETEO.

Figura 3.25. Arquitectura de la aplicación de validación

La parte del desarrollo J2ME (Dispositivo móvil) se realizó enteramente sobre PROMETEO utilizando la versión 2.2 Beta del WTK. La parte del servidor de aplicaciones se desarrollo utilizando el entorno de desarrollo Eclipse 3.0 y el servidor de aplicaciones Apache Tomcat 5.0. y el motor de bases de datos utilizado fue Firebird 1.5.

C. Funcionalidad

Para el desarrollo de la aplicación se utilizaron particularmente los wizards “Nuevo MIDlet”, para la creación de la clase que controlaría el ciclo de vida de la aplicación, “Nuevo Desplegable” y el editor gráfico para construir las interfaces gráficas de usuario requeridas, el wizard “Nuevo Record Store” para el almacenamiento persistente en el dispositivo móvil y el wizard “Nueva Conexión Http” para el envío de los datos hacia el servidor de aplicaciones.

La funcionalidad de la aplicación de validación se muestra a partir de un modelo de casos de uso. La figura 3.26 muestra el diagrama de casos de uso de la aplicación de validación desarrollada y a continuación se describen cada uno de los casos de uso:

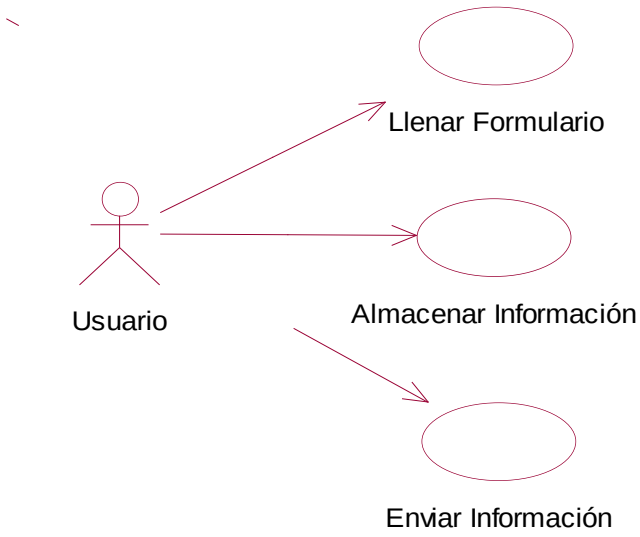

Figura 3.26. Diagrama de casos de uso de la aplicación de validación

Caso de No. 1: Llenar Formulario

Este caso de uso se inicia cuando el usuario de la aplicación selecciona *Ingresar Datos* en la pantalla inicial de la aplicación (Figura 3.27) y pulsa el botón *OK*. La aplicación responde mostrando una interfaz gráfica como la mostrada en la figura 3.28. El usuario llena la información en el formulario desplegado finalizando así el caso de uso.

Caso de uso No.2: Almacenar Información:

Este caso de uso se inicia cuando el usuario pulsa el botón ***Guardar*** en la pantalla que muestra el formulario para el ingreso de datos(Figura 3.28). Si los datos se guardan correctamente la aplicación muestra un mensaje confirmando la acción (Figura 3.29), en caso contrario, la aplicación muestra un mensaje de error(Figura 3.30).

Caso de uso No. 3: Enviar información

Este caso de uso se inicia cuando el usuario selecciona *Enviar Datos* en la pantalla inicial de la aplicación (Figura 3.26) y pulsa el botón *OK*, el sistema responde enviando al servidor de aplicaciones la información que no haya sido enviada aún desde el móvil. Si la información se envió correctamente, la aplicación mostrará un mensaje de confirmación, en caso contrario la aplicación mostrará un mensaje de error.

