PROYECTO DE GRADO – INCO – AÑO 2005
FACULTAD DE INGENIERIA
UNIVERSIDAD DE LA REPUBLICA

Proyecto de Grado: Estudio y desarrollo de aplicaciones gráficas para dispositivos móviles

Informe final

Estudiantes:
Agustín Musso C.I.: 3.596.892-2
Andrés Laguna C.I.: 1.884.743-8

Tutor responsable:
Ing. Eduardo Fernández

AGRADECIMIENTOS Y CONTACTOS

En primer lugar queremos agradecer a quienes colaboraron de alguna forma con la realización de este trabajo, ya sea a través de entrevistas en las cuales compartieron sus conocimientos y/o emprendimientos relacionados con la temática de este proyecto, o con quienes facilitaron material, software y/o dispositivos móviles que permitieron el desarrollo de ésta aplicación. Ellos son:

- Ing. Eduardo Fernández, tutor de este Proyecto.
- Lic. Jorge Eguren Gerente de RRHH de la empresa GIGLOBAL.
- Personal de la empresa GlobalNet: A/S Pablo Pietropinto (Gerente IT), Lic. Juan Francisco Saavedra (C.E.O.), Ignacio Kliche (Gerencia de Marketing y Desarrollo de Nuevos Negocios).
- Juan Suárez de la empresa Microsoft Uruguay.
- Personal de la empresa ANCEL: Ing. Fabián Rodríguez, Sr. Nelson Bofia.
- Carlos Galucci de la empresa NC.
- Marcos Campal, de la empresa HandSoft.
- A/S Fernando Sansberro – Director de Batoví Games Studio.

A continuación enumeramos una serie de personas y empresas uruguayas que realizan actividades afines con la temática de este proyecto, a modo de guía para quienes estén interesados en profundizar sobre éstos temas.

Contacto	Descripción
Empresa NC Carlos Galucci: cgalucci@adinet.com.uy	El trabajo de la empresa consiste en el desarrollo de aplicaciones para dispositivos PDA y celulares inteligentes.
ANCEL - Área Tecnología de la Información www.ancel.com.uy Teléfono: 480 44 55	Empresa de telecomunicaciones nacional, que tiene la mayoría del mercado de celulares en el Uruguay. Junto con la empresa China Huawei han realizado un proyecto piloto sobre redes de telecomunicaciones de 3G.
Empresa GlobalNET www.globalnet.com.uy Dirección: Luis Lamas 3168 Teléfono: 628 88 77	GlobaNet brinda servicios y aplicaciones de celulares (Ej.: sistemas de votación y mensajería para programas de televisión a través de celulares). Es una de las empresas que participaron en el proyecto piloto entre ANTEL y Huawei para probar una red de telecomunicaciones de 3G.
Empresa HandSoft http://www.handsoft.com.uy	Desarrollan aplicaciones sobre las plataformas PalmOS, Windows Mobile, SymbianOS, Java, BREW, protocolos WAP, SMS, MMS, xHTML, especialmente para redes personales (PAN) y locales (LAN), en forma inalámbrica a través de Bluetooth y Wi-Fi (WLAN ó 802.11).
Empresa Batoví Games Studio http://www.batovi.com/eng/ Teléfono: 619 22 15	Batoví es una empresa de desarrollo de juegos para teléfonos móviles; tienen especial interés en comenzar a explorar en este año las posibilidades de aplicaciones gráficas en 3D.
Empresa: Quartz Sistemas http://www.quartz.com.uy/ Correo: info@quartz.com.uy	Desarrollan aplicaciones para Pocket PC. • Gestión de distribución de mercaderías en rutas de visitas con funcionalidades de Preventa, Cobranza y Venta Directa • Diseño y publicación de encuestas o recolección de datos en cualquier lugar con posterior manipulación en un PC.
Empresa Roadup http://www.roadup.com.uy/	Desarrollan para PalmOS y Windows Mobile fuertemente orientados a Sistemas de Información Geográfica.
Empresa: New age data http://www.new-age-data.com/ Correo: desarrollo@new-age-data.com	Desarrollaron el Sistema TransAct, que permite optimizar las transacciones de crédito, débito y consulta de cheques de una empresa a través de un PDA.
Empresa Datasur http://www.datasur.com.uy Correo: info@datasur.com.uy	Desarrollan para Windows Mobile. Entre sus productos esta el denominado Dealer, para la toma de pedidos y facturación móvil sincronizando los datos con un PC mediante el intercambio de archivos.

RESUMEN

Este proyecto consiste en el estudio de la tecnología de dispositivos móviles (celulares y PDA), orientado principalmente al desarrollo de aplicaciones gráficas. El trabajo consta de dos partes: en la primera se realiza una investigación de la tecnología de los dispositivos móviles, con especial énfasis en las características gráficas de los mismos, y en las posibilidades de desarrollo de aplicaciones; en la segunda se desarrollan aplicaciones con componentes gráficos y que contemplen la mayor portabilidad posible entre los distintos dispositivos móviles.

La temática abarcada en la etapa de investigación del proyecto incluye el estudio del estado del arte y una breve descripción de la evolución histórica de la tecnología móvil. También se estudian las distintas características de los sistemas operativos de este tipo de dispositivos, poniendo especial énfasis en: **Symbian OS** debido a su gran difusión comercial a nivel mundial; **Palm OS**, por ser el predominante entre los dispositivos PDA; **Windows Mobile OS**, debido a su auge en la comercialización mundial, y porque apunta a una arquitectura propietaria y cerrada; y **SavaJe OS**, que tiene por objetivo alcanzar una compatibilidad total con el lenguaje de programación Java SE.

Posteriormente, se analizaron los lenguajes de programación dominantes para este tipo de dispositivos: C++ y J2ME, junto con los entornos de desarrollo de aplicaciones asociados, las APIs gráficas predominantes de cada lenguaje (*OpenGL ES* y *Mobile 3D Graphics* respectivamente) y los correspondientes emuladores de los dispositivos más utilizados. También nos detenemos en el enfoque de desarrollo de la **plataforma BREW**.

En la segunda parte del proyecto desarrollamos aplicaciones con componentes gráficos, diseñadas para ejecutarse en la mayor cantidad de dispositivos móviles posibles. La primera de las aplicaciones está orientada a la graficación en 3D, y en su diseño se han intentado atacar los principales problemas que se presentan en este tipo de aplicaciones, como ser el obtener imágenes de una calidad adecuada (considerando la escasa resolución de pantalla de los dispositivos móviles). Otro de los problemas fue crear la aplicación para ser ejecutada en tiempo real sin empobrecer los gráficos y teniendo en cuenta la escasez de recursos de los dispositivos móviles (velocidad de procesamiento y capacidad de memoria). La aplicación consiste en un laberinto, dentro del cual el usuario mueve un personaje y donde además un conjunto de "enemigos" que lo perseguirá. El desarrollo de la aplicación se realizó en C++ con la API gráfica OpenGL ES, para poder ejecutarse en dispositivos Symbian (aunque el código podría adaptarse para PalmOS versión Cobalt y Windows Mobile), pero sólo ha llegado a ejecutarse en emuladores. También se realizo una versión de la aplicación en J2ME, que además de ejecutar en emuladores, pudo ser probada en algunos celulares de forma exitosa. La segunda aplicación desarrollada agrega un componente de comunicaciones, el cual es fundamental en muchas de las aplicaciones para móviles. Se trata de una versión del "Truco" para ser jugada entre dos dispositivos que soporten Bluetooth (protocolo de comunicaciones inalámbricas de corto alcance). La parte gráfica de la aplicación esta desarrollada con el API 3D; si bien la apariencia de las cartas es 2D la superposición de cartas y el ocultamiento de las porciones no visibles es resuelto por la API gráfica en 3D. Se utilizan además texturas, escalado, rotación y traslación, utilizando una mínima cantidad de imágenes para dibujar las diferentes cartas. La aplicación fue probada con éxito en los emuladores.

PALABRAS CLAVE: 2G, 2.5G, 3G, 2D, 3D, Symbian, Windows Mobile, Palm, SavaJe, J2ME, C++, BREW, M3G, OpenGL ES.

ÍNDICE

1. Introducción

En los últimos años, la tecnología de los celulares y PDA ha realizado avances substanciales. Se han ido incorporado nuevas funcionalidades a sus capacidades primarias (teléfono y agenda), transformándose en dispositivos de propósito general, con variados usos que van desde la trasmisión de fotos, herramientas para la localización de personas, conectividad con redes LAN y WAN, conexión a Internet, acceso al correo electrónico y bases de datos, envío de faxes y mensajes de texto, actividades de esparcimiento, etc. Además del considerable aumento en las capacidades y funcionalidades de este tipo de dispositivos, se ha registrado un crecimiento exponencial en el uso comercial de los mismos, existiendo actualmente una gran variedad de modelos de dispositivos con diversas capacidades relacionadas con el hardware de los mismos, diferentes sistemas operativos y plataformas de desarrollo. Si bien existen algunas tendencias, no existen estándares que sean seguidos por la totalidad de los diferentes fabricantes de dispositivos y desarrolladores de aplicaciones.

Ante este escenario, se planteó la necesidad de realizar un relevamiento de las tendencias actuales, descripción de las prestaciones usuales, y las diferentes posibilidades de las tecnologías existentes. En la segunda parte del proyecto, se desarrollaron aplicaciones con componentes gráficos, las cuales se pueden ejecutar en una gran variedad de dispositivos.

1.1 Motivación

El desarrollo de aplicaciones para dispositivos móviles está creciendo a nivel mundial, principalmente en aplicaciones de entretenimiento y en usos comerciales como ser: la posibilidad de realizar compras a través del celular, consultar cuentas bancarias, realizar reservas en restaurantes, etc. La principal dificultad con la que se encuentran los desarrolladores es la gran variedad de dispositivos, sistemas operativos y herramientas de desarrollo disponibles, lo cual dificulta obtener una solución "global" para los diferentes dispositivos existentes.

Por tanto, utilizando los estándares existentes y las herramientas de desarrollo que aseguren de la mejor forma posible la reutilización del código, se propuso desarrollar aplicaciones con componentes gráficos capaces de ser ejecutadas en un amplio abanico de dispositivos, dejando constancia de los elementos a considerar para crear código reutilizable.

1.2 Objetivos

El proyecto se divide en dos partes. La primera consiste fundamentalmente en el estudio del estado del arte referente a la tecnología de celulares y PDA, en particular orientado hacia las aplicaciones gráficas, además de investigar las herramientas disponibles para el desarrollo de aplicaciones. Se plantean como objetivos de ésta parte:

- Realizar un estudio sobre estado del arte en la tecnología de celulares y PDA.
- Establecer estándares actuales y en desarrollo que faciliten la convergencia de las tecnologías.
- Delimitar entornos de desarrollo que faciliten la programación y el testeo de aplicaciones para celulares y PDA.

En la segunda parte del proyecto se pretende crear algunas aplicaciones con componentes gráficos (principalmente enfocado en las prestaciones en 3D), desarrolladas con las herramientas estudiadas anteriormente. Los objetivos propuestos para ésta parte son:

- El diseño y desarrollo de al menos dos aplicaciones con componente gráfico, utilizando las herramientas estudiadas en la parte anterior.
- Lograr que las aplicaciones ejecuten en la mayor cantidad de dispositivos posibles.
- Como requerimiento opcional, incluir en alguna de las aplicaciones alguna funcionalidad relativa a las comunicaciones entre dispositivos; si bien el proyecto está orientado al desarrollo de aplicaciones gráficas, el área de las comunicaciones es una de las más importantes dentro de los dispositivos móviles.

1.3 Resultados esperados

Los resultados esperados en la primera parte del proyecto son:

- Documentación que brinde una visión general del estado del arte referente a la tecnología de dispositivos móviles.
- Establecer un conjunto de dispositivos móviles, sistemas operativos y herramientas de desarrollo que sea representativo de los existentes en el mercado, para poder centrar en ellos la segunda parte del proyecto.
- Lograr obtener un buen pool de esas herramientas de desarrollo, si es posible en forma gratuita o versiones trial, a efectos de basarse en ellas para el desarrollo de aplicaciones en la segunda parte.

Para la segunda parte del proyecto, los resultados esperados son:

- Establecer una metodología de desarrollo que facilite la creación de código reutilizable para las distintas plataformas.
- Al menos un par de aplicaciones con componentes gráficos (y opcionalmente algún componente de comunicaciones). Dichas aplicaciones deben correr en la mayor cantidad de dispositivos posible, o al menos en aquellos dispositivos que cumplan con los estándares sobre los cuales se han basado el diseño e implementación de las aplicaciones.

1.4 Conclusiones

En lo relativo a la primer parte del proyecto creemos que hemos logrado desarrollar material que brinda una visión general sobre el avance de la tecnología de los dispositivos móviles y orientar al lector en las diferentes prestaciones y servicios disponibles en la actualidad. También se han estudiado las distintas características de los sistemas operativos, lenguajes de programación, entornos y herramientas de desarrollo, bibliotecas gráficas y del hardware más utilizado en los dispositivos móviles; esta información es complementada por una metodología para desarrollar aplicaciones en los dos lenguajes de programación más utilizados C++ y Java. Por éstas razones consideramos que, si bien puede extenderse y profundizarse el estudio de los diferentes temas tratados, logramos establecer un punto de partida para quienes pretendan incursionar en la temática del proyecto, sobre todo en lo referido a aplicaciones gráficas en 3D, las cuales se encuentran en su etapa inicial de desarrollo a nivel mundial, y que aún no han comenzado a desarrollarse activamente en Uruguay.

Respecto a la segunda parte del proyecto se logró establecer una metodología de desarrollo de aplicaciones portables entre los diferentes sistemas operativos y dispositivos, principalmente basada en el desarrollo de aplicaciones Java. También se han implementado dos aplicaciones gráficas, una de ellas consiste en el juego de cartas Truco, que contiene un componente de comunicaciones entre dos celulares y implementada en Java; ésta aplicación solamente ha podido ser probada en emuladores, debido a la carencia de dispositivos adecuados para su funcionamiento. La otra aplicación es un juego en 3D que fue implementado en Java y C++; las pruebas de ésta aplicación se lograron hacer exitosamente en los emuladores correspondientes, y además pudo realizarse una prueba práctica en dos modelos de celulares que soportan las características de 3D para Java, quedando pendiente la prueba en un dispositivo basado en el sistema operativo para el cual se creó la aplicación basada en C++; ésta prueba tampoco pudo llevarse a cabo debido a la imposibilidad de obtener un dispositivo de dichas características.

Cabe destacar que la dificultad en conseguir dispositivos adecuados para ejecutar las aplicaciones del proyecto, se debe a que los dispositivos que soportan gráficos en 3D pertenecen a la última generación de celulares, los cuales aún no se encuentran lo suficientemente difundidos en el mercado uruguayo. El enfoque que hemos realizado a este proyecto, fue basado principalmente en las aplicaciones 3D, ya que existe suficiente conocimiento y aplicaciones comerciales que utilizan gráficos en 2D, y aún empresas que desarrollan ese tipo de aplicaciones en Uruguay.

1.5 Organización del documento

Se describe a continuación la organización general de este documento, destacando los aspectos más importantes de cada capítulo.

1. **Introducción**: este capítulo del documento.
2. **Estado del Arte**: en este capitulo de realiza una breve descripción de la evolución de la tecnología celular hasta la actualidad, describiendo especialmente la tecnología utilizada en Uruguay y ubicándola en el contexto mundial. También se describe brevemente el estándar de comunicaciones ITM-2000 y los principales servicios brindados por la tercera generación de celulares (3G). Luego se realiza un paneo entre los principales sistemas operativos de dispositivos móviles, lenguajes de programación, plataformas de desarrollo, IDEs y SDKs; se incluyen varios anexos que complementan la información brindada en este capítulo.
3. **Desarrollo de aplicaciones**: aquí se detallan aspectos referentes a la metodología de desarrollo. Se divide en dos secciones, una dedicada al desarrollo en Java y otra a aplicaciones C++ para Symbian OS.

4. **Aplicaciones desarrolladas en el proyecto**: aquí se detallan aspectos referentes a la implementación de las aplicaciones desarrolladas y las decisiones que se tomaron en cada caso. Se divide en dos secciones, una dedicada a la aplicación "Laberinto" y la otra a la aplicación "Truco".
5. **Conclusión y trabajos a futuro**: se describen las principales conclusiones que resultaron del proyecto, además de los trabajos que se pueden encarar en el futuro para complementar este trabajo.

   Glosario: se definen y explican los acrónimos así como las principales terminologías usadas en este documento.

   Referencias: sitios y publicaciones que se han tomado como referencia para la realización de este trabajo.

   Anexos: índice de todos los anexos que complementan a este informe.

2. Estado del arte

Según [10] se considera que la telefonía celular tuvo su inicio en 1973 cuando Martin Cooper, empleado de Motorota, creó el primer teléfono que funcionaba con ondas de radio, en lugar de la tecnología habitual basada en la transmisión a través de los cables telefónicos. En sus inicios, los teléfonos celulares fueron concebidos exclusivamente para la transmisión de voz, utilizando tecnología analógica. En el año 1979 aparecen en Japón los primeros sistemas comerciales de telefonía celular, implantados por la compañía NTT. En Europa los sistemas comerciales comienzan a desarrollarse en 1981, y en el año 1983 comienzan a operar en los Estados Unidos. Hasta la década de los 90 se utilizaron tecnologías similares (y analógicas), pero debido a la creciente demanda, este tipo de tecnología fue resultando insuficiente para los requerimientos del mercado, perdiendo calidad en las comunicaciones, y se proyectaba que de acuerdo al crecimiento anual de los usuarios, el sistema iba a colapsar en poco tiempo. Estas características agrupan a la llamada primera generación de la tecnología celular (1G). A pesar de las dificultades, paulatinamente se comienza a ver a la comunicación inalámbrica como una alternativa a la telefonía convencional, aunque se hace urgente un cambio sustancial en el sistema de comunicaciones.

En los años 90 se incorpora la tecnología digital, dando origen a la segunda generación (2G), la cual convive con la 1G, y permitió solucionar el problema de saturación que presentaba la generación anterior. La tecnología digital permitió además agregar nuevos servicios como fax y mensajería SMS (Short Message Service), dando origen a la transmisión de datos, aunque con ciertas limitaciones aún, sobre todo en la velocidad de la transferencia de los datos. También en la década de los 90 comienzan a desarrollarse los primeros PDA (Personal Digital Assistant), que rápidamente incorporaron la posibilidad de sincronización con los PC, además de las funcionalidades de agenda (la cual incluían desde un principio). Posteriormente los PDA incorporan funcionalidades de correo electrónico en sincronización con los PC, y se comienza a vislumbrar la posibilidad de dotar a los dispositivos móviles, de funcionalidades hasta entonces reservadas exclusivamente para las computadoras. El principal problema es aún las restricciones tecnológicas existentes para los dispositivos móviles como ser la poca capacidad de procesamiento y memoria, y la baja velocidad de transferencia, a pesar de las mejoras respecto a la generación anterior. Algunos modelos de celulares empiezan a incorporar pantallas con resoluciones que permiten el despliegue de información más compleja y completa que los nombres y números telefónicos de la agenda (resoluciones de 96 * 96 píxeles y 128 * 128 píxeles); las comunicaciones comienzan a incorporar protocolos de encriptación, mejorando sustancialmente la seguridad de las comunicaciones, y comienzan a desarrollarse aplicaciones orientadas a los dispositivos móviles. Se alcanzan los 700 millones de usuarios a nivel mundial, superando ampliamente los niveles alcanzados por la 1G, y paralelamente se va eliminando la antigua tecnología analógica, aunque los estándares de comunicaciones permiten la utilización simultánea y la interconexión de los dispositivos analógicos y digitales, debido a la persistencia de ambas tecnologías.

La 3G tuvo su inicio comercial en Japón en el año 2001, extendiéndose a Europa y Asia en el año 2002, y posteriormente a Estados Unidos y otros países. Los nuevos avances de la tecnología, permiten agregar paulatinamente nuevos servicios, como ser la posibilidad de conexión a Internet, correo electrónico, y la descarga y reproducción de archivos Multimedia desde los dispositivos móviles (celulares y PDA), dando comienzo a la tercera generación de celulares (3G). La característica principal de ésta generación de dispositivos, es la gran velocidad de transmisión incluso en situaciones de alta movilidad del usuario. Con una movilidad limitada de hasta 10 Km./h (caminando) se pueden obtener velocidades de transferencia de hasta 2 Mbps; conduciendo automóviles a una velocidad menor que 120 Km./h se pueden obtener tasas de transferencia de hasta 384 Kbps, y desplazándose a más de 120 Km./h se pueden obtener transferencias de hasta 144 kbps. Se estima que 3G alcanzará a 1,150 millones de usuarios a nivel mundial. También comienzan a aparecer los Smart-phones, que son teléfonos móviles con mayor capacidad de procesamiento y memoria, que incorporan algunas de las funcionalidades de PDA. Casi simultáneamente, algunos modelos de PDA incorporan la posibilidad de realizar llamadas telefónicas, por lo tanto comienza a volverse difusa la línea que separa a los PDA de los teléfonos móviles; en el contexto de este trabajo utilizamos el término "dispositivo móvil" para referirnos a cualquiera de ellos: teléfonos celulares, Smart-phones y PDA.

En la actualidad, en muchos países del mundo aún no se han desarrollado los servicios 3G, principalmente por un problema de costos; mientras que los países más desarrollados económicamente pudieron migrar la totalidad de sus servicios (Ej.: Japón); en otros países lo que se está realizando es una migración gradual desde 2G a 3G, agregando paulatinamente nuevos servicios a la 2G, y realizando paralelamente los cambios necesarios en la red. Ésta estrategia de cambio de tecnología ha recibido el nombre de generación 2,5 (2.5G), y se trata como mencionamos de un híbrido de la 2G, que en algún momento de su evolución debería culminar en la 3G.

2.1 Tecnología existente en Uruguay

En la actualidad existen tres compañías que brindan servicios de telefonía móvil en Uruguay, utilizando una tecnología GSM (Global System for Mobile Communications), que es un estándar internacional de comunicaciones digitales celulares que tuvo origen en la 2G, y al cual se le han ido agregando paulatinamente servicios de 3G; por lo tanto puede considerarse que en Uruguay las redes existentes son de 2.5G.

Las compañías de telefonía móvil que brindan servicios en territorio nacional son:

- *Ancel*: compañía nacional y estatal de telecomunicaciones. Su red *GSM* (2.5G) esta basada en la tecnología *GPRS*. Junto con la empresa China de Telecomunicaciones Huawei realizó una prueba piloto de una red 3G, la cual tenía cobertura parcial en la cuidad de Montevideo y permitió realizar pruebas de todos los servicios 3G, destacándose la TV Móvil. La red piloto (aún activa para test), dispone de una velocidad de transferencia de 384 Kbps.
- *Movistar*: representante del Grupo internacional *Telefónica Móviles*; posee una red *GSM* y *CDMA 1x* (paso previo a tecnología 3G). Recientemente a incorporado el servicio de TV Móvil a una velocidad de transferencia de 144 Kbps (inferior a los niveles alcanzados por la prueba piloto Ancel-Huawei).
- *CTI Móvil*: empresa subsidiaria de la internacional *América Móvil*. Utiliza una red *GSM* propia, basada en la tecnología *GPRS*.

Estas tres compañías además de brindar el servicio de telefonía, ofrecen mensajes *SMS* (los cuales pueden enviarse entre dispositivos arrendados a cualquiera de las tres compañías), Internet, correo electrónico, WAP y también permiten cierto grado de trasmisión de archivos multimedia (*MMS*). También comienzan a investigar las posibilidades de explotación de aplicaciones para dispositivos móviles, auque

todavía esta posibilidad no es explotada comercialmente en forma masiva en el Uruguay[1]. También soportan *roaming* internacional, es decir que si el propietario de un teléfono celular nacional necesita viajar al exterior, puede mantener el servicio de cobertura durante el viaje; para ello debe informar a su proveedor de servicio telefónico hacia donde viajará, y en caso de ser una zona con la cual el proveedor tiene convenio se le habilitarán las llamadas tanto en forma local como hacia el exterior (siempre dentro de los convenios manejados por el proveedor).

En el contexto de América Latina, la red de telefonía celular Uruguaya se encuentra muy bien posicionada, ya que no existe en el continente una red 3G completa, y los servicios brindados en Uruguay igualan a las redes de mayor porte del continente; el único punto que Uruguay no explota masivamente es el desarrollo de aplicaciones para dispositivos móviles, a diferencia de otros mercados mayores como México, Brasil y Argentina. El proyecto piloto impulsado por la empresa Huawei realizando conjuntamente con ANTEL (Ancel), es el primer paso dado en el continente orientado exclusivamente a una red completamente de 3G; se busca con este proyecto probar la compatibilidad de la tecnología existente en el país con la tecnología 3G y agregar nuevos servicios propios de la última generación, como ser la trasmisión de TV a través de los dispositivos móviles, mejoras en otros aspectos como ser la velocidad de transferencia y aumentar la seguridad en las comunicaciones respecto al servicio 2.5G existente. Si bien el proyecto finalizó formalmente, las instalaciones siguen siendo operativas y se continúan realizando algunas pruebas.

