Magíster en Ingeniería de Software

Universidad Nacional de La Plata

Facultad de Informática

Tesis

Identificación y Clasificación de Patrones en el Diseño de Aplicaciones Móviles.

Dirigida por Dr. Gustavo Rossi

Desarrollada por Ing. Darío Yorio

Índice General

A Valentina, por estar siempre a mi lado

CAPITULO 1

INTRODUCCIÓN Y OBJETIVO

1.1 INTRODUCCIÓN

Durante los últimos años hemos sido testigos de un gran crecimiento tecnológico en materia de comunicaciones inalámbricas. Han surgido innumerables mejoras en las redes de comunicación, protocolos más eficientes, mayores anchos de banda, mayores áreas de cobertura, entre otras. Y por otra parte se han mejorado las prestaciones de los dispositivos móviles, los cuales cuentan ahora con baterías de mayor duración, displays de mayor resolución, mayor poder de cómputos y un número creciente de nuevas prestaciones. Esta revolución tecnológica ha facilitado el desarrollo de aplicaciones móviles más complejas que sus predecesoras, pero que a su vez deben poder evolucionar con mayor rapidez. En otras palabras las nuevas aplicaciones móviles deben adaptarse a los cambios tecnológicos, siendo flexibles y extensibles de forma que puedan cambiar su funcionalidad o extenderse para soportar una nueva.

Actualmente el desarrollo de sistemas para escenarios móviles se ha difundido considerablemente. De aplicaciones casi experimentales, solo utilizadas en ambientes universitarios, se ha llegado a aplicaciones comerciales de uso a mayor escala, con una variedad que va desde las orientadas a empresas, pasando por las de uso personal y sin dejar de lado aquellas utilizadas simplemente para entretenimiento.

El mercado ha cambiando, o visto de otra forma el usuario ha cambiado, sus necesidades son otras, su nivel de requerimientos es otro. Por citar un ejemplo, no hace mas de cinco años enviar mensajes SMS era cosa de unos pocos, hoy existen prestadoras de servicios celulares que permiten enviar mensajes a solo aquellos “contactos” que se encuentran dentro de las proximidades del usuario, y esto ya es visto como algo habitual, el usuario se acostumbra a este servicio y cada vez resulta mas difícil sorprenderlo con nuevas funcionalidades. Los diseñadores deberán agudizar su ingenio para lograr nuevas aplicaciones, situación que sin lugar a duda sucederá. En este escenario aparecen nuevas necesidades, cada vez más desafiantes y complejas que las anteriores, y que obligan a soluciones que deben ser puestas en servicio cada vez con mayor rapidez.

Si bien no se puede negar que el dominio de las aplicaciones móviles, es un dominio completamente ligado a los avances tecnológicos, y hasta se podría afirmar que el mismo es de carácter sumamente restrictivo, al momento de diseñar una aplicación de este tipo existen otros puntos a tener en cuenta, más allá que las limitaciones tecnológicas. Por ejemplo diferentes grados de personalización de la aplicación por medio del usuario, según sus preferencias o su contexto actual, son requerimientos que pueden afectar el diseño de una aplicación móvil y que distan de tener una componente tecnológica que los afecte.

En esta evolución surgen desafíos más que interesantes para la ingeniería de software. La importancia de analizar el diseño, antes comenzar a escribir una sola línea de código, para aplicaciones que debe evolucionar rápidamente, es sin lugar a dudas un punto que no puede ser menospreciado. El modelado de una arquitectura a nivel conceptual permite al diseñador decidir cuestiones que tendrán influencia a lo largo de todo el ciclo de vida de la aplicación.

En la literatura, se encuentran varios trabajos relacionados con el desarrollo de aplicaciones para escenarios móviles [Bur03], [CD03], [Chu04], [KCL03], [LB03], [Rot01], [RR04], y dentro de los mismos existen varias corrientes que propician la utilización de patrones. Argumentan el éxito obtenido en otros dominios para replicar la idea al dominio de aplicaciones móviles. En algunos de ellos se presentan jerarquías de patrones, en otros se los agrupa según clases y los mas avezados, presentan al estilo de la tabla periódica, una matriz donde se predice cual seria el patrón ha ser descubierto para cada fila/columna.