Figura 3.27. Pantalla inicial aplicación de validación

Figura 3.28. GUI almacenar datos

Figura 3.29. mensaje de confirmación

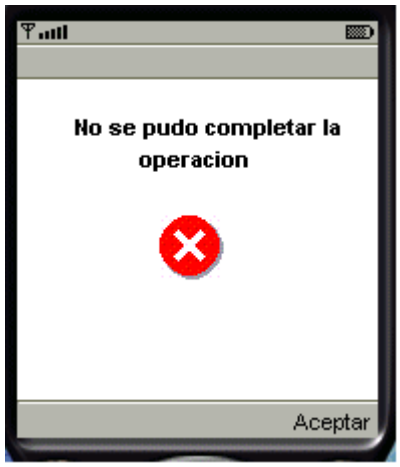

Figura 3.30. mensaje de error

3.3 Arquitectura Funcional de Prometeo

La figura 3.31 muestra la arquitectura funcional de Prometeo, a continuación se hace una descripción de las distintas capas y de las tecnologías que se emplearon en cada una de ellas.

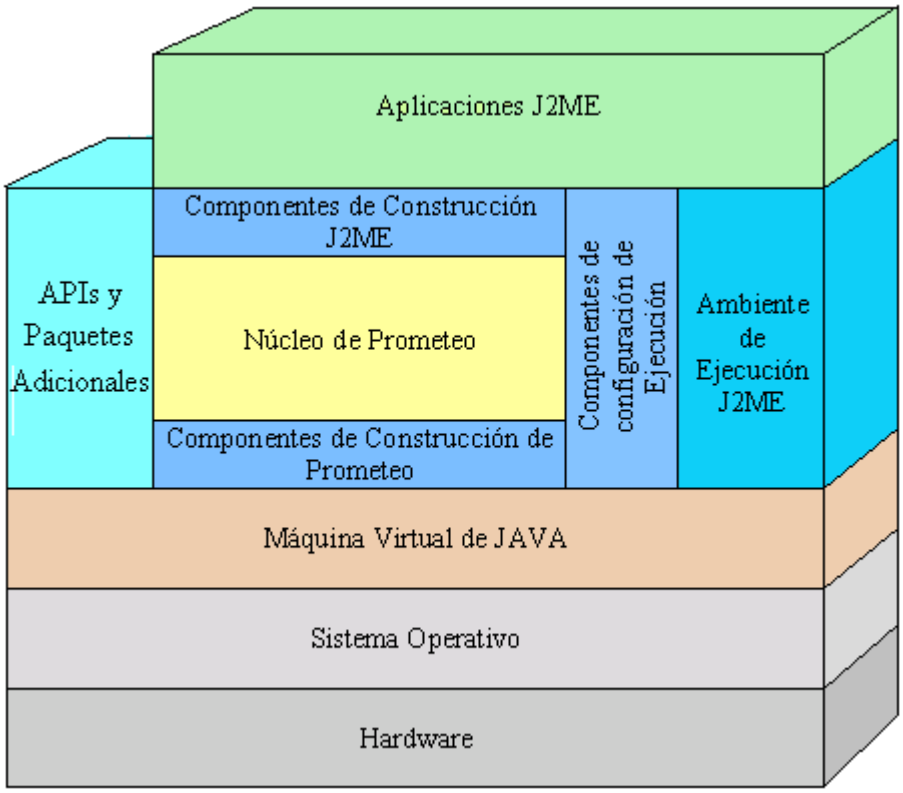

Figura 3.31. Arquitectura funcional de Prometeo

3.3.1. Sistema Operativo

En el desarrollo de Prometeo se utilizaron sistemas operativos de la plataforma MS Windows y de la plataforma GNU/Linux, pero particularmente en la fase de implementación, los sistemas operativos utilizados fueron:

- Windows 2000 Professional
- Windows XP
- Linux Mandrake 9.1
- Linux Slackware 10

3.3.2. Máquina Virtual de JAVA

La máquina virtual de java utilizada, corresponde a la del JAVA 2 SDK (Software Development Kit), el cual es la tercera versión importante del lenguaje de programación JAVA, en el que se desarrollo la plataforma, pensando en características como portabilidad, potencialidad de un lenguaje orientado a objetos y la capacidades de las APIs ya desarrolladas. La máquinas virtuales utilizadas específicamente corresponden a las versiones 1.4, 1.4.1_01 y 1.4.2_04

3.3.3. APIs y paquetes adicionales

A. JDOM

Es una API construida enteramente en java, que fue adicionada a la plataforma de desarrollo a manera de biblioteca. Permite crear una representación en objetos java de un documento XML, proporciona las clases y métodos necesarios para la creación, manipulación y serialización de documentos XML de una manera eficiente.