Dentro de la familias de tecnologías GSM (2.5G), las utilizadas en Uruguay son:

- *GPRS*: que permite la transmisión de datos a alta velocidad en redes inalámbricas, admitiendo conexiones telefónicas móviles de hasta 115 Kbit/s, y proporciona servicios de acceso a Internet y e-Mail.
- *EDGE*: permite suministrar servicios de telefonía móvil de 3G. Soporta la transmisión de datos, servicios y aplicaciones multimedia hasta 384 Kbit/s.
- *UTMS/WCDMA*: tecnología de voz y datos a alta velocidad que integra la familia de normas inalámbricas 3G (ITM-2000).

Si bien las velocidades mencionadas en el párrafo anterior corresponden a las redes más potentes a nivel mundial, en Uruguay las redes de mayor velocidad no superan los 144 Kbps en producción, y alcanzan los 384 Kbps solamente en la red Piloto de 3G utilizada en el convenio Ancel y Huawei.

Desde el punto de vista de las aplicaciones gráficas para dispositivos móviles, tenemos que algunas de las empresas han desarrollado sistemas de información geográfica, trasmisión de imágenes, detección de movimiento mediante cámaras digitales remotas (que funciona como un sistema de alarma al dispositivo móvil), algunos juegos en 2D, y recientemente se incorpora la TV móvil. Sin embargo, no se han desarrollado **aplicaciones gráficas basadas en 3D**, en los cuales exista una proyección en perspectiva, detección de colisiones entre objetos, y otras técnicas habituales en la programación en 3D. Por ésta razón y dado que el estudio de aplicaciones gráficas en 2D está suficientemente explorado (aún a nivel nacional), nos proponemos en este proyecto incursionar principalmente en el desarrollo de aplicaciones gráficas en 3D.

2.2 El estándar mundial de redes 3G (ITM-2000)

[1] Existen algunas empresas nacionales que desarrollan aplicaciones y brindan servicios basados en dispositivos móviles (Ej.: GlobalNet – WazzUp brinda estadísticas obtenidas a través de votos con mensajes SMS a programas de la televisión Uruguaya); de todas formas, salvo algunos casos puntuales el desarrollo de aplicaciones se realiza hacia clientes del exterior, no explotándose mayormente el mercado interno.

El estándar ITM-2000 fue creado por la Unión Internacional de Telecomunicaciones (ITU), y en colaboración con otras organizaciones (3GPP, 3GPP2, UWCC, etc.), con el objetivo de valorar y especificar los requerimientos de las normas de celulares para los servicios de 3G, y unificar los sistemas de acceso inalámbrico a la infraestructura global de telecomunicaciones, tanto por medio de los sistemas satelitales como terrestres.

El resultado fue la familia de normas denominadas ITM-2000, cuyos principales objetivos son:
- Obtener eficacia operacional, en la transmisión de datos y servicios de multimedia.
- Brindar flexibilidad a los usuarios, uniformidad de diseño y transparencia en la provisión de servicio global de telecomunicaciones.
- Ofrecer un costo accesible, para expandir mundialmente el servicio de telecomunicaciones.
- Mejorar la calidad del servicio brindado, asimilándolo al de una red fija.
- Prestación de servicios por más de una red, en cualquier zona de cobertura.

Ésta norma garantiza la comunicación entre distintas redes y dispositivos móviles, contemplando desde la 1G a la 3G, y este tema se trata con más profundidad en el anexo *Tecnología Celular*.

2.3 Hardware

A continuación describiremos brevemente características generales del hardware de los dispositivos móviles, para ésta sección se ha consultado [18] , [23] y también se han utilizado algunas imágenes de ésos sitios.

2.3.1 Arquitectura

Se listan dos posibles arquitecturas que diferencian aspectos del tratamiento de gráficos; en la primera arquitectura (**Figura 2-1**) la resolución de los gráficos debe realizarse utilizando el procesamiento y memoria global del dispositivo.

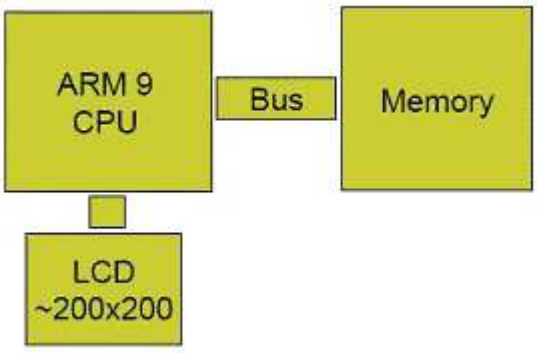


Figura 2-1: Arquitectura 1

En la arquitectura mostrada en la **Figura 2-2**, se cuenta con una unidad de procesamiento de punto flotante independiente y hardware grafico especifico, que funciona de forma independiente del procesamiento y memoria del dispositivo; este tipo de arquitectura permite acelerar notoriamente el rendimiento de las aplicaciones con componentes gráficos.

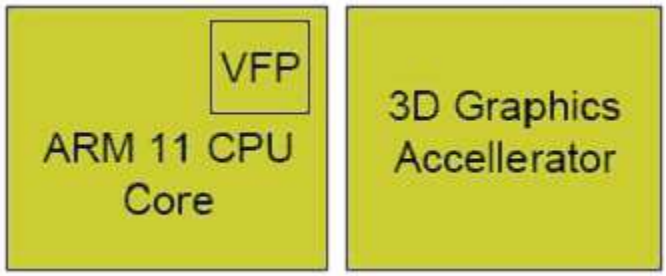


Figura 2-2: Arquitectura 2

2.3.2 Pantallas

La pantalla es menor en los dispositivos móviles que en los PC, pero se utilizan a una distancia más cercana de los ojos, por lo que se necesita mayor resolución para obtener la misma calidad. Algunas resoluciones de pantalla utilizadas en los celulares son:

Colores

Bits	**# colores**
16	65.536
18	262.144
24	16:777.216

Resoluciones
128 x 128
128 x 160
176 x 208
240 x 160
240 x 320 (QVGA)
352 x 416 (Doble resolución)

2.3.3 Procesador

Las marcas de procesadores predominantes en los dispositivos móviles son Intel y ARM. ARM es una familia de microprocesadores RISC y entre sus principales modelos se encuentran el ARM 7, el ARM 9 y el ARM 11. El ARM 7 es usado en los teléfonos de gama baja (con pocas prestaciones), el ARM 9 lo usan los de gama media que pueden realizar procesamiento de imágenes y el ARM 11 es usado en Smartphones, que pueden disponer de procesamiento de punto flotante y una unidad de procesamiento gráfico independiente. En [2] se puede encontrar una lista con los dispositivos móviles que utilizan ARM.

En [3] se menciona que el diseño de los procesadores ARM se utiliza en el núcleo computacional de mas del 85% de los celulares del mundo. Por ejemplo: Intel implementa este diseño y lo denomina Xscale, siendo su principal línea la PXA27x. En [14] se puede encontrar una lista con los dispositivos móviles que utilizan Intel.

2.3.4 Energía

El acceso a memoria es una de las operaciones más caras en cuanto a batería se refiere. En [1] y [7] se menciona que la tecnología que determina la duración de la carga de las baterías aumenta en un 9% anual, mientras las mejoras en el procesamiento lo hacen en un 40% anual. Esto muestra que la tecnología de las baterías no aumenta tanto como la de la resolución y procesador, por lo que es un factor limitante a tener en cuenta en el desarrollo de las aplicaciones. Aunque se disponga de un buen poder de procesamiento y resolución gráfica, el acceso a la memoria va a estar restringido por las limitantes de

autonomía dados por la batería del dispositivo, por lo tanto al desarrollar aplicaciones se tendrá que tener muy en cuenta este factor.

2.4 Principales servicios brindados por 3G

A continuación se describen brevemente los servicios más importantes brindados por los dispositivos móviles de 3G.

2.4.1 Servicio de Mensajería Multimedia (MMS)

MMS es un servicio disponible únicamente en las comunicaciones móviles de 3G (y algunas de 2.5G); permite reunir contenidos de texto formateado, imágenes, video y audio en un único mensaje, y obtenidos en tiempo real. Los mensajes se pueden enviar tanto a otros usuarios MMS, como a direcciones de correo electrónico vía Internet. Generalmente, los teléfonos móviles que soportan este servicio de mensajería, vienen equipados con cámaras de fotos. Este servicio comenzó a utilizarse comercialmente en Europa en Mayo de 2002, y posteriormente fue extendiéndose paulatinamente a otras redes distribuidas mundialmente.

2.4.2 Servicios de Mensajes Cortos (SMS)

SMS es un servicio que ya estaba disponible en 2G, se utiliza para enviar y recibir mensajes de texto a través de un Centro de Mensajes del operador telefónico, que asegura la recepción del mensaje una vez que el destinatario ponga en línea su receptor (teléfono móvil). Ésta tecnología fue introducida por primera vez en Europa en el año 1991 como parte de la especificación GSM. Posee la limitante de permitir el envío de 160 caracteres por mensaje.

2.4.3 Protocolo de aplicaciones inalámbricas (WAP)

WAP es un protocolo global y abierto que ya estaba disponible en 2G, aunque con algunas limitaciones respecto a la versión 2.0 utilizada actualmente en 2.5G (Ej.: inicialmente no se incluían imágenes, actualmente esto es posible). Este servicio permite el acceso e interacción desde un dispositivo móvil a Internet, independientemente del sistema operativo de dicho dispositivo. A diferencia del servicio SMS, WAP soporta una cantidad ilimitada de caracteres. El e-mail WAP además supera al e-mail/chat SMS, permitiendo crear grupos y carpetas para clasificar los mensajes. La mayoría de las terminales WAP, admiten la inclusión de gráficos, o pequeñas ilustraciones, aunque no en todos los casos son soportadas.

Comparado con el acceso habitual a Internet mediante HTTP, el protocolo WAP posee aún algunas limitaciones debidas principalmente a la menor capacidad en la velocidad de transferencia del medio inalámbrico a través de dispositivos móviles. De todas formas, algunas de las capas del protocolo WAP están basadas en la especificación de HTTP/1.1 y HTML, con las limitaciones mencionadas anteriormente, debido al medio de transmisión.

2.4.4 Juegos y multimedia para móviles

Dentro de los servicios de juegos y multimedia, los dispositivos 3G brindan al usuario la posibilidad de bajar desde su móvil, vía Internet: melodías y tonos para sus celulares, logos y fondos para celular, juegos, y videos multimedia para Celulares.

2.4.5 TV móvil

La TV móvil es una Industria que actualmente se encuentra en etapa experimental y que seguramente será incluida en forma definitiva en la 4G (aunque no hay mucha información aún, la ITU informa en su página web que la cuarta generación de dispositivos móviles ya se está investigando en Japón). Su finalidad es recibir señales de TV desde dispositivos móviles, pero aún no se ha establecido una norma Internacional que estandarice este servicio. Varias compañías han presentado sus propios estándares como ser la tecnología MediaFlo de la compañía Qualcomm, o la DVB-H de Nokia.

2.4.6 Programación para dispositivos móviles

Muchas de éstas funcionalidades, sobre todo las que requieren cierto grado de "inteligencia" como ser los juegos, están implementadas en lenguaje como Java, adaptados especialmente para dispositivos móviles (Ej.: J2ME para dispositivos móviles esta basado en J2SE con varias restricciones). Las prestaciones de éstos lenguajes se irán extendiendo paulatinamente con el avance de la tecnología, por ejemplo actualmente existe un sistema operativo orientado a los dispositivos móviles llamado SavaJe, que tiene por uno de sus objetivos garantizar la compatibilidad total con J2SE.

Se puede obtener mayor información sobre las características el estándar ITM-2000, las diferentes redes utilizadas en cada una de las generaciones de celulares y mayor información sobre los teléfonos celulares y PDAs en los anexos correspondientes: *Tecnología Celular* y *Tecnología PDA*.

2.5 Sistemas Operativos de dispositivos móviles

Se mencionan a continuación algunos de los sistemas operativos orientados a dispositivos móviles; algunos de ellos, debido a alguna de sus características principales, fueron estudiados de forma más exhaustiva en la etapa inicial de este proyecto. En ésos casos se adjuntan como anexos los documentos resultantes.

2.5.1 Symbian OS

Symbian es un sistema operativo con 32-bits, diseñado para teléfonos móviles y smart-phones, capaz de gestionar datos y aplicaciones. Debido a su amplia difusión comercial, soportando una gran cantidad de dispositivos de la marca Nokia, y también a que junto con PalmOS es uno de los sistemas operativos que soportan el desarrollo de aplicaciones en C++, se ha tomado este sistema como uno de los casos de estudio del proyecto, y por más información se puede recurrir al anexo: *Symbian OS*.

Algunas de las características generales de Symbian OS son:
- Integración de la tecnología móvil con la informática (Ej.: sincronización de correo electrónico entre el PC y dispositivo móvil, aplicaciones Web para dispositivos móviles, etc.).
- Ambiente de aplicaciones abierto (open application environment): permite que los teléfonos móviles sean una plataforma para desarrollo de aplicaciones y servicios, en una amplia gama de lenguajes.

- Interoperabilidad y normas abiertas: proporciona un conjunto de APIs y tecnologías que son incluidas en todos los teléfonos Symbian. Soportan los principales estándares de la industria.
- Multitarea: basado en una arquitectura micro kernel, e implementa multi-tasking y threading.
- Orientado a objetos: el sistema operativo ha sido diseñado desde su comienzo enfocado a dispositivos móviles, y utilizando técnicas avanzadas de programación orientada a objetos.
- Interfaz de usuario de diseño flexible: promueve la innovación y personalización del aspecto gráfico de los móviles, mediante su interfaz gráfica flexible y abierta.
- Robustez: el sistema asegura la integridad de datos, aún si la comunicación no es fiable y ante la escasez de recursos como memoria, almacenamiento y potencia del móvil.

La última versión es la 9.1 y sus características más importantes son:

- Protocolos de comunicaciones: soporta una amplia gama de protocolos de red, incluyendo TCP/IP (modo dual IPv4/v6), WAP 2.0, IrDA (infrarrojo), Bluetooth (comunicación inalámbrica de corto alcance, utilizada principalmente para sincronización con los PC) y USB.
- Telefonía Móvil: Symbian OS v9.1 está listo para el mercado 3G, soportando WCDMA (3GPP R4); intercambio de voz y datos a través de redes GSM (CSD, EDGE CSD, GPRS y EDGE GPRS); también soporta CDMA (IS-95 y 1xRTT).
- Plataforma de seguridad: mecanismo de la defensa de sistema proactivo basado en la concesión y monitoreo de capacidades de la aplicación. Infraestructura para permitir a las aplicaciones tener acceso privado a datos protegidos almacenados. Brinda además, servicios de encriptación y administración de certificados, protocolos de seguridad (HTTP, SSL Y TLS) y WIM Framework.
- Tiempo real: multithreaded kernel proporciona la base para la obtención de un teléfono robusto y eficiente.
- Soporte de Hardware: soporta las últimas arquitecturas de CPU, periféricos internos y externos, y los tipo de memoria externa disponibles actualmente.
- Rasgos específicos de CDMA: incluyendo roaming de redes CDMA, third party OTA API, NAM programming mode, CDMA SMS stack, NAI handset identification, interfaces para permitir IP móvil, y los modos de operación bridge y router gateway.
- Soporte Internacional: soporta Unicode Standard versión 3.0.
- Sincronización de Datos: soporta la sincronización Over-The-Air (OTA), que usa el estándar OMA; además permite la sincronización con los PC mediante Bluetooth, infrarrojo y USB.
- Desarrollo para Symbian OS: las opciones de desarrollo incluyen: C ++, Java (J2ME), MIDP 2.0, y WAP; existen herramientas disponibles para desarrollar aplicaciones C ++ y Java.

Arquitectura de Symbian 9

En la **Figura 2-3** se muestra un diagrama de la arquitectura de Symbian 9.

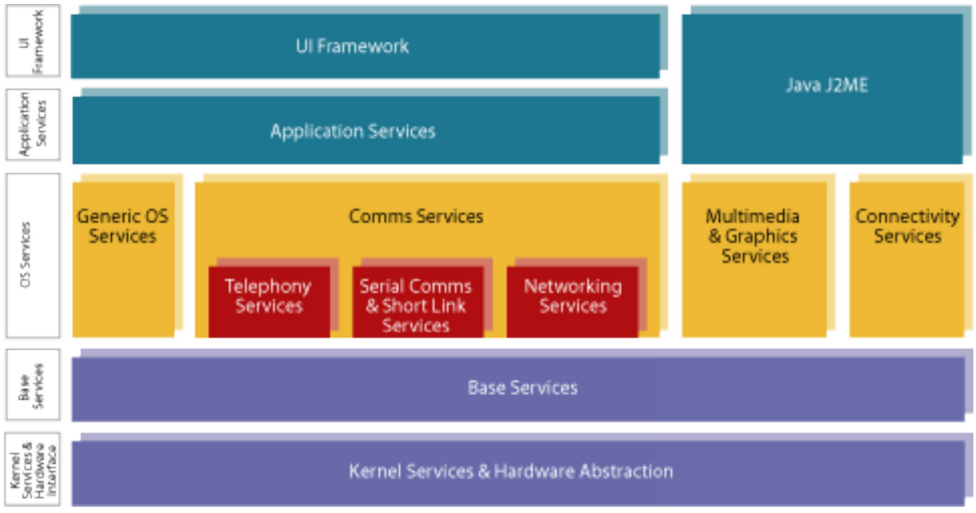


Figura 2-3: Arquitectura de Symbian OS 9

Plataformas de referencia de Symbian OS

Las plataformas de referencia, son interfaces gráficas de usuario basadas en Symbian OS, que brindan una serie de funcionalidades para el conjunto de dispositivos que las incorporan; facilitan la tarea de desarrollo de aplicaciones, ya que brindan soluciones para la creación de menús y formularios de ingreso de datos, además de permitir a los usuarios realizar la personalización de su interfaz. Mencionaremos las más difundidas. Esas plataformas son:

- **Series 60**: adoptada por las compañías Nokia, Panasonic, Samsung, Siemens, etc.
- **UIQ**: adoptado por las compañías Sony Ericsson, Motorola, BenQ, etc.

La elección de la plataforma de referencia, o interfaz, dependerá del tipo de dispositivo móvil sobre el cual ejecutará la aplicación a desarrollar, ya que las distintas compañías adoptan una u otra interfaz para sus distintos modelos de teléfonos. En cuanto a la Series 60 fue desarrollada por Nokia, y está disponible para los teléfonos de esa marca, y a través de acuerdos también para otras marcas como: LG, Panasonic, Samsung, Sendo y Siemens. La plataforma UIQ es una interfaz gráfica desarrollada por UIQ Technology, que ha sido adoptada por las compañías Sony y Motorola, para sus modelos que ejecutan Symbian OS.

APIs soportadas para gráficos

Symbian es un sistema operativo que permite el desarrollo de aplicaciones tanto en el lenguaje C++ como en J2ME, brindando una serie de APIs para diferentes funcionalidades. En el caso particular del desarrollo de aplicaciones gráficas, y en particular de 3D, soporta OpenGL ES (para C++) y M3G (para J2ME). Más adelante en este documento y en anexos, profundizaremos las características y funcionalidades de ambos lenguajes y también de las API correspondientes.

2.5.2 Windows Mobile

Windows Mobile es un Sistema Operativo desarrollado por la empresa Microsoft, pensado para utilizarse principalmente en dispositivos móviles como ser Pocket PC (nombre que Microsoft le da a los PDA que utilizan este sistema operativo), Pocket PC Phone Edition y Smartphones, que accedan a redes informáticas, bases de datos o necesitan sincronizar o acceder a información contenida en un PC con Sistemas Operativos desarrollados por Microsoft. La gran ventaja de este sistema radica en la posibilidad de abrir documentos creados con herramientas de oficina MS-Office, Bases de Datos SQL-Server, redes Microsoft y también permite el acceso a Internet, ya sea a través de WAP o http. Por ser un sistema orientado a plataformas Microsoft exclusivamente, no posee facilidades que permitan la reutilización de código C++ desarrollado en otras sistemas operativos, y aunque existen maquinas virtuales Java, el soporte de J2ME no está oficializado. Debido a la gran interconectividad entre los dispositivos Windows Mobile y otros productos desarrollados por Microsoft, el carácter sumamente propietario del sistema y una buena penetración en el mercado de Norte América, se considero este sistema como uno de los cuatro casos de estudios planteados, y se profundiza su estudio en el documento anexo correspondiente.

Las características más importantes de Windows Mobile son:

- Conexión a redes: el sistema Windows Mobile permite realizar conexiones a redes inalámbricas Wi-Fi, brinda soporte nativo para Bluetooth, y si el dispositivo móvil lo soporta permite la conexión a redes LAN a través de una tarjeta de red externa.
- Correo electrónico y mensajes: permite enviar y recibir correo electrónico, y en particular admite la integración con Microsoft Exchange 2003 (Contactos, Tareas, etc.).
- Multimedia: a través de Windows Media Player brinda acceso a una gran variedad de archivos multimedia: música, películas, imágenes. Dispone de servicios que permiten sincronizar el contenido digital desde un PC o Internet.
- Fotografías: permite almacenar, editar y desplegar fotografías (también tomar fotografías si el dispositivo lo permite).

- Soporte Microsoft .NET Compact Framework: brinda la posibilidad de utilizar las facilidades brindadas por esta plataforma de desarrollo de aplicaciones.
- Acceso a Internet: Windows Mobile no solamente soporta el acceso a páginas WAP, sino que también admite el acceso a páginas vía http. De esta forma, se puede acceder a cualquier sitio Web disponible en Internet, aunque pueden existir problemas de visualización, causados por la menor resolución de pantalla disponible en los dispositivos móviles, en comparación con la utilizada en los PC. Se puede además en las aplicaciones .NET prever el tipo de dispositivo que accede a la página, dejando disponibles distintas interfaces según el tipo de dispositivo utilizado para acceder a la misma (por ejemplo, la aplicación puede distinguir si el acceso se realiza desde un Pocket PC o desde un PC, mostrando distintas interfaces en cada caso). El acceso a Internet se realiza a través de la versión de Internet Explorer para Windows Mobile.
- Acceso a documentos de MS-Office: en la versión de Windows Mobile para Pocket PC y Pocket PC Pone Edition, es posible acceder a los documentos personales de MS-Office (Word, Excel, Power-Point). El acceso a los archivos puede hacerse a través de la red corporativa, Internet o accediendo a un PC mediante *ActiveSync*.
- Acceso a bases de datos: también es posible acceder a bases de datos SQL Server a través de la red, ésta facilidad se implementa a través de CE DB.
- Redes GPRS: tanto la versión Pocket PC Phone Edition, como la Smatphone soportan el acceso a redes GPRS.

La estrategia de Microsoft consiste en unificar la plataforma de desarrollo de aplicaciones, para TODOS sus sistemas operativos agrupándolos en el Visual Studio .NET. En el caso de las aplicaciones para dispositivos móviles actualmente, se generan con dos herramientas diferentes: *Embedded VC++* (aplicaciones Native) y *Visual Studio .NET* (aplicaciones Managed y Server Side). La herramienta de desarrollo Embedded C++ Permite escribir en código Visual C++, admitiendo solamente las clases soportadas por el sistema Windows Mobile; Microsoft va a discontinuar el uso de ésta herramienta, la cual será absorbida por Visual Studio .NET, que actualmente no soporta este lenguaje. Este cambio en la herramienta no implica una supresión del lenguaje Visual C++ (el cual seguirá utilizándose para crear las aplicaciones Nativas), sino que simplemente se realizará la unificación de las herramientas de desarrollo de aplicaciones.

APIs soportadas para gráficos

Windows Mobile brinda algunas API para el manejo de gráficos, como ser *Home Screen* que se utiliza para que el usuario pueda personalizar su interfaz gráfica; contiene varios modelos predefinidos, los cuales se pueden editar y modificar o crear nuevos diseños. También incluye la API *User interface/Shell*, que facilita las opciones de accesibilidad disponibles en la mayoría de los sistemas operativos Microsoft para PC y estaciones de trabajo, orientados a los usuarios con discapacidades; también provee algunas funcionalidades propias de los Pocket PC, como ser el manejo de la *Stylus*. La API *HTML Control* contiene todo lo necesario para la visualización de HTML con imágenes embebidas, es decir que desde los dispositivos Windows Mobile se puede navegar por Internet de la misma forma que se haría desde un PC.

Windows Mobile, debido a sus características de carácter propietario y cerrado, no soporta en forma oficial a la API OpenGL ES, ni tampoco al lenguaje de programación Java, a pesar de esto existen maquinas virtuales Java, implementaciones de M3G y de OpenGL ES desarrolladas por terceros.