De todas maneras, es claro que el uso de patrones como herramienta para capturar conocimiento o experiencia de los diseñadores mas experimentados ha sido uno de los grandes logros de la ingeniería de software y por lo tanto es natural tratar de replicar este concepto en el dominio de las aplicaciones móviles.

1.2 OBJETIVO

El propósito de esta tesis es conformar un catalogo de patrones arquitectónicos para el dominio de las aplicaciones móviles. Para esto se tendrán en cuenta algunos de los trabajos realizados en el área, y se identificaran y clasificaran patrones desde aplicaciones actualmente en uso.

Las principales contribuciones que surgen de la tarea propia de alcanzar el objetivo, se detallan a continuación:

- Caracterización de las aplicaciones móviles por medio de la compresión de los requerimientos propios de las mismas.
- Clasificación de las aplicaciones móviles en función de su utilización con el objeto de facilitar su análisis.
- Análisis detallado de restricciones propias del dominio móvil que afectan el diseño de aplicaciones.
- Generación de una clasificación de patrones para el dominio de aplicaciones móviles.
- Demostración de cómo el uso de los patrones encontrados facilita la comprensión de aplicaciones existentes y el diseño de nuevas aplicaciones.

1.3 ORGANIZACIÓN DE LA TESIS

Este trabajo se encuentra estructurado en siete capítulos y tres anexos, acompañados de un glosario con los términos que pueden llegar a facilitar la compresión general. A continuación se detalle brevemente el contenido de cada uno de los módulos.

Capitulo 1. Breve introducción global y enunciado del objetivo

Capitulo 2. Trata las aplicaciones móviles desde un elevado nivel de abstracción. Comienza enunciado los requerimientos que una aplicación de este dominio debería cubrir para luego presentar una clasificación según las áreas de interés más habituales, esto es: el uso de portales, de bases de datos y de archivos compartidos. Se analizan en detalle cada una de estas áreas presentando una opción de arquitectura para cada caso.

Capitulo 3. Detalla una serie de consideración que de una forma u otra afectaran las decisiones de diseño de una aplicación móvil. A diferencia del capitulo anterior el nivel de abstracción es mucho menor, llegando a darse ejemplos de implementación de algunas soluciones.

Capitulo 4. En este capitulo se hace una breve introducción al concepto de patrones para luego pasar a analizar tres trabajos relacionados con patrones en el dominio de las aplicaciones móviles. Para cada uno de ellos se realiza una síntesis donde se detallan los puntos más sobresalientes.

Capitulo 5. Constituye el núcleo central de esta tesis. Presenta un catalogo de patrones arquitectónicos para el dominio de las aplicaciones móviles. Se establece una clasificación y se ubican los patrones extraídos, dentro de la misma. Cada patrón es ajustado a un formato similar al utilizado en libro [GHJV94] y acompañado de un diagrama UML simplificado.

Capitulo 6. A partir de los patrones extraídos se analiza como los mismos pueden ser utilizados para describir un aplicación actualmente en uso y como pueden colaborar en el diseño de una nueva.

Capitulo 7. Se presentan las conclusiones generales de esta tesis y se sugieren los posibles ejes para futuros trabajos.

Anexo 1. En este modulo se presentan los fundamentos de las aplicaciones *context-aware*; como contexto, categorías de contexto y funciones *context-aware*. Se analizan dos trabajos representativos del área. Y por ultimo se dan algunos detalles sobre tecnología entorno a este tipo de aplicaciones y mecanismos para la detección del contexto.

Anexo 2. Se presenta a la Arquitectura de Software como disciplina, comenzado con una breve recopilación histórica de la misma, para luego describir sus conceptos fundamentales.

Anexo 3. Introducción al concepto de patrones, partiendo desde sus principios en la Arquitectura tradicional ligado a los trabajos de Christopher Alexander, para luego llegar a las diferentes representaciones de los mismos en el ámbito del desarrollo de Software. Se analizan las distintas definiciones, sus propiedades y presentaciones, y se presenta una breve descripción de lo que ha dado por llamarse “lenguaje de patrones”. Se profundiza sobre los Patrones de Diseño y los Patrones Arquitectónicos. Por ultimo se analiza brevemente el concepto de antipatrones.