JDOM es un conjunto de utilidades Java, exclusivo para trabajar con XML, diseñado para permitir un rápido desarrollo de las aplicaciones XML. Su diseño abarca el lenguaje Java desde la sintaxis hasta la semántica, entre sus principales características se citan las siguientes:

- **JDOM es específico de la plataforma Java.** La API utiliza el soporte String incorporado del lenguaje Java siempre que ello es posible para que los valores de texto siempre estén disponibles como String. También hace uso de las clases de colección de la plataforma Java 2, como List e Iterator, proporcionando un rico entorno para los programadores que están familiarizados con el lenguaje Java.
- **No existen jerarquías.** En JDOM, un elemento XML es una instancia de Element, un atributo XML es una instancia de Attribute y un documento XML es, en sí, una instancia de Document. Puesto que todos representan conceptos distintos en XML, siempre se hace referencia a ellos mediante sus propios tipos, nunca como un "nodo" amorfo.

- **Dirigidos por clases.** Puesto que los objetos JDOM son instancias directas de clases como Document, Element y Attribute, la creación de un objeto es algo tan fácil como utilizar el operador new en el lenguaje Java. Esto también significa que no existe ninguna interfaz de fábrica que deba configurarse, JDOM puede utilizarse directamente.

B. JavaHelp 2.0

"Es un sistema extensible, completo e independiente de la plataforma que permite a los desarrolladores y autores incorporar ayudas en línea en Applets, componentes, aplicaciones, sistemas operativos y dispositivos"[28]. JavaHelp es un producto de Sun Mycrosystems de libre distribución que utiliza archivos XML para su configuración. En la plataforma desarrollada, esta API fue incorporada como biblioteca y se utilizó para gestionar el sistema de ayuda.

3.3.4. Ambientes de Ejecución J2ME

A. El J2ME Wireless Toolkit

El J2ME WTK es una herramienta que provee Sun Microsystems para la compilación, configuración y emulación de aplicaciones J2ME. En la plataforma desarrollada el uso de esta herramienta es transparente para el desarrollador dado que se ha integrado para hacer uso de sus funcionalidades desde el entorno de desarrollo. Se utilizaron las versiones 1.4, 2.0, 2.1, y 2.2 Beta para comprobar el correcto desempeño e independencia, de la plataforma con respecto a cada una de estas versiones. Entre las características de la versión 2.0 están: soporte para MIDP 1.0 y 2.0, CLDC 1.0, Wireless Mesagging API (JSR-120), Mobile Media API (JSR-135). Adicionalmente, a partir de la versión 2.1 se ha adicionado soporte para CLDC 1.1 y servicios web (JSR -172).

B. EL SDK Nokia serie 90

Otro de los ambientes de ejecución J2ME utilizados es el provisto por el Nokia Series 90 midp concept SDK Beta 0.1 for Linux, el cual permite ejecutar aplicaciones J2ME en la plataforma de dispositivos Nokia de la serie 90. Es una herramienta que por ser de ejecución "Stand Alone" puede ser integrada con entornos de desarrollo. Entre sus

características están: MIDP2.0, CLCD1.0, Nokia UI API, Wireless Messaging API(JSR-120), Mobile Media API(JSR 135), BlueTooth(JSR 82).

3.3.5. Componentes de configuración de ejecución

Son componentes que sirven para que la aplicación desarrollada pueda correrse en el ambiente de ejecución de manera transparente para el desarrollador, fundamentalmente estos componentes son los encargados de invocar los comandos necesarios del ambiente de ejecución pasandoles los parámetros requeridos para emular las aplicaciones.

3.3.6. Componentes de construcción de Prometeo

Son componentes desarrollados con dos propósitos, el primero, la creación rápida de interfaces gráficas de usuario con características particulares y el segundo, ocultar al desarrollador la utilización de herramientas provistas por la maquina virtual de JAVA, como por ejemplo la herramienta de compilación, de generación de archivos JAR, etc.