2.5.3 Palm OS

Palm OS es un sistema operativo usado para PDA's y Smartphones. Cuenta con una cantidad superior a los 20.000 programas desarrollados, entre los que se encuentran aplicaciones profesionales para casi cualquier trabajo (legal, médico, inmobiliario, etc.), herramientas de productividad (bases de datos, correo electrónico, tratamiento de texto, hojas de cálculo, etc.), programas personales (mapas de viaje y traductores, asesores de salud y entrenadores deportivos, incluso aficiones como la astronomía) y ocio (juegos, libros electrónicos, música y vídeo). Debido a la amplia difusión de este sistema operativo dentro de

los PDA, se ha tomado este sistema como uno de los casos de estudio de este proyecto y su estudio se realiza en forma más exhaustiva en el documento anexo: *Palm OS*.

La principal versión es el Garnet (5.4). El principal cambio dado en la versión 5 fue la nueva plataforma de hardware. Se usan procesadores *ARM* de compañías tales como Intel, Motorola y Texas Instruments. Esto permitió sustanciales mejoras en cuanto a velocidad y capacidad de la plataforma y el software creado. Es importante destacar que el OS independiza al desarrollador de saber con que hardware trabaja, por lo que no se necesita recompilación de acuerdo a la versión del ARM usado. Se creo *PACE* (Palm Application Compatibility Environment) para poder desarrollar aplicaciones tanto para esta versión del OS, como para versiones anteriores sin necesidad de hacer cambios debido al nuevo hardware. PACE interpreta la familia de instrucciones de varios tipos de hardware y la hace correr en el procesador ARM.

Soporta pantallas de alta densidad como ser las de 320 X 320. El sistema permite a las aplicaciones ignorar la densidad definiendo coordenadas del 0 a 159 (ideal para las pantallas de 160 X 160); para pantallas de mayor resolución puede utilizarse igualmente este sistema de coordenadas. Dependiendo de si existe doble densidad será la cantidad de píxeles dibujados, por ejemplo si quiero dibujar una línea de (0, 0) a (5, 0), esto hará que en una pantalla de alta densidad se dibujen 10 píxeles, mientras que en una de baja densidad se dibujaran 5 píxeles. En caso de que se quiera usar la alta densidad explícitamente se provee una función de nombre WinSetCoordinateSystem() que permite que las coordenadas machen exactamente con lo píxeles en la pantalla.

También permite manejar pantallas con densidad 1.5x. El termino 1.5x se refiere a pantallas de 240 X 240, en contraposición a baja densidad que es 160 X 160 y doble densidad (o alta densidad) que es de 320 X 320. Como mencionamos también permite manejar pantallas de Quarter VGA, para esto como base se usa el soporte de 1.5x. Quarter VGA son pantallas de 240 X 320. Para soportarlas en esta versión de OS se divide en un área cuadrada de 240 X 240 mas un área de 240 X 80 que es la llamada dynamic input area.

La ultima versión de este OS es el Cobalt y es la evolución del OS Garnet y aprovecha plenamente las capacidades inalámbricas (WiFi/ Bluetooth). Esta previsto que a comienzos del 2006 comiencen a lanzarse dispositivos con este OS, por ahora se puede usar el simulador de esa versión para probar los programas.

En cuanto al desarrollo C++ es el lenguaje dominante, por lo que hay una gran cantidad de ayuda en foros, documentación, etc. No se necesitan instalar runtime enviroments por lo que la aplicación es mas pequeña y eficiente. Ejemplos de herramientas son desde GCC-Tools gratuitas (PRC-Tools) hasta entornos pagos como ser Metrowerks CodeWarrior. También se puede desarrollar en Java contando con varias herramientas.

APIs soportadas para gráficos

PalmSource introducirá la implementación OpenGL ES en el sistema operativo Palm OS Cobalt, permitiendo así la creación de PDA's que incorporen chips ensamblados con esta tecnología así como software con soporte de gráficos 3D y multimedia más avanzados.

2.5.4 SavaJe OS

SavaJe OS es un sistema operativo y una plataforma de aplicaciones para teléfonos móviles avanzados. Proporciona un gran conjunto de APIs y un conjunto completo de aplicaciones Java, además de una UI escrita que puede ser personalizada, y un juego completo de funciones de seguridad, incluso apoyo de codificación y certificado.

Es un sistema operativo completo y una plataforma de aplicaciones, esto significa que incluye un kernel y JVM (Java Virtual Machine) integrado, esto se diferencia de otras soluciones Java para móviles pues la JVM es una parte integral de la plataforma, que permite la ejecución eficiente del código de Java. Como Java es el lenguaje de aplicación para SavaJe OS, el JVM es "menos virtual" que en otras plataformas móviles.

SavaJe OS apunta a brindar todas las funcionalidades de **J2SE**, extendiendo las disponibles en **J2ME**; actualmente el sistema se encuentra en un punto intermedio entre ambas, pero se siguen agregando constantemente funcionalidades que se utilizan exclusivamente en J2SE. Eso es necesario recalcarlo ya que el conjunto de APIs con las que se cuenta es mayor al de cualquier otro Sistema Operativo que solo incluyen APIs del J2ME. Por este enfoque de compatibilidad con J2SE, se ha tomado como uno de los casos de estudio de este proyecto, en particular para crear aplicaciones portables, no sólo entre diferentes dispositivos móviles y sistemas operativos, sino entre los dispositivos móviles y PCs; se profundiza la descripción de este sistema en el documento anexo: *SavaJe OS*.

Además utiliza los componentes Swing de J2SE para el desarrollo de UI. Una ventaja significativa de este acercamiento consiste en que todas las aplicaciones de Java automáticamente adoptan el look and feel. El look and feel es la apariencia y comportamiento de los componentes gráficos de una aplicación. Esto permite una vista consistente a través de aplicaciones embebidas y aplicaciones de terceros descargadas, por ejemplo MIDlets.

SavaJe soporta todas las APIs de J2SE, entre las APIs disponibles en J2SE que no dispone J2ME están:
- Java Foundation Classes/Swing (J.F.C./Swing) components
- Abstract Window Toolkit (AWT)
- CORBA
- Java DataBase Connectivity (JDBC) API
- Jni network technology
- Java Remote Method Invocation (RMI)
- Java 2 Security Model

Este sistema adopta como una de sus metas la clásica frase de Sun: "Write Once, Run Anywhere", permitiendo que se escriba una aplicación y la misma corra en desktop y en dispositivos móviles. En noviembre de 2005 se lanzo el primer dispositivo creado por LG y en el 2006 se prevé su producción en masa. En diciembre de 2005 Group Sense PDA Limited anunciaron el lanzamiento de otro dispositivo.

2.5.5 Mobillinux OS

Fabricado por MontaVista, disponible para los teléfonos de las marcas Openwave, Motorola y Panasonic. Este sistema operativo se basa en el Kernel 2.6 de Linux, y tiene como características principales que optimiza el consumo de energía y aprovecha al máximo los procesadores. Además se debe considerar el hecho de que Linux es una plataforma de código abierto, por lo que podría generar la formación de una comunidad de desarrolladores para las aplicaciones móviles, similar a la que existe hoy días para las PC's. Aún no dispone del suficiente desarrollo desde el punto de vista del desarrollo de aplicaciones gráficas.

Características principales:
- Avanzado soporte de tiempo real.
- Entorno de desarrollo DevRocket.
- IPv6 y IPv4 , buen soporte de la TCP/IP Suite.

2.5.6 MotoJuix OS

Distribuido por Motorota, es una combinación de Linux con Java. Como sistema operativo no posee muchas funcionalidades, y no se proyecta como uno de los sistemas que vayan a permanecer en el mercado. Su característica positiva predominante es el buen sistema de alertas, útiles para los servicios de agenda.

2.5.7 Windows CE

Es un sistema de tiempo real con sus propias APIs para desarrollo y sus propios drivers para el hardware independientes de los otros Sistemas operativos Windows. Soporta múltiples arquitecturas de CPU y permite integrar los dispositivos móviles con Windows e Internet. Consume pocas cantidades de RAM, y soporta TCP/IP, PPP, y IrDA. Es el sistema antecesor de Windows Mobile.

APIs:
- Incluye mas de 500 de las APIs más utilizadas de Win32. Con estas APIs se crearon la versión de Word y Excel.
- Tiene Internet Explorer y el shell tradicional de Windows.
- Posee una maquina virtual Java y soporta Visual Basic Script.

2.6 Lenguajes de programación

A continuación se describen las características de los principales lenguajes de programación soportados por los diferentes sistemas operativos y dispositivos móviles.

2.6.1 J2ME

La plataforma J2ME (Java 2 Platform Micro Edition) es ofrecida a través de distintas API bundles llamados: *configurations (configuración), profiles (perfiles) y optional packages (paquetes opcionales)*. El ambiente de aplicación J2ME, incluye tanto la *configuración* CLDC, como el *perfil* Mobile Information Device Profile (MIDP). Además, se pueden agregar *paquetes opcionales*, que proporcionan la capacidad de brindar funcionalidades extra en las áreas específicas, como la mensajería inalámbrica y captura y reproducción de multimedia. La capacidad de escoger entre varios bundles permite a los diseñadores y desarrolladores de productos equiparar las capacidades de software con las capacidades del hardware. De esta forma, se pueden utilizar APIs permitan acceder fácilmente a funcionalidades específicas de dispositivos particulares, sin el overhead de APIs diseñado para otras funcionalidades que el dispositivo no soporta.

Una *configuración* proporciona un conjunto básico de bibliotecas y los rasgos que deben estar presentes en la máquina virtual para poner en práctica un ambiente J2ME. Define los requerimientos de la Java Virtual Machine y las clases principales. La *configuración* CLDC brinda una sólida plataforma Java, ideal para los dispositivos móviles, debido a que esta orientada a dispositivos con posibilidades limitadas en la interfaz, bajo poder de procesamiento, memoria limitada, etc. CLDC posee la máquina virtual Java KVM, preparada para procesadores de 16 y 32 bits RISC/CISC, y con memoria limitada a unos cientos de Kb.

Un *perfil,* es un conjunto de APIs estándar que soportan una categoría más estrecha de dispositivos dentro del marco de una configuración escogida. Un *perfil* específico es combinado con una *configuración* como CLDC para proporcionar un ambiente de aplicación Java completo y para la clase de dispositivo escogido. MIDP es un *perfil* diseñado para funcionar específicamente con CLDC, proporcionando en forma conjunta un rico ambiente de tiempo de ejecución. Las clases que agrega MIDP a CLDC son:
- Javax.microedition.midlet: para el ciclo de vida de la aplicación.
- Javax.microedition.lcdui: interfaz de usuario.
- Javax.microedition.rms: sistema de mantenimiento de registros.
- Javax.microedition.io: uso de redes.

En la **Figura 2-4** se puede observar la arquitectura del MIDP.

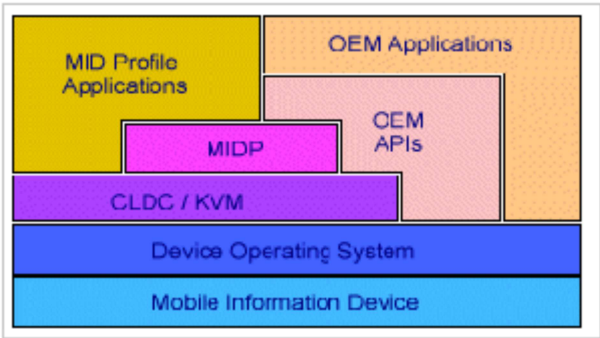


Figura 2-4: Arquitectura del MIDP

Los MIDlets son aplicaciones creadas para el perfil MIDP de J2ME, y que ejecutan directamente en el dispositivo móvil. Se brinda una descripción más detallada de J2ME en el documento anexo correspondiente.

2.6.2 C++

El lenguaje C++ es soportado enteramente por sistemas operativos como Symbian y Palm; la limitación aparece en las APIs disponibles, que son en definitiva la que dan el valor agregado importante a los desarrolladores de aplicaciones. Desde el punto de vista gráfico, la API de referencia para los dispositivos móviles es OpenGL ES, que es una versión adaptada de la librería abierta OpenGL. Se profundiza la descripción de la librería OpenGL ES en este documento, y también en el anexo: *OpenGL ES*.

2.6.3 Comparación C++ y Java

Lo que sigue es una comparación de características entre los lenguajes de programación C++ y J2ME.

Característica	C++	J2ME
Portabilidad entre sistemas operativos.	Soporta portabilidad del código, pero se necesitan adaptaciones en los proyectos para compilar la aplicación en distintos sistemas operativos.	"Write once, run anywhere": el código generado funciona de la misma forma, independientemente de la plataforma en que ejecute.
Portabilidad entre dispositivos.	En ocasiones es necesario recompilar el código para adaptar la aplicación a otros dispositivos con hardware diferente y no todas las características del hardware de un dispositivo pueden ser reutilizadas en otros dispositivos.	"Write once, run anywhere": el código funciona de la misma forma, independientemente del dispositivo en que ejecute. El código se adapta aún ante algunas diferencias de hardware, como ser la resolución de pantalla.
Código generado.	Compacto y nativo para cada dispositivo.	Menos compacto y requiere la existencia de una JVM para su ejecución; el código es portable a otros dispositivos.
Rendimiento.	Se obtiene un gran rendimiento en la aplicación, ya que genera código nativo para el dispositivo.	Rendimiento más pobre, genera código para la máquina virtual Java, la cual lo traduce a código nativo en tiempo de ejecución.
Aprovechamiento de las capacidades del dispositivo.	Se puede utilizar el 100% de las capacidades de cada dispositivo.	Sólo las disponibles en la máquina virtual Java.
Protección de memoria.	Debe implementarse en la aplicación.	Existen controles disponibles en el lenguaje.
Acceso a memoria.	Soporta acceso directo.	No soporta acceso directo.

A pesar de que en Java no se puede acceder a todas capacidades de bajo nivel del equipo, esta desventaja puede paliarse con algunas soluciones parciales, a costa de perder algunas de las ventajas de portabilidad del lenguaje. La solución consiste en que las funcionalidades que no son cubiertas por J2ME, se dejan liberadas a los fabricantes conocidos como OEM (Original Equipment Manufacturer) para que las implementen. Éstos fabricantes implementan APIs para acceder a las funcionalidades particulares de cada equipo y de esta forma explotar alguna de las características del hardware de determinado dispositivo, aunque perdiendo portabilidad en la aplicación resultante. En la **Figura 2-5** vemos una posible arquitectura que ilustra el caso mencionado.

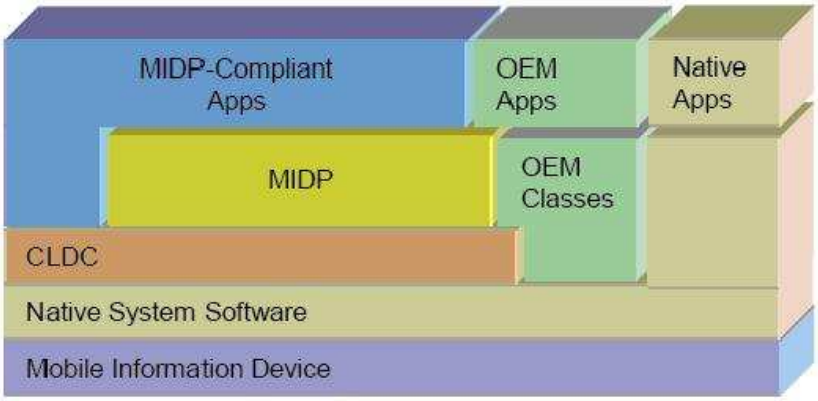


Figura 2-5: Arquitectura de APIs OEM

En la **Figura 2-6** se compara el rendimiento entre aplicaciones creadas en código nativo (generado a través de C++), J2ME, *Jazelle* y *HotSpot*. *HotSpot* es un maquina virtual altamente optimizada en relación a la KVM (máquina virtual Java estándar utilizada en dispositivos móviles). *Jazelle* son optimizaciones hechas al procesador para que pueda ejecutar Java bytecode nativamente en el HW. La comparación se realizó en operaciones de *vertex transformation* e *image downsampling*. El *Vertex transformation* es una de las operaciones más utilizadas en computación gráfica, que consiste en aplicar una o varias transformaciones combinadas (traslación, rotación, escalamiento, sesgo) a una geometría determinada. El *downsampling* de imágenes es reducir el tamaño de la imagen digital desechando partes de la imagen. Ambas operaciones requieren gran consumo de CPU y por ese motivo fueron elegidas para realizar la comparación de la **Figura 2-6**.

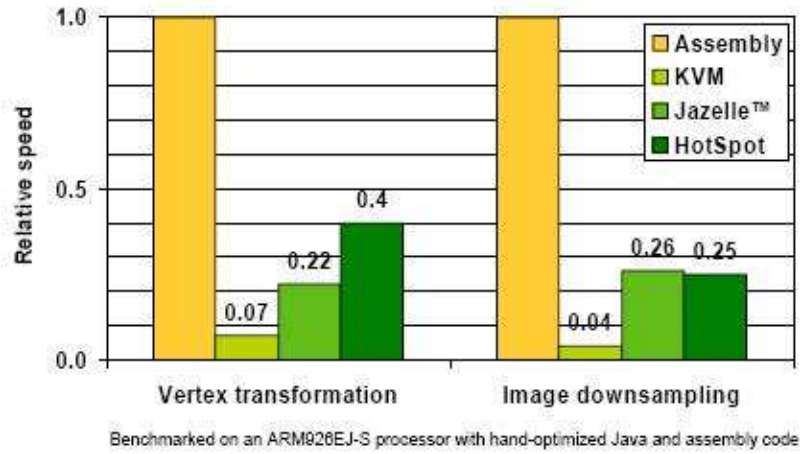


Figura 2-6: Comparación de velocidad del código nativo (Assembly, generado con C++) y distintas implementaciones de la maquina virtual J2ME (KVM, Jazelle, HotSpot).

El pobre rendimiento de las aplicaciones basadas en Java es un hecho que se da en los PC, y que se ve agravado en los dispositivos móviles debido a su menor capacidad de procesamiento. Las razones principales que originan este problema, es que ningún compilador puede resolver completamente estas cuestiones:

- Chequeo de tipos.
- Garbage collector.
- No hay acceso a las instrucciones SIMD de la CPU.
- Calls nativos desde las librerías Java built-in costosos.
- Maquina virtual basada en stack.

Las dos primeras son características built-in de Java por lo que no se pueden evitar. En cuanto al acceso a la CPU implica que no se pueden escribir rutinas críticas en ensamblador. Ya que el bytecode de Java esta basado en un stack, dificulta a la JVM compilar el código y transformarlo en código basado en registros como funciona la CPU, y esa es una de las razones principales del bajo rendimiento. Por éstas razones Java seguirá siendo más lento que el código nativo y a pesar de que esa brecha puede ir reduciéndose, nunca desaparecerá completamente.

C++ es considerado el lenguaje nativo por excelencia, apto para desarrollar software en los sistemas operativos más grandes y usados. Desarrollar una aplicación en código nativo implica que el archivo ejecutable que instalaremos en el equipo está expresado en código ensamblador entendible por el sistema operativo y por el procesador del equipo. Esto quiere decir que el compilador que utilicemos para desarrollar realizará la traducción a dicho código ensamblador.

En contrapartida al código de Java se lo conoce como *código manejado*; cuando trabajamos con este tipo de código, lo que se genera al compilar el proyecto no es código ensamblador entendible por el hardware y el sistema operativo del equipo, sino que es un código que es entendible por un aplicativo intermedio entre nuestro programa y el hardware, llamado Máquina Virtual. Esta máquina virtual interpreta el código manejado y lo convierte en tiempo real (Just in Time) a código ensamblador subordinado al sistema y hardware en que se encuentra.

Resumiendo todo lo dicho anteriormente, podemos enumerar las ventajas de un lenguaje respecto al otro de la siguiente forma:

Ventajas de Java respecto a C++

- Mayor utilización en el mercado, tanto desde el punto de vista de los dispositivos como de los sistemas operativos que lo soportan.
- Actualmente la difusión comercial de aplicaciones Java es muy superior respecto a C++[2].
- Incrementa la productividad:
  - Protección de memoria, tipos seguros -> menos bugs.
  - Menos bugs -> mejor productividad.
- "Write once, run anywhere": el código funciona independientemente de la plataforma en que corra, por lo que existe mayor compatibilidad con diferentes modelos y sistemas operativos. Con un solo proyecto y compilación podremos ejecutar nuestra aplicación en diversos sistemas y hardware. Hemos comprobado que para el código creado en C++, se necesitan hacer algunas adaptaciones, según el sistema operativo en el cual se va a ejecutar el código (Ej.: los proyectos Symbian en C++, no son iguales a los de Palm OS).

Ventajas de C++ respecto a Java

- Código más compacto.
- Aplicaciones más eficientes.
- Aprovecha la totalidad de los capacidades del hardware del dispositivo.

[2] Según [24] se vendieron 580 millones de teléfonos con soporte Java hasta febrero de 2005. En [25] la empresa Symbian manifiesta que se vendieron mas de 25 millones de teléfonos con Symbian OS hasta el 22 de marzo de 2005. De todos modos ha habido un gran aumento en estos últimos tiempos de dispositivos Symbian, alcanzando a fines del 2005 los 48 millones [26] y a fines del primer cuarto del 2006 los 70.5 millones [27] .

- Mejor y más amplio manejo de tipos de datos; por ejemplo en la aplicación de Laberinto en C++ la estructura de memoria que implementa las celdas del laberinto, ocupa la mitad de espacio que la correspondiente a la versión en Java, debido a que en C++ se pueden utilizar tipos de datos sin signo de un solo byte al cual se le pueden aplicar operaciones de corrimiento de bits, mientras que en Java éstas operaciones solo están reservadas para los tipos enteros con signo, que ocupan dos bytes de memoria.

Éstas conclusiones fueron tomadas en base a información recabada, para la cual hemos consultado [17] , [24] , [28] y nos hemos basado en la experiencia de los desarrolladores consultados, así como la nuestra. La conclusión es que cuando se necesita crear una aplicación de alto rendimiento y máxima funcionalidad es preferible implementarla en lenguaje C++; en cambio cuando la característica fundamental de la aplicación debe ser su portabilidad, es preferible implementarla en Java. La comparación entre el rendimiento de una aplicación creada en ambos lenguajes, no ha podido ser comprobada en este proyecto debido a la imposibilidad de conseguir un dispositivo Symbian OS o Palm OS, con los cuales se podría haber probado la aplicación Laberinto creada en C++, y compararla con la versión en Java (J2ME). La prueba de la aplicación en C++ solo ha podido realizarse en los emuladores, con lo cual no se ha podido determinar el rendimiento real de la aplicación.

2.7 La plataforma BREW

La plataforma BREW fue creada por la empresa QUALCOMM para facilitar el desarrollo de aplicaciones en dispositivos móviles; su principal componente es una capa de software, micro-programada en un chip que se incluye en los dispositivos BREW; esta capa no es considerada un Sistema Operativo de dispositivos móviles como Symbian, Palm OS o Windows Mobile, sino que simplemente implementa un conjunto de interfaces disponibles para los desarrolladores de aplicaciones y fabricantes de dispositivos móviles; de ésta forma, los desarrolladores pueden crear aplicaciones que se ejecuten en cualquier dispositivo BREW, independientemente de las características de hardware del dispositivo, su sistema operativo, o el fabricante que lo haya creado. El código de las aplicaciones BREW trabaja directamente con ésta capa abstracta, sin tener que manipular directamente el hardware de cada dispositivo, y por lo tanto las aplicaciones creadas son de un nivel de abstracción mayor que las aplicaciones nativas desarrolladas para un determinado dispositivo en particular.

Además de la implementación de la capa que interactúa con el hardware del dispositivo, la plataforma BREW incluye además una serie de productos y servicios enfocados a los distintos actores involucrados en la comercialización de las aplicaciones para dispositivos móviles: proveedores de contenido, fabricantes de dispositivos, operadores de telefonía y consumidores de las aplicaciones. Entre los componentes de la plataforma BREW se encuentran un SDK (Software Development Kit) con el cual se pueden desarrollar las aplicaciones BREW, un sistema de distribución (y venta) de aplicaciones, que además de utilizarse en los dispositivos BREW, permite la descarga de aplicaciones en dispositivos basados en sistemas operativos como Windows Mobile o Palm OS y que ejecutan bajo esos sistemas operativos. También se ofrecen varios productos más, que brindan un valor agregado a la plataforma.

Entre las principales características que se plantearon proporcionar en la plataforma, están la de ser:
- *Liviana:* ser una plataforma de tamaño reducido, capaz de ejecutar en dispositivos con poca capacidad (procesamiento, memoria, etc.), pero al mismo tiempo, explotar al máximo las capacidades del hardware.
- *Rápida:* permitir la creación de aplicaciones nativas de gran rendimiento (lenguajes C/C++).
- *Abierta:* incluye el soporte para Java.
- *Extensible:* BREW utiliza un sistema de interfaces que se encuentra en una capa superior, esto permite a los fabricantes utilizar extensiones para incorporar nuevas funcionalidades a los dispositivos. También se

ofrece un sistema de actualización, que permite ampliar o corregir las funcionalidades de las aplicaciones instaladas a través de la red telefónica.
- *Económica:* la empresa asegura que BREW disminuye los costos de desarrollo y acelera la introducción de los productos en el mercado.
- *Segura:* soporta varios servicios de autentificación y permite la inclusión de firmas digitales.