CAPITULO 2

APLICACIONES MÓVILES

Los dispositivos de computación inalámbrica ha proliferado rápidamente, requiriendo aplicaciones de software que puedan manejar esta nueva realidad. Los usuarios esperan la misma funcionalidad de aplicaciones que corren en sus dispositivos móviles estando conectados o desconectados de la red. Esperan aplicaciones que puedan soportar conexiones intermitentes, anchos de banda cambiantes y que manejen eficientemente el problema del *roaming*. Y esperan que el uso de la energía sea administrado de forma de maximizar el tiempo útil de la batería sin degradar la performance. Estas expectativas presentan un nuevo conjunto de desafíos para los desarrolladores de aplicaciones.

2.1 INTRODUCCIÓN

El rango de dispositivos móviles va desde dispositivos dedicados a tareas específicas, como los teléfonos celulares, hasta aquellos dispositivos de propósito general, como notebooks. Cada uno de ellos presenta diferentes conjuntos de desafíos para el diseño de aplicaciones móviles. Algunos de estos desafíos compartidos por la mayoría de los dispositivos móviles incluyen:

- La ubicación física del dispositivo y la configuración pueden cambiar impredeciblemente a medida que el dispositivo esta conectado o desconectado de la red o se mueve entre dos puntos de conexión. La arquitectura de aplicación móvil debe soportar una operación consistente operando tanto *online* como *offline* y proveer una conectividad continua mientas el dispositivo se mueve entre puntos de conexión.
- Los dispositivos energizados por baterías pueden operar por un tiempo limitado sin recargar o reemplazar las mismas. La arquitectura de una aplicación móvil debe se diseñada para administrar la energía limitada de las baterías, mediante el uso de estrategias que prologuen la vida útil al reducir el consumo sin sacrificar la performance del sistema.

Los dispositivos pequeños dedicados para un fin especifico pueden tener limitaciones adicionales como un pantalla de tamaño reducido o un limitado almacenamiento o poder de cómputos. Una arquitectura de aplicaciones móviles debe proveer soporte para un amplio rango de dispositivos.

Las redes inalámbricas que utilizan tecnología Wi-Fi operan a velocidades de una red de área local, estas permiten a los usuarios estar conectados sin tener ninguna conexión física, eliminar los cargos por el acceso a través de redes públicas y dejar de lado las limitaciones de la conexión a través de un modem y una línea telefónica. Sin embargo las redes inalámbricas son susceptibles a externas interferencias y atenuación lo que potencialmente afecta su confiabilidad, disminuye el ancho de banda efectivo, y presenta desafíos para una seguridad eficiente.

2.2 REQUERIMIENTOS PARA UNA ARQUITECTURA DE APLICACIONES MÓVILES

Una aplicación diseñada para ser usada en un dispositivo móvil debe cumplir con ciertos requerimientos, algunos de los cuales son propios del ambiente móvil y otros que pueden ser requerimientos de cualquier tipo de aplicación. A continuación se presentan

aquellos más relevantes [Bur03], solo con el objeto de que a través de los mismos se tenga una perspectiva de las características de este tipo de aplicaciones.