3.3.7. Componentes de construcción J2ME

Estos componentes, aunque están presentes en Prometeo, no deben ser considerados como componentes básicos de construcción de una plataforma para desarrollo J2ME, hacen parte de los componentes que buscan dar valor agregado a la plataforma y son esencialmente los que sirven para realizar un desarrollo más rápido de las aplicaciones a través de la generación de código J2ME.

3.3.8. Núcleo de Prometeo

El núcleo de Prometeo está compuesto básicamente por los paquetes vista, control y modelo, constituye el corazón lógico de la plataforma en el que se articula el comportamiento de los componentes utilizados. Adicionalmente al lenguaje de programación, una de las tecnologías utilizadas en el núcleo es XML, un estándar industrial para la representación de datos, independiente del sistema utilizado que proporciona la extensibilidad suficiente para crear las etiquetas necesarias para un

documento en particular. En esta plataforma es usado para representar y almacenar de manera persistente la información relevante en el modelo de datos del entorno de desarrollo (bibliotecas configuradas, ultimo proyecto abierto, etc), y los proyectos específicos (nombre, midlets, versión del WTK utilizado, emulador, etc).

3.4. PROMETEO y otras plataformas de desarrollo de aplicaciones para dispositivos móviles con J2ME

Otro de los aspectos tenidos en cuenta para realizar las pruebas y validación de la plataforma de desarrollo implementada, consiste en comparar PROMETEO con otras de las plataformas de desarrollo de aplicaciones J2ME, para esto se tomaron en cuenta aquellas que actualmente son utilizadas con mayor popularidad, tal es el caso de JBuilder, Eclipse y NetBeans.

La comparación se realizó teniendo en cuenta capacidad de procesamiento y espacio en disco, wizards que procvee cada plataforma, presencia de diseñador gráfico y ayudantes para completar y depurar código. Los resultados obtenidos se presentan en la tabla 3.1.

Se hace notar que los requerimientos en espacio de disco incluyen todo lo necesario para el desarrollo de las aplicaciones J2ME, esto es, aparte de la plataforma, la máquina virtual de Java y el kit de desarrollo para móviles (WTK).

	Jbuilder X	Eclipse	NetBeans	Prometeo
Espacio en Disco(MB)	250 (Windows) 600 (Linux)	175 (Windows) 167 (Linux)	150 (Linux) 210 (Linux)	60 (Windows) 60 (Linux)
Memoria RAM (MB)	256 min. 512 rec.	256 min. 512 rec.	256 min. 512 rec.	128
Diseñador Gráfico	Si	No	No	En Desarrollo
Wizards	MIDlets Desplegables MIDlet Suites Clases Generales	MIDlets MIDlet Suites Clases Generales	MIDlets MIDlet Suites ClasesGenerales	MIDlets MIDlet Suites Desplegables Record Store Conexiones HTTP Clases Generales
Ayudante para completar Código	Si	Si	Si	No
Depurador de Código	Si	Si	Si	No

Tabla 3.1. Comparación de PROMETO con otras plataformas de desarrollo

3.5. Lista de Actividades

A continuación se presentan las actividades que se consideran mas importantes durante el proceso de desarrollo de una Plataforma, tomando como referencia las realizadas en la construcción de Prometeo, cubriendo cada una de las fases en las que se dividió este proceso. Se hace notar que las actividades mencionadas deberían ser aplicadas en un proceso iterativo a medida que se dota de funcionalidad a la plataforma que se desee construir, además, en cada una de las fases deben establecerse puntos de control y evaluación del proceso seguido.

3.5.1. Fase de Análisis

Las actividades en esta fase tienen por objetivo hacerse una idea general de la plataforma que se desea construir, tratando de abstraer en unos pocos casos de uso gran parte de la

funcionalidad que se desea obtener. En el caso de las plataformas de desarrollo esta fase podría tomar más tiempo que en el caso de una aplicación para un dominio específico, ya que las plataformas de desarrollo deben servir para construir aplicaciones en diferentes dominios. En este sentido, las siguientes actividades se consideran importantes durante esta fase:

Exploración tecnológica: consiste en identificar y utilizar otro tipo de plataformas y entornos de desarrollo que tienen o pudieran tener características similares con la plataforma que se desea construir.