QUALCOMM al crear la plataforma BREW se enfoco en lo que denominó "cadena de valor" del mercado móvil. La llamada "cadena de valor" del mercado móvil, esta compuesta por una serie de actores, hacia los cuales QUALCOMM se centro brindando una parte de sus productos y servicios orientado a cada uno de ellos. Dichos actores se clasifican en:
- Consumidores de las aplicaciones.
- Editores y Desarrolladores de aplicaciones.
- Operadores de red telefónica.
- Fabricantes de dispositivos móviles.

Por más información sobre la plataforma BREW, se puede consultar el documento anexo: *La plataforma BREW*.

2.8 APIs para gráficos 3D

A continuación se describen brevemente las APIs gráficas predominantes de los lenguajes de programación utilizados para el desarrollo de aplicaciones en dispositivos móviles: J2ME y C++.

2.8.1 Mobile 3D Graphics (M3G) API para J2ME

También conocida como JSR-184 es una API para escribir programas con gráficos 3D, que consiste en aproximadamente 30 clases. Cabe aclarar que M3G no es Java 3D, ya que ésta extiende las capacidades del plataforma estándar diseñada para PC's, mientras que Mobile 3D Graphics esta orientada exclusivamente para dispositivos móviles.

M3G proporciona dos maneras para desarrollar gráficos denominados *immediate mode* y *retained mode*. En el modo inmediato (*immediate mode*) los comandos gráficos son enviados directamente al pipeline de gráficos y el motor de *rendering* los ejecuta inmediatamente. Usando este método el programador debe escribir código que específicamente diga al motor de *rendering* qué dibujar para cada frame de animación.

El modo retenido (*retained mode*) usa un *grafo de la escena* que actúa como un link entre todos los objetos geométricos del mundo 3D. Para esto se usa una estructura arborescente, la información de alto nivel de cada objeto como ser la estructura geométrica, la posición y apariencia es retenida de frame a frame. El *grafo de escena* es una estructura muy usada en las aplicaciones gráficas de 3D, se utiliza para determinar rápidamente las geometrías visibles de la escena y mejorar notoriamente el rendimiento de la aplicación, descartando el procesamiento de las áreas no visibles y eliminando la necesidad de decidir qué se debe dibujar y que no.

El modo inmediato puede ser visto como una API de bajo nivel, y el retenido como un API mas abstracta. El JSR 184 requiere la versión 1.1 de CLDC, en la **Figura 2-7** se muestra su relación.

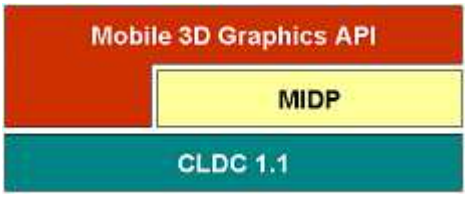


Figura 2-7: Arquitectura M3G

Además del API como mencionamos anteriormente el estándar JSR 184 también contiene un formato de archivo para manejar y desplegar contenido 3D de manera eficiente. Se definen archivos con extensión m3g que son transformados a las aplicaciones de modelado. Estos archivos se corresponden con un grafo de escena definido en el paquete para manejar y desplegar el contenido 3D como dijimos de manera mas eficiente o con simples objetos individuales. Esto permite crear contenido sobre PC's y luego ser cargados en los dispositivos móviles. Este contenido se crea con herramientas 3D, como ser Autodesk 3ds Max, Maya, Lightwave o Blender. Si se usa el modo retenido se carga la escena con una sola función, si se usa el modo inmediato es necesario extraer por partes el objeto deseado. La principal ventaja de exportar modelos 3D es que se pueden crear modelos mas complejos, pero para esta creación se necesitan mas habilidades de las que en general tiene un desarrollador, por lo que hay que trabajar en conjunto con artistas, lo cual hace que mejore la calidad final del producto.

Existen varias implementaciones como ser *Hybrid Graphics Gerbera*, *Superscape Swerve* y *HI Corp Mascot Capsule*.

- *Hybrid Graphics Gerbera*: esta implementación es construida encima del la implementación de OpenGL ES de Hybrid.
- *Superscape Swerve*: existen dos tipos el Swerve Client SR, útil para dispositivos sin aceleración gráfica de hardware y el Swerve Client ER que utiliza OpenGL ES para ejecutar muchas de las funciones de *render* requeridas en JSR 184, usando aceleradores de hardware dedicados del dispositivo por lo que solo es válido para dispositivos con aceleración gráfica de hardware.
- *HI Corp Mascot Capsule Micro3D:* consta de dos tipos, uno construida encima de OpenGL ES y otra para dispositivos sin soporte en hardware para gráficos.

Se puede obtener una descripción más completa de la API en el documento anexo: *Mobile 3D Graphics API para J2ME*.

2.8.2 OpenGL ES

OpenGL ES (OpenGL for Embedded Systems) es una plataforma de APIs para funciones completas de gráficos 2D y 3D en sistemas embebidos. Consiste en una serie de subconjuntos bien definidos de OpenGL, creando una poderosa y flexible interfaz de bajo nivel entre el software y los motores gráficos, especialmente útil para los dispositivos móviles.

Este conjunto de APIs estándares de gráficos 3D para sistemas embebidos, facilitan y hacen más económico el desarrollo de aplicaciones gráficas y juegos para muchas de las plataformas móviles existentes, sin la necesidad de utilizar otras tecnologías para la resolución de gráficos. Existen dos líneas básicas de desarrollo con OpenGL ES, denominadas perfiles 1.X y 2.X.

El OpenGL ES 1.1 está definido en relación al OpenGL 1.5, y con especial énfasis en la aceleración de hardware (API de aceleración de hardware), pero es totalmente compatible hacia atrás con el OpenGL 1.0. Proporciona muy buena funcionalidad, gran calidad de imagen y ofrece distintos grados de optimización para aumentar el rendimiento de las aplicaciones. Existe además, el denominado Extension Pack, que es una

colección de extensiones opcionales añadidas a OpenGL ES 1.1, que permite entre otras cosas, el desarrollo de aplicaciones 3D mas completas.

OpenGL ES 2.0 está definido en relación OpenGL 2.0, y pone énfasis en la programación 3D, con la capacidad de crear sombras, y escribir fragmentos de sombras en el lenguaje OpenGL ES Shading Language.

Una de las principales diferencias con su predecesor es que en caso de que no exista una unidad de punto flotante en el equipo, se puede hacer uso del tipo de datos punto fijo. Estos son representados internamente como enteros de 32 bits, los primeros 16 son la parte entera y los segundos 16 son la parte decimal. El tipo de datos se llama GLfixed. Esto no existía en la versión para desktops.

Entre sus principales ventajas se encuentran:

- Es un estándar de Industria y tiene Derechos Libres por lo que cualquiera puede bajar la especificación de OpenGL ES y implementar productos basados en el mismo. Al ser un estándar altamente aceptado permite al desarrollador concentrarse más en el contenido y menos en el código menor y detalles de plataforma.
- Previsto para bajo consumo de recursos
- Transición del software al hardware de *rendering*: Aunque el OpenGL ES define una pipeline de procesamiento de gráficos particular, las llamadas individuales pueden ser ejecutadas sobre el hardware dedicado, controladas como rutinas de software sobre la CPU de sistema, o puestas en práctica como una combinación tanto de hardware dedicado como de rutinas de software.
- Extensible y en constante evolución: OpenGL ES permite que las nuevas innovaciones de hardware sean accesibles por el API vía el mecanismo de extensión de OpenGL. Cuando las extensiones se hacen extensamente aceptadas, ellas son consideradas para la inclusión en el núcleo de OpenGL ES estándar.
- Fácil de usar: Como es basado en OpenGL, OpenGL ES es bien estructurado con un diseño intuitivo y ordenes lógicas.
- Altamente documentado: como OpenGL ES está basado en OpenGL, hay numerosos libros, y mucho código de muestra, haciendo la información sobre OpenGL ES barata y fácil de encontrar. Con la introducción de OpenGL ES, un desarrollador puede escribir ahora básicamente el mismo código para teléfonos celulares y computadoras de escritorio.

Sus principales implementaciones son Vincent que implementa OpenGL ES 1.1 y es valido para Windows Mobile y Symbian y Gerbera (Hybrid) que implementa OpenGL ES 1.X que soporta Windows Mobile y Symbian.

Se puede obtener una descripción más detallada de la API OpenGL ES en el documento anexo: *OpenGL ES*.

2.8.3 Comparación API's 3D

Éstas conclusiones fueron tomadas en base a información recabada, para la cual hemos consultado [19] , [23] y hemos considerado la experiencia de los desarrolladores consultados y la nuestra. También se han utilizado algunas imágenes de ésos sitios.

En la **Figura 2-8** se representa la arquitectura de un celular, en cuanto a las APIs 3D se refiere.

En primer lugar tenemos el SO, y el HW grafico, este ultimo opcional. Luego empezamos a diferenciar dos caminos que pueden ser complementarios o no. En primer lugar tenemos en el mundo C++, OpenGL ES y luego aplicaciones realizadas en C++. Por el lado de Java tenemos la JVM y encima M3G, a su vez M3G puede ser implementado encima de OpenGL ES.

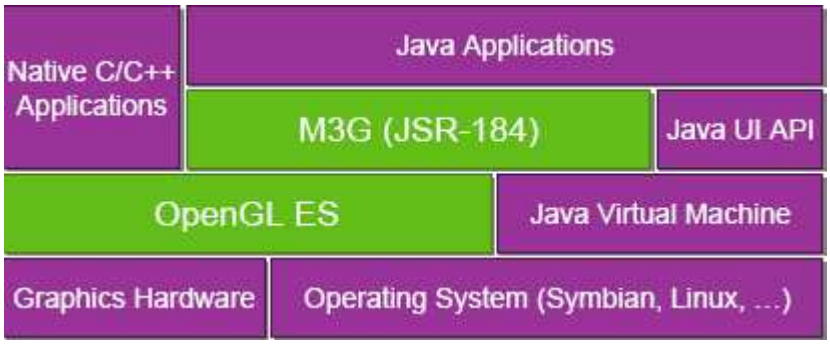


Figura 2-8: Arquitectura de APIs 3D

OpenGL ES podría ser llamada un API de bajo nivel, mientras que M3G seria de alto nivel. Las dos APIs fueron diseñadas para ser independientes, pero es importante que sean complementarias y puedan compartir recursos, por ejemplo usar el mismo motor de *rendering*, implementado en SW o HW.

M3G tiene prácticamente todo lo que se puede hacer en OpenGL ES, pero le agrega el *grafo de escena* (estructura en memoria utilizada en aplicaciones gráficas) y características para facilitar la animación, así como el formato .m3g.

El principal objetivo de M3G fue superar el pobre rendimiento debido a la JVM, para esto se creo un nivel de abstracción sobre OpenGL ES, permitiendo que la implementación sea mas optimizable, para minimizar la cantidad de código Java lento requerido para tareas de *rendering*. Ambas bibliotecas graficas aprovechan al máximo las capacidades del dispositivo.

La potencialidad de M3G actualmente es prácticamente la misma que con OpenGL, perdiendo un poco de control a bajo nivel, solución a futuro Java + OpenGL. (JSR 239: Java Bindings for OpenGL ES que permite acceder a la API OpenGL ES directamente desde un programa Java).

2.9 IDEs

Lo que sigue es una breve descripción general de las diferentes herramientas de desarrollo (IDE y SDK) que fueron estudiados en este proyecto para el desarrollo de aplicaciones para dispositivos móviles en los dos lenguajes en que nos hemos centrado. Para obtener más información sobre este tema, se debe consultar el anexo *IDEs*.

2.9.1 Java

IDEs y SDKs disponibles para el desarrollo con J2ME.

Sun J2ME Wireless Toolkit

Es imprescindible contar con esta herramienta para desarrollar en Java; los componentes de la herramienta son los siguientes:

- **KToolBar**: es una interfaz grafica que te permite seguir el ciclo completo de desarrollo de una aplicación, que consta desde la compilación, hasta la generación del jar final y su ejecución y debug en el emulador.
- **Emulador**: permite testear la aplicación.
- **Librerías**: son utilizadas durante la compilación, ya sea a través de la KtoolBar o por los IDEs.

En general todas estas actividades se hacen desde un IDE, que utiliza las librerías provistas por la Toolkit.

Eclipse

Para desarrollar aplicaciones orientadas a dispositivos móviles con este IDE se debe instalar además el plugin EclipseME, que permite la integración del IDE con las librerías y el emulador de la Wireless Toolkit; este IDE se define así mismo como un IDE para todo en general y para nada en particular [20] . Eclipse es un workbench (armazón) sobre el que se pueden montar herramientas de desarrollo para cualquier lenguaje, mediante la implementación de los plugins adecuados.

Su instalación se realiza a través del menú provisto por el IDE para instalar plugins. En primer lugar cuenta con dos wizard uno para crear un proyecto J2ME desde cero y otro para crear un MIDlet desde cero, esto es útil ya que automáticamente crea la estructura de la aplicación y el archivo jad. También permite la compilación y el empaquetamiento así como la ejecución y el debugging.

EclipseME es bastante simple y tiene las mismas utilidades del Wireless Toolkit, solo que le agrega un mejor debug y un par de wizards. Su poder radica principalmente en el uso del IDE en si, que al ser un plugin esta en constante avance por lo que se le seguirán agregando funcionalidades.

NetBeans

Es el entorno de desarrollo oficial de Sun para Java; para el desarrollo con J2ME es necesario instalar el Mobility Pack que se puede obtener del mismo sitio que el IDE. Esta pack ya viene con la ultima versión del *Sun J2ME Wireless Toolkit*, por lo que no es necesario instalarlo previamente como ocurre con otros IDEs; de todos modos se pueden elegir otros *Wireless Toolkit* diferentes para ser utilizados con *NetBeans*.

Una de sus características principales es el Diseñador Visual de pantallas y formularios, que permite crear el esqueleto principal del GUI de una aplicación utilizando la funcionalidad de "drag and drop". Entre los Wizards disponibles, encontramos:

- Crear un proyecto J2ME vacío.
- Crear un MIDlet vacío.
- Importar un proyecto del Wireless Toolkit.

También permite la compilación y el empaquetamiento así como la ejecución y el debug.

Borland JBuilder X Mobile Edition

Sus principales características son:

- Wizard de creación de proyectos y MIDlets vacíos.
- Editor visual de formularios.
- Refactoring.
- Multiples Wireless Toolkit.

Carbide.j

Este es una herramienta que se puede integrar como plugin a los siguientes IDEs.

- Eclipse
- NetBeans
- JBuilder

Sus principales características son:
- Diseñador de GUIs
- Diseñador de flujos de pantalla, que permite crear la lógica que conecta las distintas GUIs.

Conclusión

Wireless Toolkit se utiliza para hacer demostraciones del producto final y puede utilizarse incluso en maquinas que no dispongan de IDE. También es fundamental contar con él para las tareas de desarrollo, ya que además del emulador incorpora varias librerías que utilizan los IDEs.

Los IDE Eclipse y NetBeans se consideran muy similares en sus funcionalidades y generalmente cuando se liberan nuevas prestaciones en uno, son adoptadas rápidamente por el otro. NetBeans dispone de gran parte de sus funcionalidades desde su instalación, y por este motivo es considerado un IDE "pesado"; en cambio Eclipse dispone de las mínimas funcionalidades, y permite mediante la inclusión de plugins adaptarse a las necesidades del desarrollador. Algunas pequeñas diferencias son que NetBeans tiene acceso rápido a documentación Javadoc de clases, tiene un wizard para importar un proyecto del Wireless Toolkit. Por otro lado Eclipse tiene accesos rápidos para generar getters y setters. Ambos cubren todas las etapas que se deben seguir al desarrollar una aplicación y que son buscadas a la hora de elegir un IDE, como ser: escribir el código, compilarlo, depurarlo y ejecutarlo. En cuanto a la depuración y ejecución estos IDEs permiten integrar emuladores, tanto el estándar disponible con la Sun J2ME Wireless Toolkit, como emuladores de terceros (Ej.: emuladores de Nokia y Sony). La principal diferencia entre estos IDEs es que NetBeans tiene un diseñador visual de pantallas incorporado, mientras que Eclipse no. A comienzos de este año Nokia lanzo el plugin Carbide.j que se puede integrar tanto a Eclipse como a NetBeans, y que a parte de tener un diseñador visual de pantallas dispone de un diseñador de flujos que permite crear la lógica que conecta las distintas pantallas. En conclusión ambos son buenos IDEs, que pueden conseguirse de forma gratuita, con funcionalidades similares y fundamentales para maximizar la productividad; la elección entre uno y otro es en general producto de las preferencias personales del desarrollador.

JBuilder también es un buen IDE que dispone prácticamente las mismas funcionalidades que Eclipse y NetBeans, pero a diferencia de éstos es comercializado.

2.9.2 C++

Debido a que el formato de los proyectos de aplicaciones C++ dependen del sistema operativo sobre el cual se va a ejecutar la aplicación, los IDE de C++ están orientados a un sistema operativo en particular (Ej.: Visual Studio para Windows Mobile, CodeWarrior para Symbian OS).

En este proyecto nos hemos abocado al estudio de aplicaciones C++ para Symbian OS; las razones por las cuales se eligió ese sistema operativo se detallan en Aplicación: Laberinto Symbian.

A continuación describimos las principales características de los entornos de programación C++ orientados a Symbian OS.

CodeWarrior

Este IDE esta orientado exclusivamente para Symbian OS, manteniendo cualidades clásicas de otros IDE como ser *code completion* (escritura automática de código), formateo, búsqueda avanzada, rápido acceso a funciones. Hasta fines de febrero era el entorno por excelencia para desarrollar con Symbian y fuertemente patrocinado por Nokia; CodeWarrior es un IDE comercial, se puede descargar la versión trial que permite utilizarlo en forma gratuita por tres meses.

Tiene una gran variedad de wizards desde crear el esqueleto de una aplicación Series60 y para UIQ, hasta importar código escrito para Symbian hacia un proyecto CodeWarrior. Dispone además de un

poderoso debugger que demás de permitir ver el programa en código fuente despliega el código ensamblador, por lo que permite seguir el código instrucción por instrucción y comando por comando. Tiene vistas para variables, memoria y registros. Al igual que los IDEs para Java mencionados, cubre todas las etapas que se deben seguir a la hora de desarrollar una aplicación por lo que al igual que los otros permite seguir el *ciclo completo del desarrollo* que consta de las siguientes etapas:

- Escribir el código
- Compilar
- Debug en emulador
- Ejecución en emulador
- Generar el ejecutable final para el dispositivo

Borland C++ Builder X Mobile Edition

Otro IDE comercial, además de las cualidades clásicas de los IDEs, que incluyen básicamente los mismos wizards que CodeWarrior para la creación de esqueletos e importación de proyectos, su principal novedad y distinción es que tiene un diseñador visual de GUI, que es sumamente útil y ahorra bastante tiempo en la creación de pantallas y su lógica. Al igual que CodeWarrior permite trabajar con múltiples SDKs.

Carbide.C++

Este IDE esta disponible desde fines de febrero y se perfila como sucesor del CodeWarrior, el cual recordamos que es actualmente el IDE más difundido, además de ser patrocinado y recomendado por Nokia. Su ventaja sobre CodeWarrior es que es un IDE gratis con fines no comerciales y basado en Eclipse. Carbide.C++ posee prácticamente todas las funcionalidades del CodeWarrior.

Eclipse plug-in para Symbian C++

Es un plugin de Eclipse, escrito por Mikolajz en el año 2004, y ha ido evolucionando lentamente debido a que fue hecho por una sola persona. Su principal ventaja es que tiene disponible el código fuente para seguir mejorándolo y participar en el proyecto. Su principal desventaja es que no tiene un buen debuguer, fundamental a la hora de desarrollar para Symbian OS.

VistaMax IDE

Es un IDE basado en Eclipse y que como principal atractivo tiene un diseñador visual. Tiene disponible una versión no comercial. También tiene wizards para importación y exportar proyectos a otros IDEs como el CodeWarrior. Su principal desventaja es que tiene limitado el numero de objetos gráficos a usar en la versión no comercial, por lo que simplemente servirá para evaluación.

Carbide.vs

Es un plugin desarrollado por Nokia que permite de una manera fácil desarrollar aplicaciones para Symbian en el IDE Visual Studio, teniendo varios wizards de modo de que la configuración manual sea prácticamente nula. Estos wizards permiten crear una aplicación desde cero tanto como importar una existente desarrollada en otro IDE. Permite el desarrollo con múltiples SDKs y el uso de varios emuladores.

Conclusión

La compañía Nokia considera que Carbide es el primer IDE gratuito orientado a Symbian OS y con fines no comerciales basado en Eclipse. Sin embargo existen otros antecedentes, como el plugin de Mikolajz mencionado anteriormente, y el IDE VistaMax.

Respecto al desarrollo de GUIs, los IDEs se pueden clasificar entre los que lo soportan, y los que no; por ejemplo Borland soporta este tipo de desarrollo mientras que CodeWarrior no; con VistaMax se puede crear un proyecto con interfaz gráfica, y luego exportarlo a CodeWarrior para continuar normalmente con su desarrollo, aunque este mecanismo es artificial, y aunque no cambia en nada las características de

CodeWarrior respecto a la GUI, es una forma posible de desarrollar fácilmente una interfaz y trabajarla con CodeWarrior (aunque es necesaria la ayuda de otro IDE).

En lo referente a los wizards, se puede afirmar que tanto Borland como CodeWarrior poseen más o menos las mismas funcionalidades, y que su principal diferencia es en las configuraciones especificas para Symbian OS, ya que CodeWarrior esta especialmente diseñado para la filosofía de trabajo de ése sistema operativo y tiene más funcionalidades en ese sentido.

En cuanto a versiones no comerciales, tanto para aprendizaje o desarrollo de open source, el plugin de Mikolajz era una buena opción hasta fines de febrero; con el arribo de Carbide se abre una nueva posibilidad y con muchas mas funcionalidades que las presentadas por el plugin.

De todo lo expuesto consideramos que CodeWarrior es el entorno por excelencia para aplicaciones comerciales, y lo seguirá siendo por un tiempo mas, hasta que la nueva versión comercial de Carbide este disponible en este año. Para fines educativos y desarrollos de código abierto Carbide es la opción mas adecuada.

3. Desarrollo de aplicaciones

Se describen a continuación una metodología de desarrollo de aplicaciones orientadas a dispositivos móviles en los dos lenguajes de programación considerados.

3.1 Desarrollo de aplicaciones en Java

Lo que sigue a continuación es una descripción introductoria para el desarrollo de aplicaciones en Java para dispositivos móviles; los conceptos aquí indicados son válidos para cualquier tipo de aplicación a desarrollar.

3.1.1 Herramientas utilizadas

Para ésta sección se ha consultado [15] y también se han utilizado algunas imágenes de este sitio.

En la **Figura 3-1** se muestra un diagrama de capas con las herramientas utilizadas; el desarrollo de las aplicaciones se realiza en un PC que disponga del software indicado desde las capa 2 a 4, independientemente del sistema operativo del dispositivo en que vaya a ejecutarse la aplicación (capa 1).

Por detalles de uso y como obtener dichas herramientas ver el documento anexo *IDEs*.

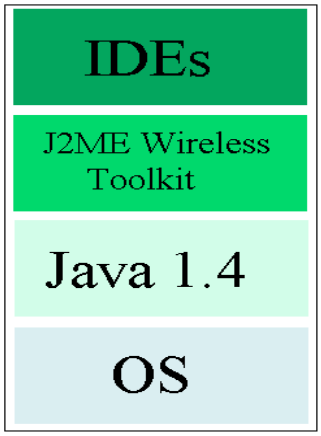


Figura 3-1: Capas de herramientas de desarrollo utilizadas

3.1.2 Estructura de una aplicación J2ME

Una aplicación J2ME está formada por un archivo JAR que es el que contiene a la aplicación en sí y un archivo JAD (*Java Archive Descriptor*) que contiene diversa información sobre la aplicación. Las etapas para construir una aplicación J2ME son:

- **Escribir el código**.
- **Compilar el código**: consta de dos sub-etapas; la primera es la compilación en si misma, o sea la generación de los .class. La segunda sub-etapa se llama Preverificación; este proceso asegura de que la clase sea valida para la configuración para la cual se este generando, por ejemplo si se genera para la configuración MIDP2.0/CLDC 1.0 y se usan flotantes, la preverificación va a fallar.
- **Empaquetarlo**: consiste en generar el archivo .jar, a partir del los .class. Existen dos opciones, la primera es crear un archivo .jar común para todo el código, y la segunda es crear un paquete denominado "Obfuscated Package". Este proceso de "ofuscación" tiene como objetivo detectar y remover clases, campos, métodos y atributos no usados, optimizar el bytecode y remover instrucciones no usadas. También renombrar los identificadores para que el .jar resultante sea mas chico y sea difícil de aplicar ingeniería reversa.
- **Ejecutarlo**: consiste en correr la aplicación en el emulador y posteriormente en el dispositivo.