- *Operación consistente tanto online como offline.* En varias arquitecturas, los datos son almacenados en un sistema compartido accesible a través de la red, en forma de documentos, registros de datos o archivos binarios, donde se tiene un acceso coordinado a una copia de la información. Una aplicación móvil debe ser diseñada de forma de que los usuarios puedan acceder a los datos sin importar si lo hace en forma *online* o en forma *offline*. Cuando se trabaja *offline*, el usuario percibe que la información compartida esta disponible para lectura y escritura. Cuando la conectividad regresa, los cambios en la información local son integrados a la copia de red y viceversa.
- *Conectividad Continua.* Una aplicación diseñada para movilidad debe trabajar con un agente o servicio Proxy para permitir un manejo transparente de los cambios en la conectividad. La conectividad no tiene que ser un requerimiento para la funcionalidad y cortes intermitentes e inesperados en la conexión con la red deben poder ser manejados satisfactoriamente. Así mismo este agente o servicio Proxy debe poder seleccionar la red óptima de las disponibles en ese momento, y manejar las tareas propias de la comunicación como autentificación segura o autorización y direccionamiento lógico.
- *Clientes que soporten multi-plataformas.* Una aplicación diseñada para movilidad debe al menos ajustar su interacción y comportamiento al dispositivo en el que corre, como por ejemplo tipo de entrada y salida, recursos disponibles y nivel de performance.
- *Energía y performance optimizadas.* Una aplicación diseñada para movilidad debe manejar de cerca el uso de la energía de un dispositivo portátil que comúnmente funciona a baterías. Por ejemplo una constante sincronización de los datos en memoria con los del disco rígido puede consumir las baterías rápidamente, al igual que con la actividad de radio al buscar constantemente el siguiente punto de comunicación.
- *Administración de Recursos*. Un recurso como la energía, el ancho de banda o el espacio de almacenamiento puede ser consumido y existe en una cantidad finita. La administración de recursos debe permitir el monitoreo de atributos como cantidad o tasa de uso, y soportar notificaciones basadas en disparadores predefinidos por el usuario.
- *Administración del Contexto*. Contexto es cualquier información que puede se usada para caracterizar la situación de una entidad. Donde una entidad es una persona, lugar u objeto que es relevante para la interacción entre un usuario y una aplicación, incluyendo al usuario y la aplicación [Dey00]. La administración del contexto debe permitir el monitoreo de atributos como ubicación actual o tipo de dispositivo, y proveer notificación de cambios en el mismo.
- *Codificación.* La codificación involucra la modificación de los datos y protocolo en función de los requerimientos del actual contexto y recursos disponibles. Ejemplos de codificación son la encriptación, compresión y transcodificación. Una implementación de la capacidad de codificación permitirá la enumeración de los *encoders* y *decoders* disponibles. Luego con esta información disponible junto con la capacidad de administración del contexto, proveer la habilidad de negociar el uso de uno u otro método de codificación.
- *Vista consistente.* Debe proveer vista una consistente y reconciliable de datos y el estado en que es compartida en medio de un sistema intermitentemente conectado, utilizando emulación, "*pre-fetch*" y reconciliación. La emulación permite a los usuario despreocuparse si están trabajando con una copia local o remota de la información. El "*pre-fetch*" maneja el *caching* de los datos. Y la reconciliación se asegura que aquellas

modificaciones que hayan sido realizadas en estado *offline*, sean aplicadas a todas las copias de la información.

- *Almacenamiento Duradero.* La capacidad de manejar un almacenamiento duradero permite la persistencia de datos de configuración o información estática.
- *Mensajería Confiable.* La capacidad de un mensajería confiable provee la habilidad para definir y controlar la semántica del mensaje a entregar así como el tipo de entrega (sincrónica, asincrónica, etc.).
- *Políticas.* El uso de políticas permite tener un lugar común para colectar, correlacionar y reaccionar o adaptar la información y eventos provistos por las otras habilidades. Estas reciben los eventos y los confrontan a un conjunto de reglas previamente almacenadas. Si el resultado de esta confrontación es verdadero se ejecuta la acción previamente asociada al evento. A estos conjuntos de eventos, condiciones y acciones se los conoce comúnmente como ECA (**E**vent-**C**ondition-**A**ction). Una implementación de políticas debe permitir definir, almacenar y administrar ECAs, y de ser capaz de interactuar con otras capacidades, recibiendo eventos e invocando métodos. Por ejemplo una condición como "bajo ancho de banda", puede tener una acción asociada como "usar compresión". Entonces al recibir el evento de bajo ancho de banda, en base a las políticas se evaluara el mismo y de ser verdadero, se usará la capacidad de codificación, compresión donde sea posible.
- *Seguridad.* Para evitar las consecuencias de ataques maliciosos, aplicaciones con diseños pobres, y errores inadvertidos de usuarios, se deben tomar ciertas medidas de seguridad como ser: Sistemas y usuarios deben ser autenticados, autentificación se sistemas, usuarios y acciones deben ser autorizados, y acciones e interacciones deben ser auditadas.

Obsérvese que los requerimientos planteados son en gran medida requerimientos no funcionales, esto se debe a la naturaleza sumamente restrictiva implicada en un escenario móvil, y relacionada especialmente con aspecto de hardware. También debe notarse el alto nivel de abstracción de los mismos, a modo de ejemplo al tratar sobre administrar el contexto, específicamente la posición, no se hace mención al grado de precisión con que se debe conocer, solamente se la debe conocer.