Elaboración de la lista de características de la plataforma: consiste en elaborar una lista, tratando de cubrir todas las características de la plataforma a construir (el estado ideal de la plataforma) de acuerdo con las características de otras plataformas o las que deben ser adicionadas.

Definición del alcance funcional: esta actividad consiste en establecer las características que debe tener la plataforma en su versión inicial (el estado real de la plataforma), a partir de la lista de características ideal que debería tener.

Elaboración del modelo inicial de casos de uso: teniendo en cuenta la lista real de funcionalidades, identificando fundamentalmente el nombre, el propósito y el resumen de cada caso de uso.

3.5.2. Fase de Diseño

Las actividades de esta fase buscan básicamente la construcción de los modelos que soporten la construcción de la plataforma de desarrollo, dentro de estos modelos, en Prometeo cobraron gran importancia el de casos de uso, los diagramas de secuencias y los diagramas de clases, con base en esto, las actividades que se consideran importantes durante esta fase se describen a continuación.

Elaboración del modelo de casos de uso de diseño: consiste en especializar el modelo de casos de uso producto de la fase anterior, dividiendo en otros casos de uso aquellos mas generales e identificando las relaciones entre ellos, y describiendo además del

nombre, el propósito y el resumen, el flujo principal de eventos, así como los posibles flujos de excepción.

Elaboración de diagramas de secuencia: consiste en la elaboración de los respectivos diagramas de secuencia para cada caso de uso de diseño, con base en el flujo principal de eventos y los flujos de excepción.

Elaboración de diagramas de clases: los diagramas anteriores permiten identificar los objetos que interactuan en la implementación de cada daso de uso, con base en estos, se identifican las clases a las que pueden pertenecer estos objetos y las relaciones entre ellas, asi como los atributos y métodos que inicialmente tendrá cada una.

Elaboración de una arquitectura de paquetes: de acuerdo con el pópósito y funcionalidad de cada clase, éstas se agrupan en paquetes que contengan clases con funcionalidades y/o propósitos comunes, con base en estos paquetes y la relación entre cada uno de ellos, se construye una arquitectura inicial de paquetes de la plataforma.

3.5.3. Fase de Implementación

Esta fase busca llevar a un lenguaje de programación específico lo modelos de la fase anterior, se debe notar que hasta este punto no debería importar el lenguaje sobre el cual se pretenda construir la plataforma y por ende los modelos deberían ser válidos para una implementación en cualquier lenguaje. Las actividades que se consideran importantes en esta fase son:

Exploracion tecnologica: es una actividad muy importante y puede disminuir considerablemente el tiempo de implementación, consiste en identificar las bibliotecas o APIs que estén disponibles y que puedan servir para proveer las funcionalidades requeridas por la plataforma. Es recomendable que las bibliotecas o APIs que se utilicen sean de libre distribución.

Implementación de clases: consiste en llevar a un lenguaje específico los modelos y diagramas de clases que se tienen. Durante la implementación de cada clase deben realizarse las respectivas pruebas de unidad que garanticen su correcto funcionamiento.

Cuando se utilicen APIs o bibliotecas adicionales a las del nucleo del lenguaje de implemetación utilizado, se recomieda realizar las pruebas de unidad en diferentes sistemas operativos (cuando sea el caso) así mismo, es recomendable que para considerar terminada la implementación de una clase se cuente con su respectiva documentación (una clase no está terminada hasta que no está documentada).

Corrección de modelos: la implementación de clases siempre trae consigo la aparición de nuevos atributos, métodos o incluso nuevas clases a las que originalmente se encontraban en los modelos, que pueden o no depender del lenguaje de implementación utilizado. Con el fin de mantener los modelos de acuerdo con la implementación realizada, deben hacerse las respectivas modificaciones mediante un proceso de ingeniería inversa.