Existe una aplicación de nombre JBenchmark que permite medir el rendimiento grafico de un dispositivo que soporte Java. Actualmente existen 4 versiones de la aplicación de acuerdo al perfil y HW grafico que soporte el dispositivo, por lo que tenemos una versión para dispositivos MIDP 1.0, otra para dispositivos MIDP 2.0, otra para dispositivos que soporten M3G y otra para dispositivos con HW grafico especifico para 3D. La aplicación se basa en correr de 4 a 5 tests según la versión y luego permite enviar los resultados al sitio Web de JBenchmark. En el sitio se encuentra una base de datos con todos los resultados recabados, lo que permite a los desarrolladores y consumidores medir y comparar las fortalezas y limites de una gran cantidad de móviles. Por ejemplo en el caso de la versión para dispositivos que soporten M3G los test ejecutados consisten en medir la cantidad de triángulos por segundo que se pueden desplegar para formar figuras 3D y en medir la cantidad de texels (unidad básica de una textura) y frames por segundo. En [16] se pede obtener mas información así como descargar la aplicación al dispositivo móvil.

3.1.3 Estructura de los fuentes

MIDlet

La aplicación Java se conoce como MIDlet, todo software debe tener por lo menos una clase que herede de MIDlet. La estructura básica es la siguiente:

```
import javax.microedition.midlet.*;

public class MiMidlet extends MIDlet {
    public startApp() {
        /**
         * Aquí incluiremos el código que queremos que el MIDlet ejecute cuándo se active.
         */
    }

    public pauseApp() {
        /** Aquí incluiremos el código que queremos que el MIDlet ejecute cuándo entre en
         * el estado de pausa (Opcional).
         */
    }

     public destroyApp() {
        /** Aquí incluiremos el código que queremos que el MIDlet ejecute cuándo sea destruido.
         * Normalmente aquí se liberaran los recursos ocupados por el MIDlet (Opcional)
         */
     }
}
```

El método que debe implementarse obligatoriamente es **startApp()**.

Pantalla

En la **Figura 3-2** se observan las clases que implementan el dialogo con la pantalla. La clase **Display** representa el manejador de la pantalla y los dispositivos de entrada. Toda aplicación debe poseer por lo menos un objeto Display. En los objetos Display podemos incluir tantos objetos Displayable como queramos. La clase **Displayable** representa a las pantallas de nuestra aplicación. La clase **Canvas** permite manejar eventos de bajo nivel y dibujar cualquier cosa por pantalla. Para dibujar en pantalla se usa un objeto de la clase **Graphics** que es el que contiene las herramientas gráficas necesarias. También existe la clase **GameCanvas** que es especifica para la implementación de juegos.

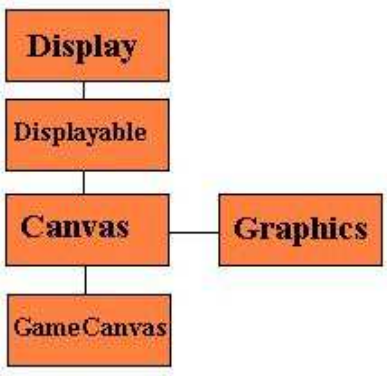


Figura 3-2: Clases de Pantalla

Teclas y menús

El manejo de los eventos del usuario se puede dividir en dos, en primer lugar está la interacción con las teclas y en segundo el manejo del menú.

Teclas

Esto se realiza con el método *getKeyStates()* de la clase, para esto se debe extender la clase GameCanvas. Este método retorna que tecla fue presionada, si tal es el caso, luego simplemente basta con comparar con el código de cada tecla para saber cual fue presionada.

Menús

Para el manejo de menús se debe implementar la interfaz CommandListener y sobrescribir el método *commandAction()*. Este método se llama cuando un menú fue seleccionado, por lo que en el mismo se procesan los comandos, recibiendo como parámetro de entrada el comando del menú seleccionado.

Loop de la aplicación gráfica

Este loop es basado en el uso de la clase GameCanvas. En primer lugar esta clase debe implementar la interfaz Runnable, para que pueda ejecutarse como un hilo. En la **Figura 3-3** se puede ver el funcionamiento del loop.

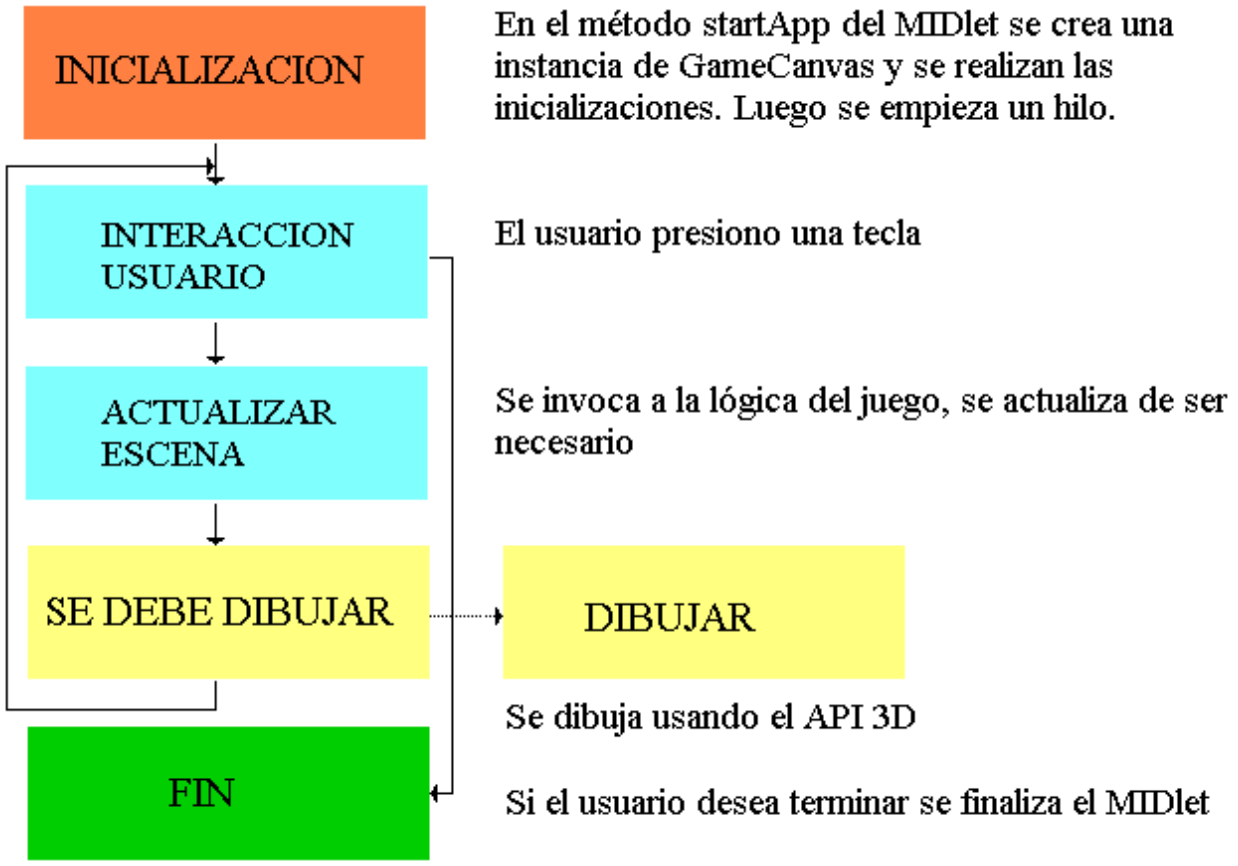


Figura 3-3: Loop de aplicación gráfica J2ME

Debido a que no siempre se debe refrescar el contenido mostrado en la pantalla, **se reduce la carga del procesador y extiende el tiempo de vida de la batería**.
Ésta es una de las recomendaciones de Apple para el desarrollo de interfaces de usuario para dispositivos portables, en este caso adoptado por J2ME para el desarrollo de aplicaciones orientadas a dispositivos móviles.

3.2 Desarrollo en Symbian OS

3.2.1 Herramientas utilizadas

En la **Figura 3-4** se muestra un diagrama de capas con las herramientas utilizadas. El Perl y la JVM se necesitan para correr los scripts que contiene el SDK. El SDK contiene las librerías necesarias y el emulador. Para desarrollar es suficiente con las cuatro primeras capas. Por más detalles sobre el uso y como obtener dichas herramientas ver el anexo IDEs.

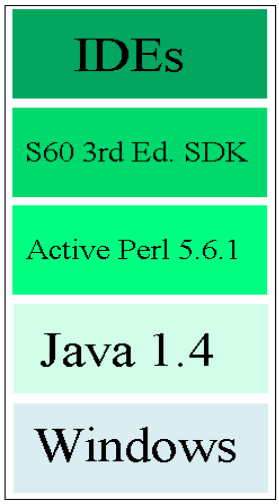


Figura 3-4: Capas de herramientas de desarrollo utilizadas

3.2.2 Estructura de una aplicación

Estructura base

Para ésta sección se ha consultado [22] y también se han utilizado algunas imágenes de este sitio.

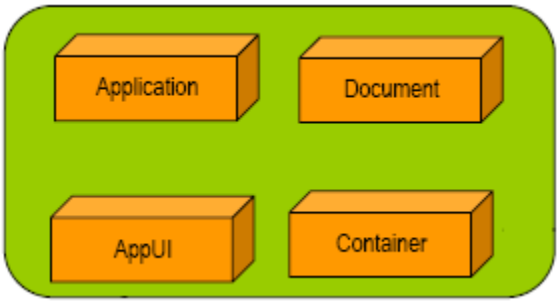


Figura 3-5: Symbian UI application architecture

Las aplicaciones Symbian deben tener estas cuatro clases, en general estas son generadas por los wizards de los IDEs. La misma se denomina "Symbian UI application architecture". En la En la **Figura 3-5** se puede observar como esta compuesta esta arquitectura.

- **Application**: es el punto de entrada de toda aplicación, desde aquí se crea el Document.
- **Document**: almacena el modelo para aplicaciones basadas en archivos, y desde aquí se crea el AppUI. Este clase en cuando a OpenGL se refiere, se usa solo para crear el AppUI, dado que no es necesaria la habilidad para editar documentos.
- **AppUI**: maneja los eventos del usuario. Desde aquí se crea el Container.
- **Container**: es la vista, lo que se ve en la pantalla.

Teclas y menús

El manejo de los eventos de menú se realiza sobrescribiendo la función HandleCommandL, que recibe el menú seleccionado.

El manejo de los eventos de teclas se realiza sobrescribiendo la función HandleKeyEventL, que recibe la tecla consumida.

Loop de la aplicación gráfica

En primer lugar existe una etapa de inicialización de las banderas de OpenGL. Luego se inicia un hilo en el que en primer lugar se espera por un evento del usuario. Si el usuario presiona una tecla, se invoca a la lógica del juego y se actualiza el estado de ser necesario. De acuerdo al estado se puede o no dibujar en pantalla. En el caso de que se debe dibujar se invocan a las primitivas de OpenGL ES dentro del método AppCycle del programa principal. En caso del que el evento de usuario corresponda a una terminación, se liberan los recursos y se finaliza el programa. En la **Figura 3-6** se muestra un diagrama del loop de la aplicación.

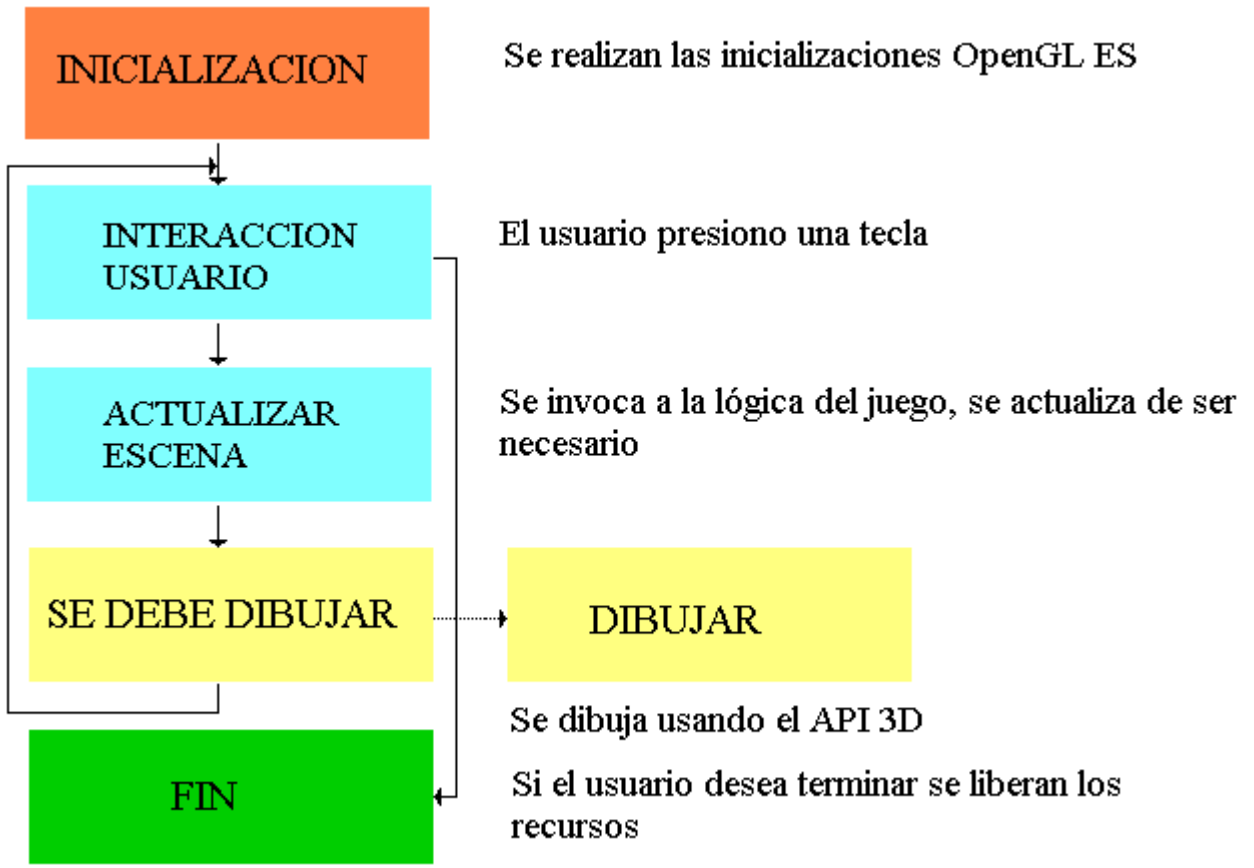


Figura 3-6: Loop aplicación gráfica Symbian

En este caso la interacción con el usuario es a través de teclas y no de menús.

3.2.3 Conclusiones preliminares

Para ésta sección se ha consultado [9] , [21] , [30] y [31] .

Complejidad

Symbian OS contiene cientos de clases y miles de funciones, básicamente tiene 1200 clases, las Series 60 le agregan otras 300 y las UIQ le agregan otras 700, por lo que es bastante difícil conocerlas todas. Para poder aprovechar su potencialidad (al menos gran parte de ella), es necesario procurar un previo entendimiento de las clases básicas.

Documentación

Existen partes del API que no esta muy bien documentadas y es un poco caótico en cuanto al orden ya que faltan muchas descripciones de funciones y sus argumentos, y hay poca cantidad de ejemplos con el API, por este mismo motivo se tiene que recurrir a ejemplos de otros desarrolladores. Si se compara con el MSDN de Windows, la documentación de Symbian deja mucho que desear y el sistema tiene una empinada curva de aprendizaje, por lo que para poder competir realmente se tiene que gastar un periodo prolongado de tiempo.

Manejo de excepciones

Symbian presenta un manejo de excepciones para aplicaciones C++ (el lenguaje no lo soporta explícitamente, sino que lo deja librado al programador). El sistema de manejo de excepciones de Symbian, esta basado en lo que se llama "cleanup stack"; en general las aplicaciones escritas en Symbian llevan mas tiempo que una tradicional en C++, esto es porque Symbian diseñó su sistema para aplicaciones que necesitan ejecutarse por un largo periodo en dispositivos con recursos limitados, y el precio que se paga por esa robustez que debe tener es el tiempo de desarrollo. No se usa el clásico TRY/CATCH, una excepción se denomina "leave"; si una leave ocurre se salta a un punto de código previamente definido con la macro "TRAPD". Un punto a tener en cuenta es que si una "leave" ocurre todos los objetos creados con new() que están el en heap quedan huérfanos, causando un fallo, para solucionar esto existe lo que se llama "cleanup stack", la idea es guardar dichos objetos en el cleanup stack, de modo que el sistema automáticamente los encuentre y los limpie si ocurre un leave. Esto implica tener que hacer ciertos push y pop en el cleanup stack se manera seguida, lo que puede llegar a causar errores de programación por olvidarse de hacer alguno. Existen funciones que implícitamente hacen push, es necearlo saber cuales son pues, luego de llamar a esa función se debería hacer un pop. Estas son llamadas las "asimetrías" del cleanup stack.

Constructores de clases

El proceso de construcción de clases se llama "two-phased construction", esto se debe a que se intenta prevenir que en la creación de objetos sucedan excepciones, por lo que se divide el proceso en dos partes.
La primera es la instanciación del objeto, reservando memoria para el mismo. Se usa el método "*NewL*". La segunda fase es cuando se inicializa con datos. Se usa el método "ConstructL".

```
clase CTest:

NewL(parámetros){
        CTest()
        ConstructL(parámetros)
}

ConstructL(parámetros){
        asignar_parametros
}
```

Strings

No existe la clase String propiamente dicha, la implementación de Strings es a través de lo que Symbian llama "Descriptores", la justificación es por un asunto de robustez, economía y eficiencia. Existen varios tipos de descriptores de acuerdo a la función del API que se use; la razón es usar el mínimo preciso de memoria necesario para el almacenamiento según la API que se esté usando.

4. Aplicaciones desarrolladas en el proyecto

Se detallan a continuación las características principales de las aplicaciones desarrolladas en este proyecto en sus distintas versiones, detallando los problemas encontrado y las herramientas utilizadas para su implementación.

4.1 Aplicación Laberinto

El objetivo de ésta aplicación es explorar las prestaciones de 3D en dispositivos móviles ejecutando en tiempo real, teniendo en cuenta las limitaciones propias del hardware disponible, y la escasa resolución de pantalla que presentan este tipo de dispositivos. También se intenta crear una aplicación que sea portable a la mayor cantidad de dispositivos posibles, sin tener que realizar mayores cambios o adaptaciones en la aplicación para los distintos modelos de dispositivos.

Por tales motivos, el diseño de ésta aplicación se realizó considerando las siguientes condiciones de hardware mínimas (soportadas por la gran mayoría de los dispositivos con capacidad para ejecutar este tipo de aplicaciones):

- Disponer de al menos 1 Mb de memoria RAM.
- Carencia de dispositivos de hardware gráfico con capacidad de procesamiento independiente.
- Resolución de pantalla de 128 x 128.
- Para el desarrollo de la aplicación en C++, debe soportar la biblioteca gráfica *OpenGL ES.*
- Para el desarrollo con J2ME, debe soportar el perfil / configuración: *MIDP 2.0/CLDC 1.1* y la biblioteca gráfica *Mobile 3D Graphics API*.

Las elecciones de las condiciones del hardware, se basan en las características mínimas que disponen los diferentes dispositivos que pueden correr aplicaciones gráficas. La elección de las bibliotecas gráficas se realizó por ser las más difundidas en sus respectivos lenguajes de programación, y además porque son soportadas por una gran cantidad de dispositivos; por ejemplo el Open GL ES es soportado en todos los dispositivos con Symbian OS, y también en los que utilizan el sistema Palm OS versión Cobalt y Windows Mobile; entre éstos sistemas obtenemos gran parte de los dispositivos que utilizan C++ como lenguaje de programación. La biblioteca Mobile 3D Graphics API, es soportada por la totalidad de los sistemas operativos estudiados que se basan en J2ME con perfil / configuración MIDP 2.0/CLDC 1.1, el cual es la configuración ideal para el desarrollo de aplicaciones 3D. Además esa versión de configuración es la última disponible, y es la única que soporta punto flotante en la actualidad (requisito importante para el desarrollo de aplicaciones 3D); esta configuración será la que incluirán los nuevos modelos de celulares que se liberen, además de que ya existen varios modelos disponibles que la soportan. Si bien la versión anterior (CLDC 1.0) se encuentra más difundida en los dispositivos comerciales que la versión CLDC 1.1, para el desarrollo de ésta aplicación debemos utilizar este último, debido a que el anterior es más limitado para el desarrollo de aplicaciones, no soportando la API 3D. Por ejemplo, los modelos importados de Blender que se utilizan para desplegar los personajes del juego en su versión Java, requieren la utilización de punto flotante. También hay algunos de los métodos utilizados en la aplicación que requieren punto flotante, por ejemplo el seteo de la perspectiva de la cámara.

Debido a que las nuevas versiones de los perfiles / configuraciones MIDP/CLDC garantizan la compatibilidad con las versiones anteriores, ésta aplicación no solamente será soportada por la última generación de dispositivos que se comercializan actualmente, sino también por las generaciones futuras que soportes nuevas versión del perfil.

4.1.1 Descripción de la aplicación

La aplicación desarrollada consiste en un juego, en el cual existe un personaje comandado por el usuario que se desplaza dentro de un laberinto; la vista del juego es en 3D, con un observador ubicado detrás del personaje (al cual siempre ve de espaldas), realizando los mismos giros que el personaje principal, y trasladándose junto con el cuando pasa de una celda a otra del laberinto. Existen además una serie de personajes (enemigos), que se desplazan de forma automática dentro del laberinto intentando perseguir al personaje principal. La estrategia de persecución es de implementación sencilla, pero funciona de un modo aceptable, aunque no es lo suficientemente elaborada como para los enemigos encuentren siempre el camino mas corto que lo conduzcan a la posición del personaje principal. El énfasis de la aplicación esta puesto en los aspectos gráficos, el tiempo real, y la portabilidad de la misma, dejando de lado aspectos que podrían hacer atractiva a la aplicación desde el punto de vista comercial, y ésta es la razón por la cual no se confeccionó una estrategia de persecución mas elaborada.

El laberinto se divide en "celdas" que son espacios cúbicos de tamaño fijo; en cada celda puede haber de cero a cuatro paredes. En la **Figura 4-1** se muestran nueve celdas adyacentes, ordenadas en tres columnas y tres filas. El piso del laberinto forma parte de cada celda del mismo; los actores (personaje principal y enemigos) se desplazan sobre el piso pasando de una celda a otra. Las paredes del laberinto se consideran como parte de las dos celdas adyacentes; el paso de los actores de una celda a otra es obstruido si entre ambas celdas hay una pared. En la **Figura 4-1** también se pude ver que la celda central contiene cuatro paredes, compartidas con las celdas centrales de las filas y columnas respectivamente.

El tránsito de los actores desde una celda a la otra es libre en caso de no haber una pared entre ellas; además los actores pueden desplazarse dentro de la celda ocupando sesenta y cuatro posiciones posibles sobre su piso; en el anexo *Aplicación Laberinto* describiremos más detalladamente los movimientos dentro de una celda, y otras características importantes referentes a la implementación de la aplicación.

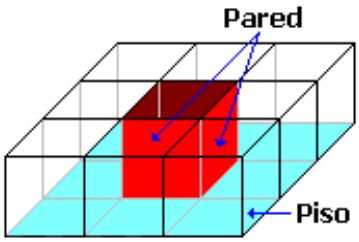


Figura 4-1: Estructura de celdas de la aplicación Laberinto

Mencionaremos algunas de las técnicas habituales utilizadas en las aplicaciones graficas que fueron incluidas en el desarrollo de ésta aplicación, y que también se explican con mayor grado de detalle en el documento anexo: *Aplicación Laberinto*. Las técnicas empleadas son:

- **Determinación de superficies visibles**: gran parte del trabajo de determinar las superficies visibles, fue realizado en forma estática, e incluida en el código de la aplicación para evitar cálculos en tiempo real que pueden atentar contra el desempeño de la aplicación, obteniendo un código más eficiente.
- **Ordenamiento espacial**: se utilizan técnicas de aprovechamiento de coherencia espacial, para reducir el procesamiento de la detección de colisiones, y determinación de los objetos visibles en la pirámide de vista (porción visible de la escena). Estas técnicas reducen la cantidad de casos a estudiar, y aumentan el rendimiento global de la aplicación.
- **Técnicas de *culling* (recorte de caras ocultas)**: se evitan dibujar paredes ocultas, y gran parte de los actores que no están visibles en un momento dado en el juego, reduciendo de esa forma el trabajo que debe realizar la biblioteca gráfica, y obteniendo un algoritmo más eficiente.
- **Volúmenes acotantes**: para la detección de colisiones entre objetos dinámicos, se implementaron los volúmenes acotantes de los actores del juego. Los volúmenes acotantes contienen espacialmente a los actores del juego, y es mucho más económico (desde el punto de vista del procesamiento) realizar el cálculo de colisiones entre los volúmenes acotantes, que entre los volúmenes ocupados por el cuerpo de los actores.
- **Sistema de control de las colisiones**: controla que ninguno de los actores del juego atraviesen las paredes del laberinto, ni que existan colisiones con otros actores del juego.
- **Nivel de detalle (LOD)**: bajo ciertas circunstancias se eliminan parte de la escena visible para obtener un algoritmo de mayor rendimiento, intentando que la pérdida de la información afecte en la menor medida posible el realismo de la escena.
- **Texturas**: las paredes y piso del laberinto se dibujan en base a diferentes texturas 2D, que permiten cambiar la apariencia de la escena.
- **Técnicas de aceleración de *rendering***: si bien la capacidad del hardware de los dispositivos (en particular la escasa memoria RAM) no favorece el uso de complicadas estructuras de datos (Ej.: Grafos de escena, Octrees, etc.), se puso especial énfasis en el diseño de la aplicación en el rendimiento de la misma en tiempo real.