2.3 MODELOS DE ADAPTACIÓN A MOVILIDAD

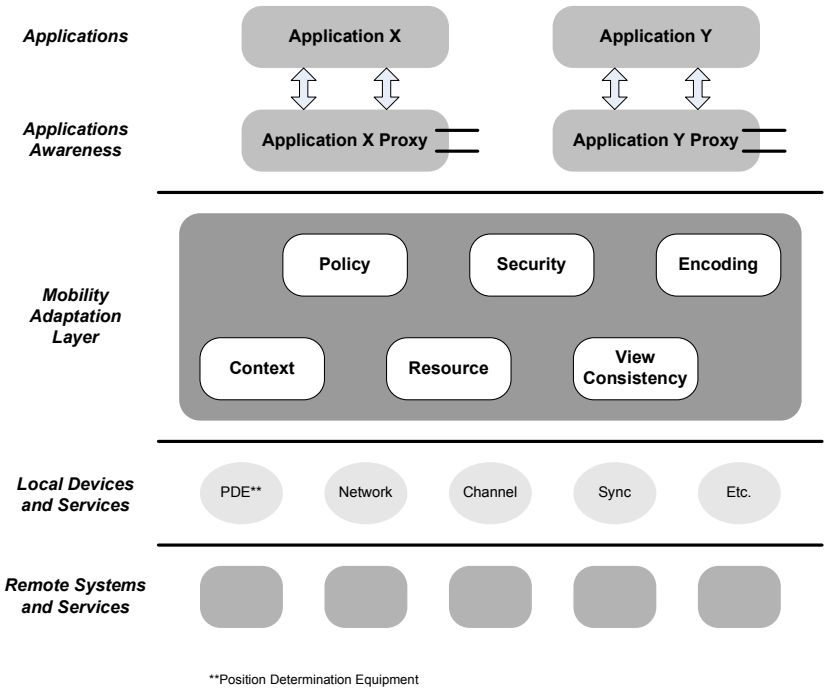
Un practica habitual en arquitectura de software o diseño es la de descomponer la funcionalidad total de una aplicación en partes, para luego asignar que función será llevada a cabo por uno u otro componente de software. Por ejemplo en una arquitectura cliente / servidor, la validación de los datos a impactar en una base de datos puede ser llevada a cabo en el cliente, en la base o en ambos. Algo similar ocurre cuando una aplicación debe lidiar con el tema movilidad, es decir que se puede descomponer la funcionalidad para luego analizar a que parte o componente se le asigna tal o cual función.

Un modelo de adaptación describe como una aplicación y fundamentalmente la plataforma ubica las responsabilidades para tratar los temas relativos a movilidad. Algunos modelos de adaptación a Movilidad conocidos [Bur03], son los siguientes:

- *Laisses-faire Application.* En este modelo cada aplicación es totalmente responsable de proveer el soporte para movilidad. Es decir debe encargarse de conectividad, sincronización de datos, etc. Se trata de las primeras aplicaciones en su tipo, donde se realizaban desarrollos a medida que luego resultaban ser poco flexibles y difíciles de escalar.

- *Application-aware collaboration*. Con este modelo la plataforma provee servicios comunes para que la aplicación pueda soportar movilidad. El rango de servicios va desde información y notificación, como por ejemplo tiempo de batería remanente, hasta funcionalidad transparente sobre administración de la conexión. En versiones mas avanzadas, varias aplicaciones utilizando una única plataforma, tienen conocimiento de la existencia entre ellas y cooperan en el uso de los recursos existentes. Ejemplo de estos son, el modelo propuesto por A. Dey, *Context Toolkit* [Dey00] y el modelo *Hydrogen Context-Framework* [Hof03] (ver apéndice 1 para más detalles sobre los mismos).
- *Application-Trasnparent system*. En este modelo la plataforma aísla a la aplicación de todo aquello relacionado a movilidad. La aplicación corre sin modificaciones y no es afectada por cambios externos. Este modelo es muy poco aplicable, su alcance solo se limita a aplicaciones muy simples.

De los modelos de adaptación descriptos, el "*Application-aware collaboration*" se presenta como el mas practico. La figura siguiente muestra la arquitectura lógica de este modelo. La capa *Mobility Adaptation* es común a todas las aplicaciones y las capacidades que implementa son compartidas también por todas las aplicaciones. Cada aplicación incluye un componente, "*Application Awareness*", encargado de interactuar con esta capa.