3.5.4. Fase de Pruebas y Validación

Las actividades realizadas durante esta fase buscan verificar el correcto funcionamiento de la plataforma y corregir los posibles errores que pudieran surgir. Las actividades que se consideran importantes en esta fase se presentan a continuación.

Elaboración del plan de pruebas de integración: consiste básicamente en elaborar un plan de pruebas que permita verificar el cumplimiento de cada uno de los casos de uso, teniendo en cuenta para cada prueba el nombre, la descripción, los resultados esperados y los resultados obtenidos. Es recomendable que el plan de pruebas de integración sea elaborado por una o varias personas que conozcan los requerimientos pero cuya participación en la implementación haya sido mínima.

Aplicación del plan de pruebas: consiste en realizar las pruebas descritas en el plan a la plataforma construida. Cuando los resultados esperados difieran de los resultados obtenidos, debería elaborarse una lista de errores identificando el caso de uso en el que se dio el error, los parámetros de entrada proporcionados a ese caso de uso, el resultado que se obtuvo y el resultado que se esperaba.

Corrección de errores: consiste en corregir cada uno de los errores que aparezca en la lista de errores producto de la aplicación del plan de pruebas.

Actualización de modelos: debido a que la corrección de errores puede traer consigo nuevos métodos, atributos, clases, y en general, modificación del código, es necesario actualizar los modelos para mantener la consistencia entre estos y la implementación finalmente realizada.

Elaboración de manuales de usuario: cosiste en elaborar las guías de instalación y utilización de la plataforma construida.

Elaboración de instaladores o ejecutables: en este punto, se debe asegurar que la plataforma construida cuenta con la capacidad operacional proyectada, se crean entonces los instaladores o ejecutables para que sea implantada en el entorno adecuado.

IV. CONCLUSIONES, RECOMENDACIONES Y TRABAJOS FUTUROS

- Si bien en el desarrollo de software basado en componentes, el análisis de los posibles dominios en los cuales va a interactuar el componente requiere mayor tiempo que el análisis de dominio de una aplicación basada en objetos, ésta desventaja se ve compensada por la notable disminución del tiempo de implementación, al explotar al máximo la reutilización de los componentes; además de proveer a la aplicación de una mayor flexibilidad y extensibilidad permitiendo ser rápidamente modificada, reemplazando los componentes por otros que posean la misma interfaz, y minimizando el tiempo de integración al realizar las modificaciones. Por lo anterior para la construcción de sistemas de mediana y alta complejidad el desarrollo basado en componentes es una opción muy favorable.

- Dentro de las desventajas de la implementación de plataformas de desarrollo se tiene el tiempo necesario para que una vez implantada, los desarrolladores aprendan a utilizarla, para minimizar esa desventaja las plataformas de desarrollo deben ser implementadas de tal manera que sean totalmente intuitivas; abarcando desde un manejo muy simple, para usuarios que empiezan en el desarrollo del lenguaje soportado por la plataforma, hasta características de configuración avanzadas para usuarios con mayor experiencia y expectativas más altas. Prometeo, por estar especializado en J2ME provee interfaces bastante intuitivas que ocultan muchos detalles de implementación al desarrollador, permitiendo una rápida implementación de aplicaciones J2ME.

- PROMETEO brinda un buen soporte al desarrollo de aplicaciones para dispositivos móviles, ya que provee un ambiente de construcción y se integra con el ambiente de ejecución de manera transparente para el desarrollador.

- A pesar de que Java es un lenguaje interpretado (no es compilado a código nativo) lo que lo hace lento frente a lenguajes compilados, su utilización como lenguaje de programación para la implementación de plataformas de desarrollo se justifica debido a su portabilidad y sobre todo a la cantidad de componentes y APIs ya existentes que disminuyen de manera considerable el tiempo de desarrollo.