4.1.2 Construcción de la pirámide de vista

La *pirámide de vista*, contiene la porción del espacio que es visible en la escena; es importante conocer la elección de esta perspectiva para entender algunas de las decisiones que fueron tomadas en el diseño e implementación de la aplicación. El espacio visible en la *pirámide de vista* esta delimitado por cuatro planos que se unen en un vértice donde se ubica el ojo del observador. De los cuatro planos que determinan esa pirámide, dos son laterales a la vista del observador (izquierda y derecha respectivamente), uno superior y otro inferior. En la **Figura 4–2** vemos una perspectiva de los cuatro planos que forman la *pirámide de vista*; esta pirámide a su vez está limitada por el *plano de vista posterior*, que indica el límite máximo que puede alcanzar la vista del observador.

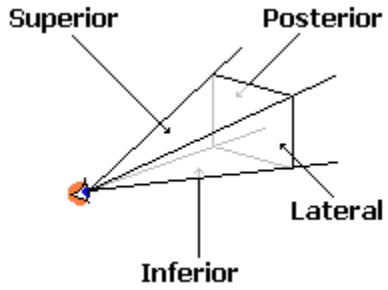


Figura 4-2: Pirámide de vista y plano posterior

El espacio visible esta delimitado además por el *plano de vista anterior*, que excluye de la escena a todos los objetos que se encuentran anteriores a él. Tanto el *plano anterior* como el *posterior* se encuentran en la dirección hacia donde mira el observador, siendo el *plano de vista anterior* el que se encuentra en la posición más cercana a este. De esta forma, el espacio visible de la escena se ubica dentro de un cuerpo espacial similar al mostrado en la **Figura 4-3**; a este cuerpo espacial es al que denominamos *pirámide de vista*, y como mencionamos anteriormente, contiene a todos los objetos o porciones de objetos que serán mostrados en la escena.

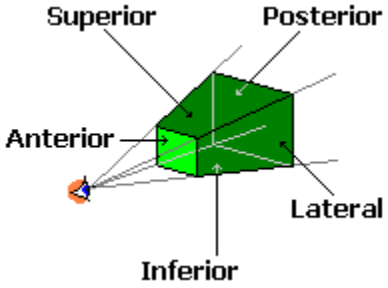


Figura 4-3: Pirámide de vista y planos anterior y posterior

Pasemos ahora a la descripción de la pirámide de vista en la aplicación Laberinto. Como mencionamos anteriormente, el observador de la escena esta ubicado detrás del personaje principal (al que siempre ve de espaldas), y puede ver hasta tres columnas de celdas (en la central siempre estará el personaje principal), con una profundidad máxima de hasta cuatro celdas. Es decir que para procesar el contenido de la *pirámide de vista* se analizan, a lo sumo las paredes de doce celdas del juego, lo que implica un acotamiento en la cantidad de celdas a procesar para la construcción de cada imagen.

En la **Figura 4-4**, vemos una vista superior de una porción del laberinto, donde las celdas se visualizan como cuadrados. El punto ubicado en la tercera fila y tercera columna indica la posición del personaje principal dentro del laberinto. El observador está ubicado detrás del mismo; el *plano anterior de la vista* está ubicado en el inicio de la celda en la que se encuentra el personaje, mientras que el *plano posterior* se ubica al final de la cuarta celda desde la posición del personaje principal. En la **Figura 4-4** también apreciamos que la *pirámide de vista* (indicada como **Área vista**) sufre un recorte adicional, dado por la restricción propia del programa, de mostrar únicamente tres columnas de celdas (lo que aparece indicado como **Área eliminada**).

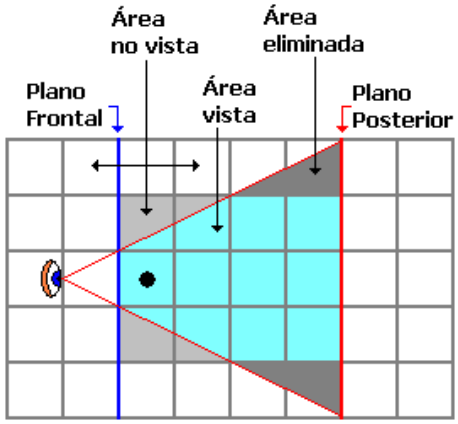
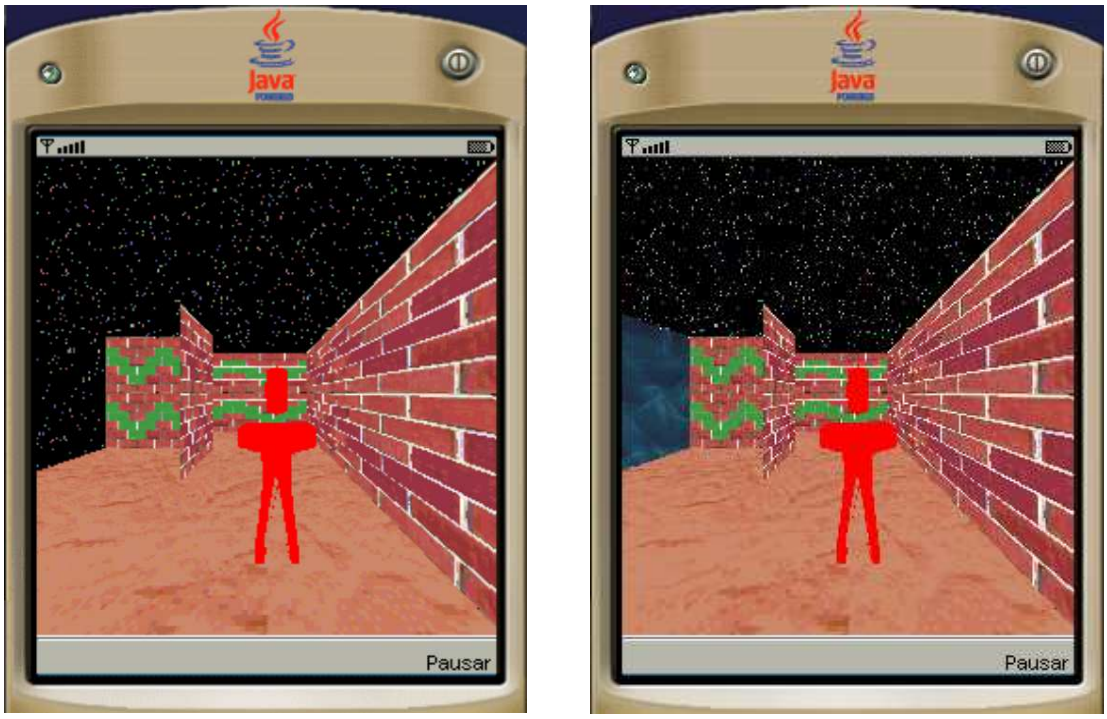


Figura 4-4: Pirámide de vista y su interacción con el laberinto

Si bien de esta forma el campo visual se ve restringido (**Área eliminada**), se logra una importante simplificación en la lógica del algoritmo y junto con otras medidas que se implementaron, se asegura un procesamiento acotado para la escena. En las aplicaciones gráficas es habitual utilizar técnicas de "aproximación" a la realidad (aún contando con hardware mucho más potente y especializado para aplicaciones gráficas en tiempo real); por ejemplo, en ciertas ocasiones los desarrolladores evitan dibujar ciertos objetos que no aportan información relevante a la escena por estar a una distancia muy grande del observador; de esta forma se obtiene un algoritmo con mejor rendimiento, aunque teniendo especial cuidado en balancear el rendimiento con el realismo de la escena mostrada. En este caso, se considera que las porciones de celdas descartadas además de ser lejanas al observador, en la mayoría de los casos permanecen ocultas detrás de otras paredes del laberinto, y en los casos que aparecen visibles aplicaremos algunas medidas paliativas que mencionaremos a continuación.



(a) **(b)**

Figuras 4-5: Laberinto sin pared lateral (a) y con pared con textura de niebla (b)

En la **Figura 4-5-a**, vemos el caso particular en que se genera un "hueco" correspondiente al **Área eliminada**. Este es uno de los casos particulares que aparecen esporádicamente, debido a que la mayor parte del tiempo éstas celdas estarían ocultas detrás de otras paredes. La mejor solución para este problema, sería procesar cuatro celdas más (correspondientes al **Área eliminada** de la **Figura 4-4**), y aplicar a esas celdas una técnica de *culling* (recorte de objetos ocultos) similar a la que se implementó en forma estática para las tres columnas de celdas mostradas. Sin embargo, se optó por no implementar ésta solución debido a que solamente requeriría ampliar el árbol de casos de recortes considerados (que actualmente son ochenta y seis), sin aportar ningún beneficio extra a la técnica de *culling* implantada; los ochenta y seis casos de *culling* mencionados se pueden ver en el anexo: *Aplicación Laberinto*.

Si ésta fuese una versión comercial de un juego, sin duda que la solución elegida sería ampliar el conjunto de casos para contemplar las cuatro celdas excluidas. Optamos en cambio por una solución paliativa al efecto de "abismo" o "hueco" que se muestra en la **Figura 4-5-a**, agregando una pared con una textura de "niebla" que atenúa ese efecto, tal como se muestra en la **Figura 4-5-b**. Cabe aclarar además, que los posibles enemigos que pudiesen estar en la posición descartada, tienen una incidencia mínima en la persecución del personaje principal, ya que deben atravesar al menos tres o cuatro celdas más para llegar a la posición del personaje, y pasarían a estar visibles al atravesar una o dos celdas respectivamente, dando la posibilidad al personaje de huir, contando con al menos dos celdas de separación de su enemigo.

4.1.3 Aplicación: Laberinto Java

A continuación mencionaremos aspectos de la aplicación, que son exclusivamente referentes a su implementación en Java.

Arquitectura

En la **Figura 4-6** se muestra la arquitectura de la aplicación. A continuación la describiremos en mayor detalle:

- **Files**: Modificación de la utilidad PTUtilities, que permite leer archivos de formato .INI y cargarlos en un hash para luego obtener el valor a partir de una clave. La modificación consiste en extender la maquina de estados de dicha clase y agregar una nueva estructura de datos para permitir leer archivos .INI que además del valor, incluyan la sección. Se puede obtener PTUtilities de [5] .
- **Gus**: Es una clase Java generada a partir de un modelo para Blender (utilidad para generar modelos animados de objetos en 3D). Se puede obtener el modelo Gus en Blender de [29] .
- **MIDlet**: es punto de entrada de la aplicación.
- **Lógica**: Contiene la lógica del juego.
- **Pantalla**: en esta clase se crean los objetos 3D, y se dispone de métodos para su dibujado.

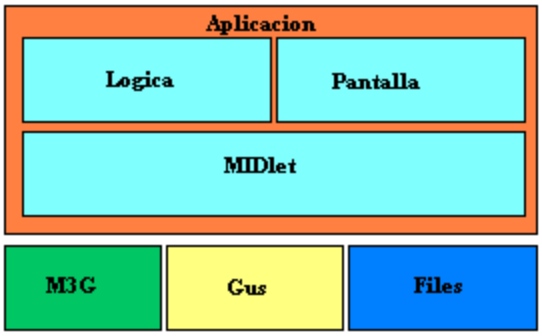


Figura 4-6: Arquitectura Laberinto en J2ME

Uso de M3G

Como mencionamos en la sección Mobile 3D Graphics (M3G) API para J2ME, existen dos modalidades de uso el modo inmediato y el modo retenido. En el desarrollo de esta aplicación se utilizó el modo inmediato, ya que las técnicas utilizadas en el diseño de la aplicación contienen algunas técnicas que se implementan en el modo retenido, las cuales consideramos que están lo suficientemente optimizadas para que la aplicación ejecute con un buen rendimiento (por ejemplo, recordamos que muchos de los casos de *culling* -recorte de objetos ocultos- fueron calculados en forma estática, y que se puede determinar cada caso en pocas instrucciones de procesador). Otro de los motivos de la elección de este modo, es para facilitar la portabilidad de ésta aplicación a la versión en C++, cambiando la lógica de la aplicación en la menor medida posible.

En la aplicación Truco del Proyecto se utilizó el modo retenido, cubriendo de ésta forma las dos modalidades soportadas por la API Mobile 3D Graphics.

Modelado de objetos

En el juego del Laberinto implementado en Java, existen tres objetos con los cuales se construye toda la escena: las paredes, pisos y los personajes. A éstos se le agrega un fondo de textura que determina el "cielo" del juego.

Paredes y pisos

Las paredes y pisos se modelan como rectángulos. Existen dos tipos de paredes, las longitudinales (que se ven a la derecha e izquierda del personaje) y las transversales (que son visualizadas de frente al personaje principal). La unidad de medida que consideramos para las paredes es en "anchos de celda"; el largo de las paredes longitudinales puede ser de 1 a 4 celdas (máxima cantidad de celdas visibles en profundidad), mientras que las paredes transversales pueden ser de 1 a 3 celdas visibles.

Para representarlas se uso el objeto *Mesh*, que representa un objeto 3D definido como una superficie poligonal formada por triángulos; en el caso de las paredes se compone de dos triángulos rectángulos que comparten la hipotenusa, generando así un objeto plano de forma rectangular (ver el rectángulo formado por dos triángulos en la **Figura 4-7**). El *Mesh* guarda información de los vértices usados y un arreglo de índices en el cual se da un orden a los vértices para indicar como se construyen los triángulos en él; con los primeros 3 índices de vértices se define el primer triangulo, los triángulos subsiguientes se definen con los dos últimos vértices del triángulo anterior, mas el siguiente vértice indicado. También contiene información de la apariencia del objeto modelado; en la caso de las paredes la apariencia se compone de una textura y la función de mapeo entre la textura y la pared. La **Figura 4-7** muestra la estructura de una *Mesh*.

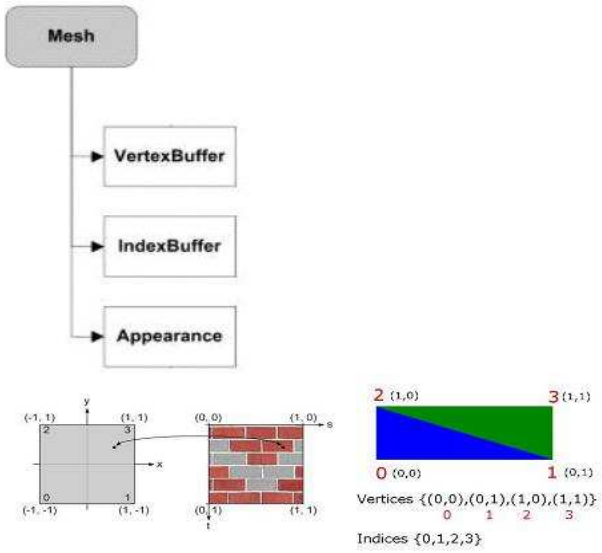


Figura 4-7: Estructura Mesh

En algunos casos las paredes se dibujan en su posición original, en otros es necesario aplicarles una traslación y luego dibujarlas para obtener la estructura visible del laberinto. El **VertexBuffer** guarda la información de los vértices de la *Mesh* y las coordenadas de la textura. El **IndexBuffer** consiste en un TriangleStripArray, el cual define un arreglo de "triangle strips", en el cual se da un orden a los vértices para indicar como se construyen los triángulos del Mesh. **Appearance** sirve para definir los atributos de como se debe realizar el *render* del objeto, por ejemplo las propiedades de la superficie.

Las figuras tienen asociadas texturas, las cuales son asignadas a la apariencia. En primer lugar es necesario cargar la imagen, esto se realiza con la función load de la clase Loader. Esto resulta en la obtención de un objeto Image2D. Luego simplemente se crea un objeto Texture2D y se asigna al objeto Appearance del Mesh.

Personaje

El personaje es la clase Gus, que fue generada a través de un modelo geométrico creado con *Blender* [6] (aplicación para el modelado de objetos 3D). La clase *Blender* generada contiene un objeto Mesh, que es al que posteriormente se realiza el *render*. Se utilizó Gus, por ser un humanoide sencillo y disponible para su descarga; pueden utilizarse otros modelos, incluso pueden diseñarse algunos (con *Blender* u otra utilidad similar), pero esa tarea no es de interés para este proyecto.

En la versión Java, los enemigos y el personaje principal están basados en el mismo modelo, solamente que el personaje principal se despliega en color rojo, mientras que los enemigos en color azul.

Archivo de niveles

Como se menciono anteriormente, se modifico la clase PTUtilities para poder leer archivos .INI con secciones y valores por sección. El archivo de niveles contiene toda la información relevante para inicializar el juego: estructura del laberinto, cantidad de enemigos en el juego, posición inicial de los actores en el juego, texturas que se utilizarán para las paredes, piso y fondo. También contiene otros parámetros que pueden modificar el desempeño de la aplicación, de ésta forma se puede modificar algunas variables del archivo de niveles, para adaptar el juego a dispositivos de diferentes características, sin modificar el código. Algunos de ésos parámetros son: el time-out que genera un nuevo movimiento automático de todos los enemigos, la cantidad máxima de enemigos que perseguirán al personaje principal (dejando en wait al resto).

4.1.4 Aplicación: Laberinto Symbian

Las aplicaciones desarrolladas en lenguaje C++ pueden ser portables entre los distintos sistemas operativos, pero existen diferencias en la creación, compilación y ejecución de los proyectos de la aplicación, que requieren un trabajo de adaptación para que la misma ejecute entre los distintos sistemas. La versión en código en C++ de la aplicación Laberinto, ha sido orientada a Symbian OS; los motivos de ésta elección son: es uno de los sistemas operativos considerados de interés del proyecto y por lo tanto uno de los más estudiados en profundidad; es el principal sistema operativo orientado a celulares que soporta este lenguaje de programación y también la librería gráfica OpenGL ES en forma oficial; por último se logró obtener una versión trial del IDE CodeWarrior, orientado exclusivamente a ese sistema operativo, lo que permite explorar además las capacidades de este IDE tan particular. Lamentablemente, no fue posible conseguir para este proyecto un dispositivo con las características necesarias para ejecutar la aplicación, por lo tanto la prueba de la misma solamente pudo llevarse a cabo en el emulador de Symbian.

Como mencionamos anteriormente, el código generado es portable a otros sistemas operativos que soporten la librería OpenGL ES (Ej.: PalmOS versión Cobalt, Windows Mobile), pero deben realizarse cambios referidos a como se debe armar y compilar el proyecto y a su prueba en el emulador y dispositivo correspondiente; si la aplicación será ejecutada en un PDA en vez de un teléfono celular o un smartphone, también deberán hacerse cambios en la interfaz de entrada, ya que las mismas difieren entre teléfonos y PDA (Ej.: existen PDA que sólo tienen dos botones, para esos casos debería incluirse comandos para el juego implementados con el uso del *Stylus*). En otros aspectos, el código de la aplicación no debería sufrir modificaciones.

A continuación mencionaremos aspectos de la aplicación, que son exclusivamente referentes a su implementación en C++.

Arquitectura

En la **Figura 4-8** se muestra la arquitectura de la aplicación. A continuación la describiremos en mayor detalle:

- **Ug**: biblioteca que funciona sobre OpenGL ES. Su funcionalidad principal son las primitivas de figuras geométricas 3D, como ser cubos, esferas y tubos entre otras. Esta implementación es parte de la librería *Vincent 3D* [13] .
- **Utils**: utilidad provista en el SDK, que entre otras funcionalidades tiene loaders de imágenes.
- **Files**: clase que permite abrir una archivo en modo lectura y leer char por char. Esta librería se puede obtener en [11] .
- **Main**: Es el programa principal. Esta compuesta de los siguientes métodos:
  - **AppInit**: se realizan las inicializaciones OpenGL ES, como ser las banderas y la perspectiva; también se cargan las texturas.
  - **AppExit**: se liberan los recursos.
  - **AppCycle**: es el método para dibujar.
- **Lógica**: contiene la lógica del juego.
- **Pantalla**: en esta clase se crean los objetos 3D, y se dispone de métodos para su dibujado.

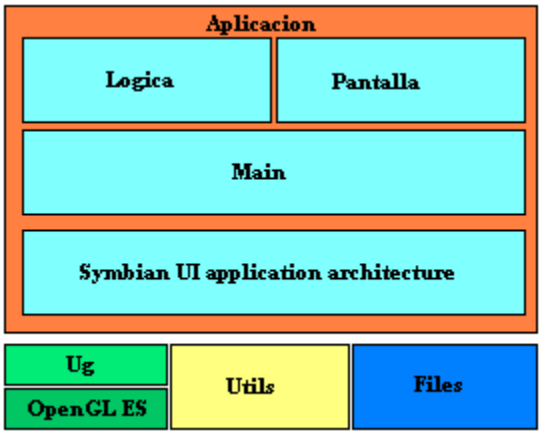


Figura 4-8: Arquitectura Laberinto en Symbian

Modelado de objetos

Al igual que en la versión Java, en el Laberinto implementado en C++ implementa los objetos: paredes, pisos y personaje. En ésta versión, no se utilizan modelos de Blender para los personajes, sino que los mismos se han creado a partir de algunas primitivas de OpenGL ES (esferas, conos, cilindros). Otra diferencia con la versión en Java, es que aquí los personajes principales y los enemigos son diferentes, como explicaremos más adelante.

Paredes y pisos

Las paredes y pisos se modelan como rectángulos. Como en la versión de Java, existen dos tipos de paredes, las longitudinales (largo 1 a 4) y las transversales (largo 1 a 3).
Para dibujarlas se hace uso de las primitivas:

- *glVertexPointer*
- *glDrawArrays*

La primitiva glVertexPointer guarda información de los vértices usados. La primitiva *glDrawArrays* sirve para indicarle que dibuje la geometría y se le debe especificar la primitiva a usar para unir los puntos, entre estas están: GL_POINTS, GL_LINE_STRIP, GL_LINE_LOOP, GL_LINES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, y GL_TRIANGLES.

Existe otra primitiva que se puede usar en lugar de *glDrawArrays*, esta es *glDrawElements.* Al igual que en el caso de Java se necesita especificarle el orden en que son tomados los vértices para formar los triángulos, de ahí la diferencia entre las dos primitivas, la cual es que la primera toma para los índices el orden usado al definir los vértices y la segunda se le debe especificar los índices usados en una estructura aparte, por ejemplo un arreglo como es el caso de Java.

Personajes

Existen dos tipos de personajes un hombre (personaje principal) y mujeres, que persiguen a nuestro héroe dentro del laberinto. Para modelarlos se uso de la librería Ug.

1) Hombre

El modelo esta compuesto por cuerpo, cabeza, pantorrillas y piernas:

- Cuerpo: se usa la primitiva ugSolidCubef, que recibe como parámetro el lado.
- Cabeza: se usa la primitiva ugSolidSpheref, que recibe como parámetros el radio, los slices y los stacks. El stack se refiere a la latitud ósea las horizontales. El slices se refiere a la longitud ósea las verticales.
- Pantorrillas y piernas: se usa la primitiva ugSolidTube, que recibe como parámetros el radio, la altura, los slices y los stacks. La diferencia entre ambos objetos esta en el radio.

2) Mujer

El modelo es el mismo que el del hombre solo que se le agregan los senos; para modelarlos se usa la primitiva ugSolidConef, que recibe como parámetros la base, la altura, los slices y los stacks.

Texturas

Las texturas son cargadas de archivos jpeg usando las librería "Image Converter Library" de la clase Utils. En primer lugar se deben habilitar antes de usarlas, esto se hace con glEnable(GL_TEXTURE_2D); posteriormente se definen las coordenadas de la textura, utilizadas para realizar el mapeo entre la textura y el polígono. El mapeo realizado fue asignar los vértices extremos de las texturas a los del polígono en la cual se va a aplicar; los vértices de las texturas son bidimensionales y abarcan el intervalo [0,1].

Para realizar el mapeo se definen un array de coordenadas, por ejemplo [0, 1, 1, 1, 0, 0, 1, 0] y luego con la primitiva glTexCoordPointer se asignan. Los parámetros son el numero de coordenadas por elemento, el tipo de cada elemento, y el array.