Tres áreas de aplicación son de particular interés para los usuarios en un ambiente móvil, teniendo en cuenta la frecuencia de uso, estas son: los portales, compartir documentos y el acceso a las bases de datos. En las siguientes paginas se describirá con un poco mas de detalle el significado de estas áreas en un ambiente móvil.

2.4 ARQUITECTURA DE PORTAL PARA APLICACIONES MÓVILES

La función primaria de un portal es la de agregar e integrar diversas y distribuidas fuentes de información, y presentar el resultado al usuario en una vista simple concisa y pertinente a través de un Web Browser.

Un portal es típicamente dirigido a un específico grupo o tipo de usuario. Por ejemplo en la Intranet de una compañía, el sector de atención al cliente puede acceder a información relacionada con clientes (promociones vigentes, descuentos, etc.), pero no puede acceder a información financiera, la cual solo estaría autorizada para los integrantes del sector de finanzas.

Un portal puede contener diversos tipos de contenidos:

- Datos relativamente estáticos, como banners, gráficos y estructura general.
- Contenido dinámico, información que cambia con cierta frecuencia, el caso de las promociones vigentes para el sector de atención al cliente estaría dentro de este grupo
- Información nueva o trascendente, como notificaciones o información incremental. Por ejemplo un notificación para el grupo de ventas que indique que se ha terminado el stock de un determinado producto

Un usuario puede estar autorizado a actualizar el contenido de un portal, por ejemplo, volviendo al sector de atención, actualizar algún dato de un cliente luego de haber mantenido un contacto.

La arquitectura de un portal abarca tres tipos de funciones:

a. *Fuentes de Información* – Las fuentes de información proveen de datos al portal, tanto bajo pedido utilizando un mecanismo “push” provisto por un middelware, o por un proceso periódico de extracción, transformación y carga. Las fuentes de información incluyen bases de datos, aplicaciones u otros portales externos al sistema.
b. *Funciones del Portal* – Las funciones de un portal son básicamente las de agregar y componer la información para luego ser entrada al usuario.
c. *Funciones Independientes* – Son tecnologías persistentes o componentes, como el Web Browser.

Los componentes comúnmente incluidos en un portal son los siguientes:

- *Web Browser* – Provee una interfase del portal al usuario, si se accede a través de Internet, un protocolo Proxy soporta la comunicación con el usuario y con el portal HTTP y HTML comúnmente mejorado del lado del cliente con el uso de scripting y/o código ubicado en el Browser como ActiveX o controles Java.
- *Server de Presentación* – Crea e integra vistas de contenido a través de la interacción con otros componentes.
- *Server de Aplicación* – Ejecuta cualquier código que sea requerido dinámicamente para extraer y reformatear información desde sistema no basados en Web.
- *Agregación e Integración de componentes* – Provee extracción y reformateo dinámico de información para productos estándares como EAI (*Entrerprise Application Integration*) o RDBMS (*Relational Database Management System*)
- *Administración de Contenido, búsqueda e indexación, y colaboración* - Administrar el ciclo de vida del contenido incluyendo modificaciones realizadas por el usuario, como creación, versionado, revisiones y comentarios.

- *Servicios de Personalización* – Disponible para que cada usuario pueda configurar la vista y el contenido que quiere tener cada vez que accede al portal
- *Seguridad* – Un requerimiento para toda arquitectura de aplicaciones móviles, es el de asegurar la integridad de información sensible en sitios remotos

Debido a que es una arquitectura centrada en un Server y el rol de integrador de la información, un portal es completamente dependiente de la conectividad de red. Una solución simple para aplicaciones móviles es la de permitir el acceso *offline* a sitios Web, bases de datos y archivos que han sido previamente descargados al dispositivo móvil. El usuario interactúa con los mismos, y una vez que la conexión de red se reestablece, las diferencias entre las copias locales y las remotas se sincronizan. Esta solución solo es factible para aquellos portales más simples. Si la fuente de datos esta asociada con un gran numero de otros sistemas, o simplemente no cabe dentro del dispositivo móvil, no podrá ser aplicada.

Entonces, sin conexión de red, la creación de contenido dinámico desde un portal y sus sistemas *back-end* en tiempo de ejecución es esencialmente imposible. Sin embargo existen algunas aproximaciones que pueden ser usadas para proveer una vista *offline* del contenido:

- Prealmacenado del contenido generado en el portal
- Replicación en el sistema móvil de los datos y el código usado para generar el portal y su sistema *back-end*.