- El desarrollo de PROMETEO se realizó en un lenguaje “Independiente de la Plataforma de Ejecución”, sin embargo, no se obtiene una completa independencia, (ver Anexo C: Inconsistencias de portabilidad) se encontraron algunos métodos cuyo resultados varían de una plataforma a otra, específicamente para este caso, del sistema operativo MS-WINDOWS a sistemas operativos GNU/LINUX, que en cierta forma comprometen la portabilidad del código generado y que requieren de código adicional para lograr una portabilidad mayor. Con relación a esta conclusión, se recomienda que cuando se deba ejecutar una aplicación en distintos sistemas operativos, se pruebe la aplicación (cuando haya tenido algún cambio significativo) en un sistema operativo diferente al que se está realizando el desarrollo.

- Aunque existen otras plataformas de desarrollo que brindan soporte a la construcción de aplicaciones para dispositivos móviles, tal es el caso de JBuilder, SunOne, Eclipse y NetBeans, estás dos ultimas de libre distribución, entre otras; los requerimientos de procesamiento que estás necesitan son mucho mayores a los de PROMETEO, dado que esta se especializa en aplicaciones J2ME.

- La primera versión de PROMETO no cuenta con un ayudante para completar código, del cual si disponen las plataformas mencionadas anteriormente y que podría ser adicionado en trabajos futuros. Sin embargo, los generadores de código presentes en Prometeo para el control del ciclo de vida de las aplicaciones, interfaces gráficas de usuario, almacenamiento persistente en el dispositivo móvil y conexión con redes HTTP se hacen muy útiles y pretenden no solo dar soporte a la implementación de aplicaciones sino también a mejorar el aprendizaje de J2ME.

- PROMETEO se desarrolló específicamente para su utilización con el perfil MIDP 2.0, sin embargo, sus característica de configuración permiten dar soporte no solo a este perfil, sino también al perfil MIDP1.0 y a otros que puedan surgir sobre las configuraciones CLDC1.0 y CLDC1.1.

- El trabajo realizado durante la construcción de Prometeo y las clases y componentes finalmente implementados y utilizados, permiten definir una arquitectura funcional para

plataformas de desarrollo construidas sobre la tecnología JAVA y que específicamente sirvan para el desarrollo de aplicaciones en dispositivos móviles con tecnología J2ME.

- La arquitectura funcional definida para PROMETEO permite caracterizar los componentes mínimos necesarios para construir una plataforma de desarrollo de aplicaciones para móviles utilizando J2ME. En los componentes de construcción de Prometeo, además de los componentes que permiten crear las interfaces gráficas del ambiente de desarrollo, entre los que se destacan un editor y un mecanismo de representación de archivos (un árbol para el caso de PROMETEO), se necesitan componentes que sirvan de puente entre la aplicación J2ME y el ambiente de compilación (J2SDK). De igual forma, los componentes de configuración de ejecución - como se han llamado - sirven de interfaces entre la aplicación J2ME y el ambiente de ejecución (J2ME WTK), de forma que la utilización de este sea transparente para el desarrollador.

- El trabajo desarrollado a los largo del planteamiento, formulación, ejecución y validación del proyecto que dejó como resultado la construcción de PROMETEO, permite de manera general y sin pretender entrar en el planteamiento de una metodología, definir ciertas actividades que se consideraron de importancia en todo el proceso de desarrollo y que deberían ser tenidas en cuenta en su respectivo momento, cuando se trate de la implementación de una plataforma de desarrollo y más específicamente, una plataforma de desarrollo de aplicaciones para móviles con tecnología J2ME.

- PROMETEO cuenta con funcionalidades que lo convierten en una buena herramienta para desarrollar aplicaciones para dispositivos móviles con J2ME, sin embargo, posee ciertas restricciones que dan pie a pensar en trabajos futuros que lleven a mejorarlo y mantenerlo conforme evoluciona la tecnología J2ME y se hacen más complejas las necesidades de los desarrolladores. Dado que Prometeo fue construido dentro del grupo de interés en aplicaciones para móviles W@PColombia, se recomienda particularmente a este grupo continuar desarrollando y dando soporte a la plataforma construida.

REFERENCIAS

Capitulo 1.