Posteriormente es necesario reservar memoria para esto se usa el método glGenTextures que tiene como parámetros el numero de texturas a cargar y el array donde se guardan los nombres para accederlas luego. Finalmente es necesario asignar la imagen a la textura, para esto se usan las siguientes primitivas:

- glBindTexture
- glTexImage2D

La primera se le debe pasar el nombre de la textura a asignar, la segunda asigna la imagen en memoria a la textura. Luego cada vez que se quiere asignar una textura a un poliedro se invoca a glBindTexture y luego se dibuja el poliedro.

Con la función glDisable(GL_TEXTURE_2D), se deshabilita la textura para que se puedan dibujar los siguientes objetos sin textura.

Archivo de niveles

Para cargar el archivo de niveles se usa la clase File para leer carácter por carácter, luego se determinan las *secciones*, las *claves* dentro de cada sección, y finalmente los *valores* de dicha clave. Éstos tres elementos relacionados se cargan en una lista, para su posterior búsqueda.

4.2 Aplicación Truco

Detallamos a continuación lo concerniente a la segunda aplicación desarrollada en el proyecto, describiéndola en términos generales, y profundizando los detalles de su implementación en Java.

4.2.1 Descripción de la aplicación

Se desea desarrollar el juego de cartas "truco argentino", utilizando la API M3G que corra sobre Bluetooth. El objetivo es poder jugar una partida de truco entre dos celulares por medio de Bluetooth. En cuanto al juego en si, es una versión recortada, ya que solo se pueden tirar cartas y jugar el truco, retruco y vale 4, no soportando el envido y la flor. Esto se hizo para recortar la lógica del juego, pero es perfectamente extensible. También solo pueden jugar 2 jugadores, pero la estructura soporta un posible

extensión a mas de 4 jugadores. A su vez se puede extender para soportar el truco uruguayo, lo que incluirá una muestra. El escenario es simple, se dispone de una vista de las cartas repartidas al jugador inicialmente. Luego se muestran las cartas tiradas por el otro jugador y cada vez que el jugador tira una carta, la misma es desplazada al centro de la pantalla y aparece visible en la pantalla del rival. También dispone de un menú para poder "gritar": Truco, Retruco o Vale 4, siempre y cuando lo admitan las reglas del juego.

4.2.2 Bluetooth

Para ésta sección se ha consultado [8] , [12] y también se han utilizado algunas imágenes de estos sitios. Bluetooth es un conjunto de protocolos para la comunicación inalámbrica de corto alcance. Permite interconectar dispositivos inalámbricos, empleando la banda de frecuencias 2,4 GHz (disponible en todo el mundo) asegurando la compatibilidad en todos los países. La conexión normal para celulares es de 10 metros a la redonda y no se necesita apuntar los aparatos.

En cuanto al JSR 82 cubre las siguientes funcionalidades:
- Registro de servicios.
- Descubrimiento de dispositivos y servicios.
- Establecer conexiones SPP, L2CAP y OBEX entre dispositivos.
- Usar dichas conexiones para mandar y recibir datos
- Manejar y controlar las conexiones de comunicación.
- Ofrecer seguridad a dichas actividades.

Las aplicaciones son Cliente/Servidor y las funciones de cada uno son:

Servidor:
- Crear el servicio
- Establecimiento de la conexión

El servidor realiza un open pasivo, en espera de clientes.

Cliente:
- Búsqueda de dispositivos
- Búsqueda de servicios
- Establecimiento de la conexión
- Comunicación

Un cliente Bluetooth no conoce de antemano qué dispositivos tiene a su alcance ni cuáles de ellos pueden ofrecerle el servicio que necesita. De modo que un cliente Bluetooth necesita primero buscar los dispositivos que tiene a su alcance y posteriormente les preguntará si ofrecen el servicio en el que está interesado. Luego se establece la conexión de manera activa y comienza la comunicación.

Tipo de conexiones
Existen tres protocolos:
- SPP(Serial Port Profile)
- L2CAP(Logical Link Control and Adaptation Protocol)
- OBEX (OBject Exchange)

Los primeros dos pertenecen al API javax.bluetooth, el SPP permite trabajar con las clases InputStream y OutputStream mientras que con L2CAP se usan arrays de bytes.

OBEX pertenece al API javax.obex y es un protocolo de alto nivel muy similar a http que se usa generalmente para la transferencia de archivos.

4.2.3 Arquitectura

En la **Figura 4-9** se muestra la arquitectura de la aplicación. En primer lugar se pone de manifiesto que se usan dos APIs de Java, ellas son M3G y JSR 82, la primera corresponde a gráficos 3D y la segunda a Bluetooth.

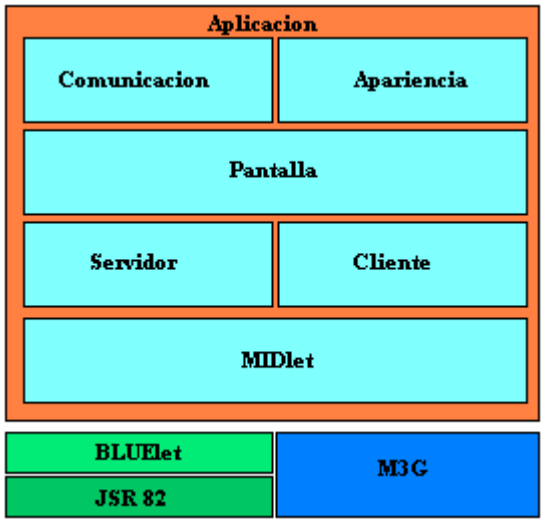


Figura 4-9: Arquitectura Truco en J2ME

El MIDlet es el corazón de la aplicación, en el mismo de acuerdo a si se actúa como servidor o cliente, se crea una instancia de Servidor o Cliente.

Por encima de la JSR 82 se utilizo una librería de uso libre de nombre BLUElet que se puede obtener en [4] , la misma es una capa abstracta que permite que el manejo de Bluetooth se haga a mas alto nivel. Servidor y Cliente usan BLUElet para gestionar la conexión y luego pasan el control a la clase pantalla.

La clase Pantalla extiende de GameCanvas por lo que es un hilo que esta en espera de eventos del usuario o entrada de datos de Bluetooth, y utiliza la clase Apariencia para saber como mostrar las cartas.

La clase Comunicación usa la conexión provista por BLUElet para que con los métodos *read* y *write* se realice el intercambio de bytes entre los dispositivos.

4.2.4 Estructuras de datos

En primer lugar esta la representación de una carta, la estructura consiste en dos atributos, uno que indica el palo y otro el valor de la carta.

Tenemos también una estructura para representar la mesa en la que ambos contrincantes juegan sus cartas, que son dos arreglos de elementos (cartas); cada jugador es propietario de uno de esos arreglos, en la que se almacenan sus propias cartas sin jugar y todas las cartas jugadas (propias y del rival).

También hay un paquete de intercambio manejado en la comunicación. Este paquete pude ser de dos tipos: el primero es el que se intercambia al inicio de cada ronda, que contiene las 6 cartas repartidas y el segundo es un paquete con un solo valor que indica la carta jugada o la opción del menú jugada, como ser

truco, retruco, etc. Para la implementación del paquete de comunicaciones existían varias posibilidades, una era al inicio de la mano repartir y enviar todas la cartas en juego, y la otra era solo enviar las que corresponden y luego ir enviando cada carta al jugarse. Desde el punto de vista de carga en la comunicación ambas son similares, por lo que se eligió la primera forma por simplicidad.

4.2.5 Uso de BLUElet

BLUElet simplifica el desarrollo de aplicaciones Bluetooth, debido a que abstrae la mayoría de los detalles de bajo nivel y posee una GUI incluida para su incorporación a la aplicación. En todas las aplicaciones Bluetooth hay etapas que siempre se repiten, como ser descubrir que dispositivos existen, mostrarlos, permitir elegir uno, descubrir que servicios existen y permitir elegir uno. BLUElet incorpora todas estas características repetitivas en el desarrollo de una aplicación en unas simples funciones integrables a la aplicación de una manera fácil.

En primer lugar el servidor y el cliente deben tener una instancia de BLUElet. El servidor crea un servicio y luego ejecuta la primitiva *acceptAndOpen*, que se bloquea hasta que un cliente se conecte. El cliente usa la primitiva *startInquiry* para descubrir el servicio y luego se conecta activamente mediante la primitiva *open*. Ambos crean streams de entrada y salida utilizando los métodos *openDataInputStream* y *openDataOutputStream* de la conexión. Luego se trabaja con la misma conexión para enviar y recibir mensajes. Esto se realiza en la clase Comunicación, que utiliza los streams creados con los métodos *read* y *write* para que se realice el intercambio de a bytes.

4.2.6 Uso de M3G

Como mencionamos existen dos modalidades de uso el *modo inmediato* y el *modo retenido*. En el desarrollo de esta aplicación se uso el *modo retenido*, para de esa forma abarcar todas las modalidades posibles de la API (recordar que para la aplicación Laberinto se utilizó el *modo inmediato*).

En primer lugar se tiene el *grafo de escena* denominado *World*. Lo primero que se hace es asignar un cámara, una luz y un fondo, para esto se crean dichos objetos y se agreguen al grafo; posteriormente el *render* se aplica a todo el grafo.

4.2.7 Modelado de objetos

Para modelar las cartas se uso el objeto Sprite3D, que representa una imagen 2D en un espacio 3D. Es rápido y el *render* lo manipula como un arreglo de píxeles de profundidad constante. Tiene la propiedad de que puede ser escalado.

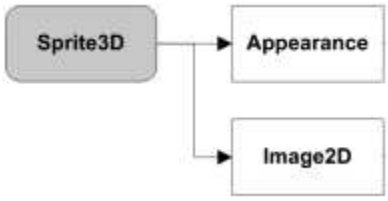


Figura 4-10: Modelado de objetos

En la **Figura 4-10** se observa la estructura del un Sprite3D. Como se ve, esta compuesto de una imagen (en este caso .png) y un objeto Appearance. Este ultimo sirve para definir los atributos de como es el *render* del objeto, por ejemplo *blending* y *fog*.

La estructura de una carta debe reflejar su numero y palo, y se deben generar 40 cartas distintas para el juego del truco, por lo que para no crear una imagen por carta, lo que se hizo fue crear dos tipos de imágenes y componerlos para formar una carta. Una primer imagen sería el palo y otra sería el número y el borde de la carta. Como cada Sprite3D se corresponde con una imagen, para formar una carta es necesario utilizar dos Sprite3D, uno para el palo y otro para el numero y el borde de la carta.

Para modelar los palos se usaron las imágenes de la **Figura 4-11**, y para los números se debieron usar 10 imágenes debido a que se debe representar los números 1, 2, 3, 4, 5, 6, 7, 10, 11 y 12. Las imágenes de los números incluyen en los ángulos superior izquierdo e inferior derecho de la carta. En la **Figura 4-12** se muestra la imagen usada para el número 1, siendo de similares características las imágenes de los otros números.

Figura 4-11: Palos de cartas

Figura 4-12: Formato de carta

4.2.8 Transformaciones

En cada instancia del juego, existen de tres a seis cartas visualizadas en la pantalla del dispositivo de cada oponente. Inicialmente cada jugador ve en la parte inferior de su pantalla solamente las tres cartas que le fueron repartidas. Posteriormente, corresponde al "mano" (contrincante que debe jugar primero en esta mano repartida) seleccionar y jugar su primera carta. El resultado visible de esta operación en la pantalla del "mano" es una traslación de la carta jugada a una posición superior en la pantalla; en la **Figura 4-13-a** se muestra un ejemplo de cómo se ve la escena en el dispositivo del "mano" luego de jugar su primer carta. La carta jugada es trasmitida a través de Bluetooth al otro dispositivo, obteniendo como resultado visible en la pantalla del "pie" (contrincante del "mano") la visualización de la carta jugada por su oponente, tal como lo muestra la **Figura 4-13-b**.

(a) Jugador "mano" **(b) Jugador "pie"**

Figuras 4-13: Primera instancia en el juego del truco

Luego de recibido el movimiento inicial, es el "pie" el que selecciona y juega la carta que crea conveniente; esto da origen a un mecanismo similar al descrito anteriormente, trasladando la carta seleccionada en la pantalla del dispositivo del "pie" tal como lo muestra la **Figura 4-14-b**, y agregando una nueva carta en el dispositivo del contrincante (en este caso el "mano"), lo cual es ejemplificado en la **Figura 4-14-a**.

Esta operativa puede repetirse hasta que los dos contrincantes hayan jugado sus tres cartas. Como se mencionó anteriormente, cada carta se compone de dos *Sprites* que conforman un único nodo del *grafo de escena*. En cada jugada, la operativa real que se realiza es la traslación de un nodo del *grafo de escena*, ya sea el correspondiente a una carta propia desplazándolo a la posición central de la pantalla, o de una carta oculta inicialmente en la escena (carta de un rival) ubicándola en una posición visible en el centro de la pantalla. En cada caso, la operación de *render* es aplicada a todo el *grafo de escena*, la visualización de las cartas en cada momento es resuelta directamente por la API *M3G*.

(a) Jugador "mano" **(b) Jugador "pie"**

Figuras 4-14: Segunda instancia en el juego del truco

4.2.9 Lógica

En esta sección se describirá la maquina de estados usada. Inicialmente un jugador procesa su estado inicial y le envía el estado resultante al opositor y se queda en espera de su próximo estado. El opositor esta en espera de que le llegue su estado. Ni bien le llega procesa ese estado y retorna el estado que le va a pasar a su opositor.

El método procesar funciona de la siguiente manera, de acuerdo a dicho estado muestra el menú correspondiente al jugador, los menús posibles son todos los caminos posibles que se pueden dar en la ejecución del truco. Luego de acuerdo al menú que eligió procesa dicho menú y según el estado de entrada determina el estado de salida que será el estado de entrada del siguiente jugador.

Los estados son los siguientes:

- REPARTIR: Reparte las cartas
- ESPERAR: Se espera por las cartas
- JUGAR: Turno del jugador mano, sin que se haya gritado "truco" todavía
- TRUCO: Se grito truco y se debe responder
- RETRUCO: Se retruco y se debe responder
- VALE4: Se hecho el Vale 4 y se debe responder
- T1JUGAR: Se esta jugando el truco. Grito el rival que es MANO.
- T2JUGAR: Se esta jugando el truco. Grito el rival que es PIE.
- R1JUGAR:Se esta jugando el retruco. Grito el rival que es MANO.
- R2JUGAR:Se esta jugando el retruco. Grito el rival que es PIE.
- V1JUGAR: Se esta jugando el vale 4. Grito el rival que es MANO.
- V2JUGAR: Se esta jugando el vale 4. Grito el rival que es PIE.

Los posibles menús son de acuerdo al estado son:

JUGAR:
- Puede irse al mazo
- Puede Gritar
- Pude jugar carta

TRUCO:
- Puede irse al mazo
- Puede Querer el truco
- Puede Retrucar

RETRUCO:
- Puede irse al mazo
- Puede querer el retruco
- Puede echar el vale 4

VALE4:
- Puede irse al mazo
- Puede querer el Vale 4

T1JUGAR:
- Si es mano
  - Puede irse al mazo
  - Puede jugar una carta
- Si es pie
  - Puede irse al mazo
  - Puede retrucar
  - Puede jugar una carta

T2JUGAR:
- Si es mano
  - Puede irse al mazo
  - Puede retrucar
  - Puede jugar una carta
- Si es pie
  - Puede irse al mazo
  - Puede jugar una carta

R1JUGAR:
- Si es mano
  - Puede irse al mazo
  - Puede jugar una carta
- Si es pie
  - Puede irse al mazo
  - Puede echar el vale 4
  - Puede jugar una carta

R2JUGAR:
- Si es mano
  - Puede irse al mazo
  - Puede echar el vale 4
  - Puede jugar una carta
- Si es pie
  - Puede irse al mazo
  - Puede jugar una carta

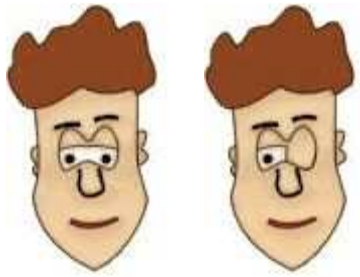
V1JUGAR:
- Puede irse al mazo
- Puede jugar una carta

V2JUGAR:
- Puede irse al mazo
- Puede jugar una carta

4.2.10 Extensión

Este truco puede ser extendido para jugar de a cuatro jugadores, en tal caso debería permitir el intercambio de señas entre las parejas de jugadores. Para esto se desarrolló una pequeña aplicación que simplemente presionando un comando, trasmite al compañero de juego la seña de todas las cartas relevantes del jugador que ejecuta el comando, para de esta forma coordinar mejor el juego entre ambos. Es habitual en el juego del truco utilizar esta técnica de comunicación.

La aplicación se llama Gestos y realiza la guiñada (una de las tantas señas posibles en el truco uruguayo). Para modelarla se usaron dos imágenes que se muestran en la **Figura 4-15**.



(a) **(b)**

Figuras 4-15: Simulación de seña del truco

Esta aplicación simplemente anima estas imágenes, mostrando una a continuación de la otra. Otras posibles extensiones serían:
- PC contra celular.
- Probar con otros protocolos de comunicación.
  - Por GPRS.
  - Por SMS.
- Extender a truco uruguayo.
- Extender reglas.
- Agregarle seguridad en las comunicaciones.

5. Conclusión y trabajos a futuro

A continuación detallamos las conclusiones de este proyecto, junto con las tareas que quedaron pendientes y otros nuevos emprendimientos que podrían realizarse como complemento del presente trabajo. Primero comenzaremos resumiendo las conclusiones de la primer parte del proyecto, que consistía en el estudio del estado del arte del tema abordado y luego pasaremos a la parte práctica, comentando los resultados de las aplicaciones desarrolladas.

5.1 Resultados alcanzados

Sistemas operativos

Symbian es un sistema operativo de amplia difusión comercial a nivel mundial, que soporta una gran cantidad de dispositivos y se actualiza permanentemente. Soporta los dos lenguajes de programación (C++ y J2ME) y el desarrollo de aplicaciones 3D basadas en OpenGL ES y M3G respectivamente; si bien su arquitectura es abierta y permite la portabilidad de sus aplicaciones a otros sistemas operativos, apuesta fuertemente al desarrollo de aplicaciones en C++, lo que dificulta adaptar el proyecto de la aplicación a otros sistemas operativos, incluso posee un entorno de desarrollo orientado exclusivamente para este sistema (CodeWarrior). Recordemos que las aplicaciones C++, además de explotar la totalidad de características del hardware de los equipos (a costa de perder portabilidad de la aplicación entre diferentes dispositivos), necesitan adaptaciones para ser compiladas en otros sistemas operativos; por lo tanto existen un gran número de aplicaciones Symbian que no son fácilmente portables a otros sistemas.

Windows Mobile está siendo utilizado en un gran número de PDAs y Smartphones; las características principales de este sistema operativo es la facilidad de integración con aplicaciones y productos Microsoft, como ser servidores de archivos Windows 2000/2003, servidores de correo Exchange, bases de datos SQL Server, estaciones de trabajo Windows XP, documentos de MS-Office y aplicaciones de uso general desarrolladas con Visual Studio; esto permite una gran interacción con las redes de trabajo Microsoft y generalmente los dispositivos Windows Mobile se utilizan como PCs portátiles de pequeño porte, ideales para quienes desempeñan tareas que requieren traslados fuera de locales físicos de la empresa (Ej.: vendedores, cobradores). Éstas características de apertura con los productos Microsoft, contrastan con el carácter cerrado del sistema respecto a la portabilidad de aplicaciones con otros sistemas operativos; existen algunos esfuerzos realizados por terceros para flexibilizar éstas dificultades de portabilidad, que han diseñado APIs para soportar el desarrollo con OpenGL ES y M3G.

El sistema operativo **Palm** está orientado a los dispositivos PDA, y junto con Windows Mobile abarcan la gran mayoría del mercado de este tipo de dispositivos. El desarrollo de aplicaciones de basa principalmente en el lenguaje C++, y para la última versión del sistema operativo (Cobalt), se soporta el desarrollo de aplicaciones 3D con OpenGL ES.

Uno de los enfoques más abiertos de los sistemas operativos estudiados es el adoptado por **SavaJe**; el desarrollo de aplicaciones en este sistema está basado en **J2SE**, es decir que pretende obtener una compatibilidad total con las aplicaciones no sólo de otros sistemas operativos que soportan **J2ME**, sino también con cualquier aplicación de uso general creada en Java ("Write Once, Run Anywhere"). Sin bien actualmente la compatibilidad con J2SE no es total, el sistema ya soportaría todas las características de J2ME, y parte de las funcionalidades correspondientes exclusivamente a J2SE. Actualmente existen pocos dispositivos que hayan adoptado este sistema operativo.

Por último, existen varios sistemas operativos orientados a **Linux**, a veces creados con propósitos específicos (Ej.: sistemas de alertas y agenda), pero debido a que son plataformas de código abierto e incorporan Java como lenguaje de programación, pueden ser en futuro una opción económica a otros dispositivos, garantizando la portabilidad de gran parte de las aplicaciones existentes.

Lenguajes de programación y librerías gráficas

Respecto a los lenguajes de programación concluimos que la mayor portabilidad de las aplicaciones puede obtenerse a través del desarrollo de aplicaciones J2ME, en particular en la perfil / configuración MIDP 2.0/CLDC 1.0, que es la más utilizada en los dispositivos móviles comerciales y además garantiza la compatibilidad con configuraciones y perfiles de versiones anteriores; sin embargo, para el desarrollo de aplicaciones 3D se debe utilizar la configuración posterior (CLDC 1.1), que incluye el manejo punto flotante y que comienza a ser difundida en los nuevos modelos de dispositivos móviles. Debido a que el perfil / configuración MIDP/CLDC asegura la compatibilidad con las versiones anteriores, podemos asegurar que las aplicaciones implementadas utilizando ésta versión de la configuración, también serán portables a los nuevos dispositivos que soporten versiones futuras de la configuración. Además las aplicaciones J2ME con perfil / configuración MIDP/CLDC, permiten independizarse del sistema operativo que ejecuta en el dispositivo móvil, ya que solamente es necesario que dicha configuración sea soportada para garantizar la portabilidad de la aplicación.

Para el caso particular de los componentes gráficos de las aplicaciones en 3D (las de mayor interés para este proyecto) desarrolladas en J2ME, hemos elegido utilizar la API adicional *Mobile 3D Graphics (M3G)*, que funciona únicamente con la configuración CLDC 1.1. Esta API dispone de dos modalidades de graficación: *retained mode* e *immediate mode*. El modo retenido (*retained mode*) utiliza una estructura en memoria para armar la escena indicándole todos los objetos disponibles (incluso los ocultos), y luego determina cuales serán los mostrados y los envía a dibujar. En el modo inmediato (*immediate mode*) los objetos se envían a dibujar de a uno, y se determina en el momento en que se van a dibujar la porción visible de los mismos. Se han utilizado ambos modos por separado, en las aplicaciones desarrolladas en el proyecto, ambas basadas en J2ME MIDP 2.0/CLDC 1.1 y la API Mobile 3D Graphics.

Resumiendo, para desarrollar aplicaciones portables a la mayor cantidad de dispositivos se recomienda el uso de **J2ME** utilizando el perfil **MIDP 2.0/CLDC 1.0**; pero si lo que se quiere realizar es una aplicación gráfica que soporte gráficos en 3D, se debe utilizar **J2ME MIDP 2.0/CLDC 1.1**, junto con la **API M3G**.

En cambio si lo que debe crearse es una aplicación de gran rendimiento y que aproveche totalmente las características del hardware de un dispositivo y el factor de portabilidad es menos importante, entonces recomendamos crear una aplicación en C++ adoptando para la creación de aplicaciones gráficas la librería OpenGL ES.

Realizando una comparación de M3G con OpenGL ES, tenemos que la primera soporta dos modalidades (retenido e inmediato), mientras que la segunda soporta una única modalidad que sería equivalente al *modo inmediato* de M3G. Con el *modo inmediato* (soportado por M3G y OpenGL ES) se tiene control en los detalles gráficos de bajo nivel. En el *modo retenido* utilizado exclusivamente por M3G, se implementa un *grafo de escena* que actúa como un link entre todos los objetos geométricos del mundo 3D y el formato de archivo M3G que permite crear modelos 3D de manera mas rápida y de mejor calidad, he incorporarlos en las aplicaciones Java de forma rápida y fácil.

Respecto a la comparación de rendimiento entre ambas librerías, tenemos que OpenGL ES es más rápida que la API M3G; una prueba de eso es que existen soluciones en desarrollo como ser el "JSR 239: Java Bindings for OpenGL ES" que permite acceder a la OpenGL ES directamente desde un programa Java. Si un dispositivo sobre el que corre una aplicación gráfica soporta OpenGL ES a bajo nivel y se crea una aplicación gráfica basada en M3G que ejecuta en este dispositivo, la API M3G actúa solo como una interfaz entre la aplicación y OpenGL ES; en cambio si el dispositivo no incorpora las características de OpenGL ES, la resolución de los gráficos recae en forma íntegra en la API M3G.