La apropiada estrategia a utilizar dependerá de factores como cantidad de datos involucrados, la complejidad de la interacción del usuario con los datos, y la frecuencia necesaria de actualización de los mismos.

A continuación se presentará de que forma una arquitectura de portal móvil puede cubrir los requerimientos planteado al principio de este capitulo para caracterizar una aplicación móvil.

Clientes que soporten multiplataformas – Los portales usualmente soportan el acceso desde diferentes plataformas, manejan diferentes caracterizaciones de dispositivos, y cualquier transcodificación de contenido requerido. Como el contenido comúnmente es dinámico y el tipo de dispositivo del cliente impredecible, estas actividades ocurren en tiempo de ejecución.

Una aplicación cliente que soporte movilidad no necesita soportar transcodificación dinámica porque el tipo de dispositivo del cliente es estático. La aplicación no necesita manejar cambios dinámicos en la personalización del dispositivo *offline*, ya que se supone que el mismo será usado por una única persona.

Capacidad de trabajar offline – Las aproximaciones presentadas en párrafos anteriores son las que permitirán el trabajo *offline* en un entorno móvil:

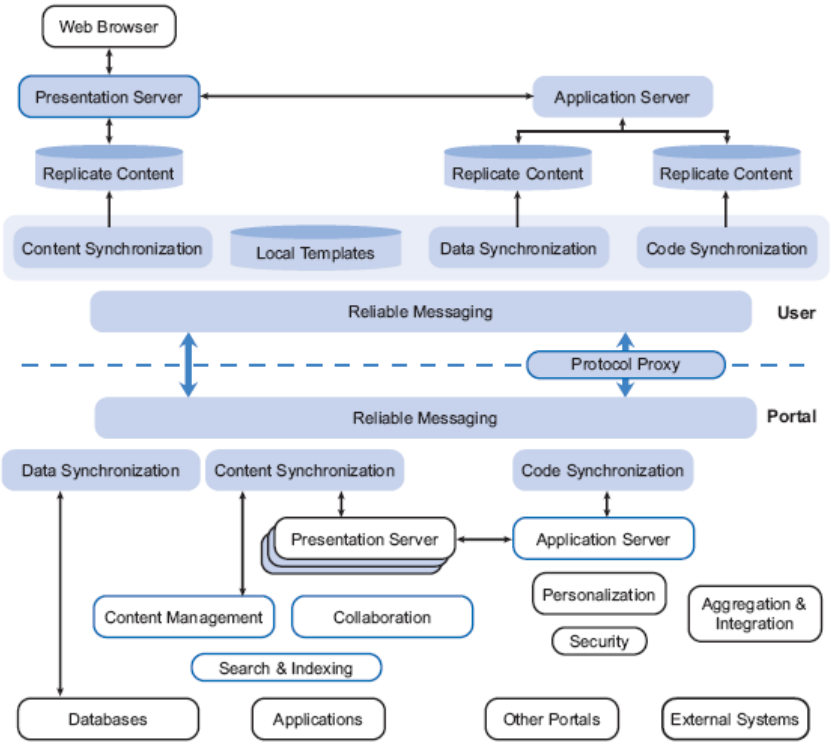
- *Prealmacenado de Contenido* – involucra el prealmacenado del contenido provisto por un portal en respuesta a un requerimiento hecho desde una URI, como una pagina Web. El código que genera el contenido no es prealmacenado. Por ejemplo un link puede ser referencia a un script JSP o ASP, el Server de aplicación corre este script y devuelve al cliente streams HTML. Estos HTML son los que están prealmacenados, no los scripts. Navegar el portal, siguiendo cada link y almacenar la salida en el sistema local para luego disponer del mismo *offline*, es un mecanismo completamente ineficiente. Además todas la paginas pueden no ser requeridas o algunas de ellas ser accesos a los sistemas back-end. Por lo tanto el prealmacenado de contenido debe realizarse bajo el control de la configuración local que especifique las páginas de interés o provea un criterio de selección.