[1] Montilva, Jonas A. “Desarrollo de Software Basado en Componentes”. Universidad de los Andes. Facultad de Ingeniería. Escuela de Ingeniería de Sistemas. Mérida – España.

[2] Griss, M. L. “Domain Engineering and Variability in the Reuse-Driven Software Engineering Business”. Object Magazine. 1996.

[3] Griss, M. L., Favaro, J. y d’Alessandro, M.. “Integrating Feature Modeling with RSEB”. Proceedings of the Fifth International Conference on Software Reuse, ICSR-5, páginas 76-85. 2 - 5 June, 1998, Victoria, B.C., Canada. IEEE-CS.

[4] Simos, M., Creps, D., Klingler, C., Levine, L. y Allemang, D. “Organization Domain Modeling (ODM) Guidebook” – Version 2.0. Technical Report STARS-VC-A025/001/00, Lockheed Martin Tactical Defense Systems, 9255 Wellington Road Manassas, VA 22110-4121. 1996.

[5] Kang, K. C. “Feature-Oriented Development of Applications for a Domain”. Proceedings of the Fifth International Conference on Software Reuse, ICSR-5, páginas 354-355. 2 - 5 June, 1998, Victoria, B.C., Canada. IEEE-CS. 1998

[6] García P, Francisco J. Barras, Juan A. Laguna S. Miguel A. Marques C, José M. “Líneas de Productos, Componentes, Frameworks y Mecanos”. Informe Técnico. Departamento de Informática y Automática. Universidad de Salamanca. 2002

[7] Moreno Navarro, Juan José. “Lenguajes, Sistemas Informáticos e Ingeniería del Software”. Curso de Doctorado LSIIS. Departamento de Informática. Universidad Politécnica de Madrid.

[8] Markiewicz, Marcus Eduardo. Lucena, Carlos J. ACM Crossroads Student Magazine. The ACM's First Electronic Publication. 2001.

[9] Nierstrasz, Oscar. Tsichritzis, Dennis. “Object-Oriented Software Composition”. Prentice Hall. 1995

[10] Martínez Ortega, José Fernán. “CORBA y Java Como Soporte de Aplicaciones y Servicios de Ingeniería Distribuida”. PI FIET 0682. Facultad de Ingeniería Electrónica Y Telecomunicaciones. Universidad del Cauca. Popayán.

[11] Sun Microsystems. “JavaBeans”. 1997

[12] http://www.programacion.com/java/tutorial/beans/

[13] http://java.sun.com/products/javabeans/

Capitulo 2.

[14] FORUM NOKIA. MIDP 2.0 Introduction Version 1.0. Mayo de 2003

[15] FORUM NOKIA. MIDP2.0: Tutorial On Signed MIDlets. Version 1.0. Mayo de 2004

[16] GALVEZ ROJAS SERGIO. ORTEGA DIAZ LUCAS. JAVA A TOPE: J2ME(JAVA 2 MICRO EDITION). Universidad de Málaga 2003.

[17] IBM Developer works. J2ME: Step by Step.

[18] Motorola, Inc. and Sun Microsystems, Inc. Mobile Information Device Profile, v2.0 (JSR-118). Noviembre de 2002

[21] PRIETO Manuel J. CURSO DE J2ME (JAVA 2 MICROEDITION), Abril de 2003.

[22] Sun Microsystems Inc. Mobile Information Device Profile, v1.0 (JSR-37). Diciembre de 2000.

Capítulo 3

[23] Documentación Nokia SDK Serie 90

[24] Elliote Rusty Harold. PROCESSING XLM WITH JAVA, 2002

[25] Larman Craig. UML Y PATRONES Introducción al análisis y diseño orientado a objetos. PRENTICE HALL, México,1999.

[26] Grupo de Ingenieria Telemática. Modelo Para Construcción de Soluciones. Universidad del Cauca, 2001

[27] Rendón Gallón Alvaro. EL LENGUAJE UNIFICADO DE MODELADO (UML). Facultad de Ingeniería Electrónica Y Telecomunicaciones. Universidad del Cauca. Popayán. Mayo de 2000

[28] http://java.sun.com/products/javahelp/