La plataforma BREW

La plataforma BREW consiste en agregar un chip a los dispositivos que la adoptan, formando una capa que independiza el desarrollo de las aplicaciones del sistema operativo del dispositivo. De ésta forma los desarrolladores BREW se aseguran que sus aplicaciones ejecutarán en cualquier dispositivo soportado, facilitando la portabilidad de las aplicaciones, aunque en un esquema más cerrado que el adoptado por J2ME. La industria de BREW está formada por: fabricantes de dispositivos, operadores de red, desarrolladores, editores de contenido. En Uruguay existen empresas que han adoptado ésta metodología de desarrollo; en el presente proyecto no se han podido realizar pruebas prácticas con ésta plataforma, debido a la carencia del equipamiento necesario para el desarrollo (tanto dispositivos móviles, como entornos de desarrollo específicos).

Entornos de desarrollo (IDEs)

Para el desarrollo de aplicaciones en Java, existen dos entornos de desarrollo que evolucionan paralelamente, y poseen prácticamente las mismas funcionalidades, ellos son: Eclipse y NetBeans. Ambos permiten seguir el ciclo completo de una aplicación de manera eficiente, integrar varios emuladores y se pueden obtener en forma gratuita. La elección entre uno y otro es en general producto de las preferencias personales del desarrollador.

Para el desarrollo en Symbian hasta fines de febrero el principal IDE fue CodeWarrior, que estaba orientado exclusivamente para este sistema operativo. Posteriormente se lanzo otro IDE también orientado exclusivamente a Symbian denominado Carbide.C++, el cual se perfila como sucesor del CodeWarrior. Carbide.C++ está basado en Eclipse y posee prácticamente todas las funcionalidades del CodeWarrior.

Si bien los IDEs de C++ permiten escribir código portable a cualquier sistema, sin embargo cada uno está orientado a un sistema en particular, ya que tienen incorporados todos los requerimientos necesarios para crear un proyecto y compilar la aplicación para un sistema operativo en particular. Por ejemplo, el Visual Studio es el IDE que permite desarrollar aplicaciones C++ para el sistema Windows Mobile. Este IDE

no pudo ser probado en el proyecto de grado, debido a que no se desarrollaron aplicaciones para ese sistema operativo, ya que nos orientamos a crear aplicaciones portables.

Aplicaciones desarrolladas

La primera de las aplicaciones llamada "Truco", incluye un componente de comunicaciones entre dos dispositivos y utiliza la modalidad de graficación llamada *retained mode* (modo retenido); ésta aplicación pudo ser ejecutada exitosamente en los emuladores disponibles, pero no ha podido realizarse la prueba en dispositivos reales, debido ha que no pudieron obtenerse dispositivos con las capacidades necesarias. La otra aplicación que llamamos "Laberinto" utiliza punto flotante y gráficos en 3D en *immediate mode* (modo inmediato); aquí también se han obtenido resultados satisfactorios en la prueba realizada en los emuladores y además la realizada en los dispositivos Sony Ericsson K700 y Sony Ericsson Z500; ésta aplicación también dispone de una versión en C++ que solamente pudo ser ejecutada en emuladores debido a la imposibilidad de acceder a dispositivos Symbian con las características necesarias.

Una vez cumplido el requerimiento de establecer una metodología de desarrollo para facilitar la creación de código reutilizable para las distintas plataformas, se han aplicado las mismas para crear las aplicaciones, de las cuales obtenemos las siguientes conclusiones:

Truco:
- Si bien la aplicación parece estar diseñada en 2D, en realidad fue realizada en gráficos en 3D (la superposición de cartas es algo que maneja la interfase gráfica M3G).
- Se asegura la portabilidad de la misma a cualquier dispositivo que soporte el perfil MIDP 2.0/CLDC 1.1 y posteriores.
- Incluye un componente de comunicaciones (objetivo opcional del proyecto).
- La prueba se realizó en forma exitosa en los emuladores, pero no pudo realizarse en dispositivos reales debido a la imposibilidad de conseguir dispositivos adecuados, es decir que soporten el perfil mencionado y simultáneamente el protocolo de comunicaciones Bluetooth.
- Se utilizó en ésta aplicación el *modo retenido* de M3G (uno de los soportados por ésta API), en el cual se utiliza un *grafo de escena*.

Laberinto:
- Versión Java basada en el perfil MIDP 2.0/CLDC 1.1 con la API M3G en *modo inmediato* (segunda modalidad soportada por M3G).
- Ésta versión pudo ser ejecutada exitosamente en los emuladores y también en los dispositivos Sony Ericsson K700 y Sony Ericsson Z500.
- Se creó una versión de la aplicación en C++ orientada para el sistema operativo Symbian, la cual solamente pudo ser ejecutada en el emulador correspondiente debido a la imposibilidad de obtener un dispositivo adecuado.
- Se asegura la portabilidad del código en ambas versiones, siempre y cuando el dispositivo soporte los estándares utilizados (J2ME: la API M3G y C++: OpenGL ES 1.0 en adelante).

Un aspecto importante además de las aplicaciones creadas, es que se logro realizar el template de una aplicación grafica Java (cualquier sistema soportado) y C++ orientada a Symbian, que incluye el código básico que debe tener una aplicación móvil y el loop principal. Este template puede servir de punto de partida para futuros emprendimientos.

5.2 Dificultades encontradas

Entre las dificultades encontradas a lo largo del proyecto destacamos la carencia de dispositivos adecuados para llevar adelante las pruebas. Actualmente el perfil / configuración dominante de los dispositivos disponibles en el mercado Uruguayo es el MIDP 2.0/CLDC 1.0; lamentablemente esta configuración no soporta la API gráfica M3G, única que permite actualmente el desarrollo de gráficos en 3D. Como existen suficientes aplicaciones estándares basadas en esa configuración y aplicaciones gráficas en 2D, hemos decidido darle a este proyecto un enfoque orientado exclusivamente a 3D, de forma de realizar un aporte innovador al desarrollo de aplicaciones gráficas para dispositivos móviles y por lo tanto nos encontramos la dificultad de que aún no se dispone fácilmente en Uruguay del hardware necesario para soportarlas.

Otra de las dificultades que se tuvo es referente a la complejidad que presentan las aplicaciones gráficas en 3D, las cuales requieren de muchos cálculos y procesamiento, y complicadas estructuras de memoria para obtener un buen rendimiento; éstas características intrínsecas en las aplicaciones gráficas disienten en general con las capacidades de hardware de los dispositivos móviles, lo que hace que el desarrollo de aplicaciones gráficas para este tipo de dispositivos sea más compleja que las desarrolladas para PC.

5.3 Objetivos iniciales y resultados obtenidos

Objetivos planteados	Trabajo realizado
Estudio sobre estado del arte en la tecnología de celulares y PDA.	Se cumplió con este objetivo; debido a la variedad y complejidad de temas tratados, este trabajo se puede seguir profundizando en las diferentes áreas.
Establecer estándares actuales y en desarrollo que faciliten la convergencia de las tecnologías.	Se establecen cuales son los estándares actuales que favorecen la convergencia de las tecnologías y se crea una metodología para el desarrollo de aplicaciones.
Delimitar entornos de desarrollo que faciliten la programación y el testeo de aplicaciones.	Se estudian los diferentes entornos de desarrollo orientados a dispositivos móviles y se prueban los que pudieron obtenerse en forma gratuita.
El diseño y desarrollo de al menos dos aplicaciones con componente gráfico, utilizando las herramientas estudiadas en la parte de investigación del proyecto.	Se diseñaron dos aplicaciones con componentes gráficos utilizando las herramientas estudiadas en la primer parte del proyecto. Las aplicaciones realizadas consisten en juegos y aunque no están lo suficientemente elaboradas como para tener un interés comercial, cuentan con las características necesarias para comprobar en la práctica los temas estudiados en la primer parte del proyecto. Lamentablemente no en todos los casos fue posible probar las aplicaciones en dispositivos reales, debido a la imposibilidad de acceder a los mismos. En todos los casos las pruebas pudieron ser realizadas en los emuladores correspondientes.
Lograr que las aplicaciones ejecuten en la mayor cantidad de dispositivos posibles.	Las aplicaciones Java garantizan la portabilidad para aquellos dispositivos que soporten la perfil / configuración MIDP 2.0/CLDC 1.1 y posteriores. Para el caso de la aplicación desarrollada en C++ se garantiza la portabilidad a los dispositivos Symbian que soportan OpenGL ES, y a cualquier dispositivo en general que soporte OpenGL ES 1.0 o posterior, con algunos cambios en la creación del proyecto a compilar; para los PDA,

Objetivos planteados	Trabajo realizado
	deben modificarse la entrada de datos, debido a la diferencia entre los dispositivos de entrada con los teléfonos celulares.
Opcionalmente, incluir en alguna de las aplicaciones una funcionalidad de comunicaciones entre dispositivos.	Se incluyó en una de las aplicaciones desarrolladas (Truco) un componente de comunicaciones a través del protocolo Bluetooth.

5.4 Aportes realizados

El principal aporte que realiza este trabajo, es la incursión en aplicaciones gráficas en 3D para dispositivos móviles, ya que hoy en día existe suficiente conocimiento y desarrollo comercial de aplicaciones de propósito general y aún aplicaciones gráficas en 2D orientadas a este tipo de dispositivos.

Otro de los aspectos importantes de este proyecto, es el haber desarrollado una metodología para el desarrollo de aplicaciones portables entre la mayor cantidad de dispositivos, lo que incluye la creación de templates de aplicaciones Java y C++ orientadas al sistema operativo Symbian.

También se utilizaron algunas de las técnicas habituales de aplicaciones gráficas en general, adaptándolas al caso particular de dispositivos móviles de escaso porte.

5.5 Tareas pendientes y extensiones posibles

El estudio de la tecnología móvil es ya de por sí un tema muy vasto y complejo además de encontrarse en permanente actualización, por lo tanto los temas tratados en este proyecto pueden considerarse como una introducción al tema y pueden profundizarse en las diferentes áreas estudiadas.

Dentro de la parte práctica del trabajo quedaron algunos aspectos pendientes, y otros que pueden resultar interesantes para complementar el trabajo teórico realizado. A continuación enumeramos esos aspectos:

- Realizar la prueba en un dispositivo real del Laberinto implementado en C++ para Symbian OS.
- Implementar la aplicación truco en C++.
- Portar las aplicaciones C++ desarrolladas para Symbian OS a otros sistemas operativos como Windows Mobile o Palm OS.
- Verificar en la práctica con dispositivos reales el desempeño más eficiente del código implementado en C++ con el código equivalente en Java.
- Implementar una aplicación Java en dos versiones diferentes, una utilizando el *modo retenido* de M3G, y otra utilizando el *modo inmediato*, y comprobar la diferencia de desempeño entre ambas versiones.
- Incursionar en otras formas de comunicación (además de Bluetooth) soportadas por los dispositivos móviles, como ser GPRS, SMS, etc.

Por último hay aspectos en las aplicaciones que podrían mejorarse, pero que no revisten mayor interés para la temática del proyecto, ya que los aspectos fundamentales que se querían probar fueron incluidos y detallados en la sección correspondiente. Los aspectos inconclusos de las aplicaciones del proyecto son los siguientes:

- El juego del Truco, es una versión recortada (sin flor ni envido) del truco argentino (sin muestras); carece de interés para el proyecto completar la lógica del juego.
- Se pensaba agregar a este juego otro componente gráfico y de comunicaciones extra, ampliar el uso a cuatro personas y poder transmitir las señas de las cartas al compañero a través de imágenes animadas; se explicó de que forma se implementaría esta funcionalidad, pero no pudo llevarse a cabo debido a los plazos fijados en el proyecto (se debería además cambiar la lógica del juego para extenderlo a cuatro jugadores).
- Dentro del laberinto se pensaba dotar a los actores del juego de la capacidad de saltar y agacharse, con lo cual el sistema de detección de colisiones debería funcionar sin cambios, si la inclusión de éstas funcionalidades se atienen a lo detallado en el anexo: *Aplicación Laberinto*.
- Se ha intuido (aunque no comprobado totalmente), un bug al mostrar algunos enemigos mientras el personaje está en determinada posición. Esto se pudo apreciar probando con laberintos muy complejos creados en la etapa final del proyecto y debido a los plazos fijados, no ha podido estudiarse a fondo.

Glosario

ActiveSync: software para los equipos basados en Windows Mobile OS para sincronizar los mismos con los PC de escritorio, este gestiona la conexión entre el equipo de escritorio y el dispositivo.

ARM: en los comienzos se denominaba ARM a una familia de microprocesadores RISC desarrollados por la empresa Acorn RISC Machine, el trabajo creció tanto que se creo una nueva compañía de nombre Advanced RISC Machines Limited. Actualmente la empresa se denomina ARM Limited y es la marca de los procesadores mas usados para dispositivos móviles. Además de fabricar microprocesadores, desarrolla procesadores 3D, memorias, periféricos y herramientas para desarrollo de SW entre otras tecnologías.

Blender: aplicación que permite crear modelos geométricos en 3D; éstos modelos pueden ser importados por otras aplicaciones para crear figuras y/o animaciones realistas.

Bluetooth: es una tecnología que permite interconectar dispositivos inalámbricos, empleando la banda de frecuencias 2,4 GHz (disponible en todo el mundo) asegurando la compatibilidad en todos los países. Se utiliza cuando los dispositivos se encuentran a corta distancia.

CDMA: Code Division Multiple Access sistema de comunicaciones móviles implementado de acuerdo al estándar US (IS 95) en el intervalo de frecuencias entre los 800 y los 1900 Mhz.

CLDC: *Connected Limited Device Configuration*; configuración asociada a la plataforma J2ME, que permite crear un ambiente de desarrollo orientado a una amplia variedad de dispositivos móviles.

Culling: término empleado en aplicaciones gráficas para identificar a las técnicas de recorte de objetos y caras no visibles en la escena, para disminuir el trabajo de *render* de la biblioteca o hardware que resuelve los gráficos.

GPRS: General Packet Radio Service, tecnología que permite la transmisión de datos en forma inalámbrica de hasta 115 Kbit/s, proporcionando servicios de acceso a Internet y Correo Electrónico.

GSM: *Global System for Mobile Communications*, estándar internacional de comunicaciones digitales celulares. La familia de sistemas GSM incluye el GSM 1800, GSM 1900 y GSM 900. Los teléfonos GSM pueden clasificarse en el nivel 1 o nivel 2 de acuerdo con el nivel de conformidad con la norma correspondiente (el segundo nivel es el que soporta actualmente mayor cantidad de servicios, aunque no todas las redes GSM lo soportan). Abarca un conjunto de servicios y procedimientos de seguridad y autenticación.

J2ME: plataforma *Java 2 Micro Edition*; proporciona un entorno de desarrollo de software y despliegue para dispositivos inalámbricos y móviles, creada por Sun Microsystems para independizar el software del sistema operativo que corre en el móvil en que ejecuta.

JVM: *Java Virtual Machine* (máquina Virtual de Java), software capaz de interpretar las sentencias de una aplicación Java.

KVM: *K Virtual Machine*, maquina virtual Java creada para ejecutar en dispositivos con limitada capacidad de procesamiento y memoria (K).

MIDP: *Mobile Information Device Profile*; perfil relacionado con la configuración CLDC de J2ME, que permite obtener un conjunto de APIs estándar para el desarrollo de aplicaciones en determinados dispositivos móviles.

MMS: *Multimedia Message Service* (Servicio de Mensajes Multimedia): servicio disponible en las comunicaciones móviles de 3G que permite reunir contenidos de texto formateado, video y audio en un único mensaje y obtenidos en tiempo real.

M3G (o JSR-184): API que permite escribir programas con gráficos 3D en dispositivos móviles basados en J2ME MIDP 2.0/CLDC 1.1; es una adaptación orientada a dispositivos móviles de la librería gráfica Java 3D (diseñada para PC's).

OpenGL ES: librería de funciones gráficas utilizada en programas C++, orientada a dispositivos móviles; está basada en la librería OpenGL para PCs.

PACE: *Palm Application Compatibility Environment*, entorno creado para compatibilidad de aplicaciones entre varias versiones de Palm OS (sistema operativo orientado a dispositivos PDA).

Render: es el cálculo complejo desarrollado por un dispositivo destinado a generar secuencias de imágenes.

SIMD: Single Instruction Multiple Data, o Instrucción Única para Múltiples Datos. Son un tipo de instrucciones de procesador que aplican una misma operación sobre un conjunto más o menos grande de datos, por ejemplo pueden realizar una operación, como una suma sobre varios números a la vez.

SMS*: Short Message Service*; servicio para sistemas digitales que permite enviar y recibir mensajes de hasta 160 caracteres a través de un Centro de Mensajes del operador telefónico, que asegura la llegada del mismo aún si el teléfono destino está fuera de servicio o fuera de alcance en el momento del envío.

Stylus: Puntero con forma de lapicera que se utiliza para apuntar y escribir en un PDA.

Referencias

Se indican a continuación los sitios y publicaciones tomados como referencias para realizar este trabajo, y de los cuales también se han extraído algunas de las imágenes presentadas en el mismo.

[1] Akenine-Möller, T. - (2005). Mobile Computer Graphics. [Online]. Disponible en: http://www.cs.lth.se/EDA075/lectures/L1.pdf
Última visita: 24/02/2006.

[2] ARM - Sitio Oficial. (2006). ARM powered products. [Online]. Disponible en: http://www.arm.com/markets/mobile_solutions/app.html
Última visita: 18/05/2006.

[3] BBC News. (Octubre 2005). Future mobiles to get chip boost. [Online]. Disponible en: http://news.bbc.co.uk/1/hi/technology/4305342.stm
Última visita: 18/05/2006.

[4] Benhui - Sitio oficial. Bluetooth Application Utility GUI Component. [Online]. Disponible en: www.benhui.net
Última visita: 20/01/2006.

[5] Berka, S. - INI file for Midlets. [Online]. Disponible en: http://forum.java.sun.com/thread.jspa?threadID=217812
Última visita: 02/03/2006.

[6] Blender - Sitio oficial. (2006). [Online]. http://www.blender3d.org/cms/Home.2.0.html
Última visita: 02/03/2006.

[7] Boletín Tele-Semana (2004). [Online]. Disponible en: http://www.tele-semana.com/archivo/Download.php?c=0946421004022-466
Última visita: 12/05/2006.

[8] Borches, P. - (2004). Java 2 Micro Edition: Soporte Bluetooth. [Online]. Disponible en: http://www.it.uc3m.es/celeste/docencia/j2me/tutoriales/bluetooth/EstudioTecnologico1_0.pdf
Última visita: 25/01/2006.

[9] Calamatta, M. - (2005). A Guide to Windows Mobile Programming for Symbian OS Developers. [Online]. Disponible en: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnppcgen/html/wmprimer_symbian.asp
Última visita: 15/03/2006.

[10] Cellular OnLine. [Online]. Disponible en: http://www.cellular.co.za/cellphone_inventor.htm
Última visita: 17/05/2006.

[11] De Santos García, G. - File. [Online]. Disponible en: http://www.todosymbian.com/files2/file.zip
Última visita: 15/03/2006.

[12] Gimeno Brieba, A. - (2004). JSR-82: Bluetooth desde Java. [Online]. Disponible en: http://www.javahispano.org/tutorials.item.action?id=49
Última visita: 24/02/2006.

[13] Hans-Martin, Will. - Vincent 3D Rendering Library. [Online]. Disponible en: http://sourceforge.net/projects/ogl-es/
Última visita: 15/03/2006.

[14] Intel - Sitio Oficial. (2006). Publicly Available Devices. [Online]. Disponible en: http://www.intel.com/design/pca/hof/index.htm
Última visita: 18/05/2006.

[15] J2ME-grasia, Sistemas Informáticos y Programación de la Univ. Complutense de Madrid. (2006). Herramientas para el desarrollo de j2me. [Online]. Disponible en: http://grasia.fdi.ucm.es/j2me/_AppsTools/index.html#IDEs
Última visita: 24/03/2006.

[16] Kishonti Informatics LP. JBenchmark. [Online]. Disponible en: http://www.jbenchmark.com/index.jsp
Última visita: 24/03/2006.

[17] Motorola – Sitio oficial. (2004). J2ME™ Developer Guide, Motorola E680 Handset. [Online]. Disponible en: http://www.motocoder.com/
Última visita: 25/02/2006.

[18] Pipkorn, P.; Bengtsson, T. - (2005). Mobile 3D Graphics APIs. [Online]. Disponible en: http://www.cs.lth.se/EDA075/lectures/L2.pdf
Última visita: 24/02/2006.

[19] Pulli, K.; Aarnio, T.; Roimela, K.; Vaarala, J. - (2005). Designing Graphics Programming Interfaces for Mobile Devices. IEEE Computer Society, 0272-1716/05, pp. 66-74.

[20] Retamar, A. - (2004). Mi primera hora con Eclipse. [Online]. Disponible en: http://www.javahispano.org/tutorials.type.action?type=j2se
Última visita: 24/03/2006.

[21] Rizan - (2005). Symbian OS design faults. [Online]. Disponible en: http://www.codeproject.com/ce/Symbian_OS_design_faults.asp
Última visita: 25/02/2006.

[22] SIGGRAPH. (2005). Developing Mobile 3D Applications With OpenGL ES and M3G: Using OpenGL ES. [Online]. Disponible en: http://people.csail.mit.edu/kapu/siggraph_course/SGcourseJVnotes.pdf
Última visita: 24/02/2006.

[23] SIGGRAPH. (2005). Developing Mobile 3D Applications With OpenGL ES and M3G: Intro and OpenGL ES Overview. [Online]. Disponible en: http://people.csail.mit.edu/kapu/siggraph_course/SGcourseKPnotes.pdf
Última visita: 24/02/2006.

[24] SIGGRAPH. (2005). Developing Mobile 3D Applications With OpenGL ES and M3G: M3G Overview. [Online]. Disponible en:

http://people.csail.mit.edu/kapu/siggraph_course/SGcourseTAnotes.pdf
Última visita: 24/02/2006.

[25] Symbian Ltd. (Marzo 2005). Symbian licenses Microsoft Exchange Server ActiveSync Protocol for Symbian OS. [Online]. Disponible en: http://www.symbian.com/news/pr/2005/pr20051772.html
Última visita: 18/05/2006.

[26] Symbian Ltd. (Diciembre 2005). Symbian announces 22nd phone for Japanese market. [Online]. Disponible en: http://www.symbian.com/news/pr/2005/pr20053314.html
Última visita: 18/05/2006.

[27] Symbian Ltd. (Mayo 2006). Symbian welcomes the smallest Nokia E series device, the Nokia E50 for mobile professionals. [Online]. Disponible en: http://www.symbian.com/news/pr/2006/pr20067931.html
Última visita: 18/05/2006.

[28] TECTIMES. (2005). Desarrollos moviles con .NET. [Online]. Disponible en: http://www.tectimes.com/lbr/Graphs/revistas/lpcu077/capitulogratis.pdf
Última visita: 25/02/2006.

[29] Völkl, G. - Gus.blend. [Online]. Disponible en: http://www.nelson-games.de/bl2m3g/default.html
Última visita: 02/03/2006.

[30] Weinstein, A. - Degel Software Ltd. (2002). Symbian OS C++ for Windows C++ programmers. [Online]. Disponible en: http://www.symbian.com/developer
Última visita: 20/02/2006.

[31] Wigley, A. - CM Group. (2005). Migrating Symbian OS Applications to Windows Mobile-based Smartphones. [Online]. Disponible en: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnppcgen/html/symbianmigration.asp
Última visita: 20/03/2006.

Anexos

En el presente informe también se han realizado referencias y tomado imágenes pertenecientes a documentos anexos del proyecto. Lo que sigue es una lista de dichos anexos.

Ítem	Nombre	Descripción	Medio
1	Informe Final	Este documento.	Impreso y CD
2	Tecnología Celular	Estado del arte.	CD
3	Tecnología PDA	Estado del arte.	CD
4	Sistema Operativo Symbian	Descripción del Sistema Operativo.	CD
5	Sistema Operativo Windows Mobile	Descripción del Sistema Operativo.	CD
6	Sistema Operativo Palm	Descripción del Sistema Operativo.	CD
7	Sistema Operativo SavaJe	Descripción del Sistema Operativo.	CD
8	J2ME basado en MIDP/CLDC	Descripción del lenguaje.	CD
9	La plataforma BREW	Descripción de la plataforma.	CD
10	OpenGL ES para C++	Descripción de la biblioteca gráfica.	CD
11	API Mobile 3D Graphics para J2ME	Descripción de la API gráfica.	CD
12	IDEs	Descripción de los IDEs.	CD
13	Anexo Aplicación Laberinto	Descripción de la aplicación.	CD
14	Manual de usuario	Manuales para uso de las aplicaciones.	CD
15	Glosario	Glosario de términos en el proyecto.	CD
16	Fuentes y ejecutables:		
	• LaberintoS.rar	Laberinto en Symbian	CD
	• LaberintoJ.rar	Laberinto en Java	CD
	• BlueTruco.rar	Truco en Java	CD
	• Gestos.rar	Aplicación de gestos en Java	CD