- *Replicación de Código* – permite que el contenido del portal sea más dinámico. El portal puede ejecutar código, por ejemplo JAVA, en el proceso de servir el contenido al usuario o en la recolección y manejo de datos de otros sistemas. El código es replicado desde el Server al cliente. Alguna replicación involucra componentes de la interfase del usuario, la mayoría esta involucrado con la colección, manipulación y almacenamiento de datos.
- *Replicación de datos* – los datos pueden ser replicados del portal al cliente, del cliente al portal o en ambas direcciones. Si los datos solo puede tener permiso de escritura en el lado del cliente, la implementación se vuelve más simple, sin embargo la implementación que permite esquemas de múltiples copias que pueden ser actualizadas independientemente, se vuelen por demás complejas.
- *Emulación* – Cuando una URI es seleccionada en un Browser, por ejemplo, ingresando su URL, el Browser intenta contactar al Server especificado en la URI, y requerir los recursos, utilizando HTTP sobre TCP/IP. Si el Browser no puede contactar el Server, retorna un error. El uso de emulación enmascara la no disponibilidad de un recurso, de forma que los datos y el servicio continúen disponibles en forma local. Normalmente el Web Browser local intenta conectarse al Server de presentación a través de la red. Cuando el sistema esta *offline*, el Web Browser deberá redirigir hacia el Server de presentación local (residente en el cliente) el cual presentará el contenido prealmacenado.

Uso optimizado de energía y performance – Idealmente una implementación con las capacidades requerida para movilidad, manejará aspectos relativos a uso de la energía y performance, donde los distintos componentes de la misma deberán tener conocimientos en todo momento de niveles y tasa de consumo de los distintos recursos.

Conectividad Continua – Dos áreas están incluidas dentro de conectividad continua, estas son administración de conectividad de red y la seguridad desde el punto de vista del usuario. Por ejemplo el usuario no tendrá físicamente que re-autenticarse cada vez que el sistema se reconecta. La conectividad continua puede ser soportada por emulación, la cual provee la apariencia de que el recurso de red se encuentra disponible. Sin embargo la emulación no maneja tópicos relacionados con disponibilidad de la red o selección de la mejor red en función del contexto. En estos casos la conexión de red pasa a ser considerada un recurso, el cual es administrado por la capa *Movility Adaptation* del modelo planteado en páginas anteriores.

Una posible arquitectura de portal para aplicaciones móviles es la mostrada en la figura siguiente, la cual refleja varios tipos de modificaciones: agregado de nuevos componentes (a los habituales de un portal no móvil), mejorado de otros componentes preexistentes y por ultimo, partición y distribución de ciertas responsabilidades.



Las modificaciones del lado del Server, se corresponden con el agregado de dos tecnologías: sincronización y mensajería confiable.

Para soportar sincronización, el administrador de contenidos, el Server de presentación, el Server de aplicaciones, deben ser mejorados para que una de sus funciones sea la de mantener un log de modificaciones. Algunos componentes como el Administrador de contenido ya mantienen esta información lo cual también puede ser utilizado para la sincronización. Y por último las bases de datos que contenga el portal pueden también ser utilizadas para sincronización ya que en la estructura normal de una base de datos la sincronización se encuentra incorporada como algo necesario.

La mensajería confiable es una facilidad requerida para el intercambio de información entre el cliente y el Server.

Las modificaciones del lado cliente son las siguientes: al igual que en el Server son requeridos componentes de mensajería confiable y sincronización. Para la operación *offline*, se incorporan un componente para almacenamiento de datos, contenido y código, un Server de aplicación y por ultimo un Server de presentación.

El uso de un Server de aplicación y un Server de presentación en el cliente reduce la funcionalidad de una aplicación que debe ser diferente entre una versión que soporta movilidad y una que no. Dos diferentes versiones solo deberán cambiar la capa de presentación. El Server de presentación asegura que el Web Browser pueda correr sin modificaciones. El Server de Aplicación ejecutará el código replicado desde el Server.

En síntesis, un portal integra fuentes de información distribuidas y presenta los resultados en una simple y consistente vista a través de la interfase de un Web Browser. En un ambiente donde la conectividad es constante, las fuentes usadas por el portal para mostrar los datos están continuamente disponibles. En un ambiente móvil, existen varios métodos que son utilizados para proveer acceso a la información aun cuando el sistema del usuario este offline. Estos métodos incluyen prealmacenado de datos generados por el portal y replicación sobre el cliente de código y datos usados para generar el contenido del portal.

2.5 ARQUITECTURA DE DOCUMENTOS COMPARTIDOS PARA APLICACIONES MÓVILES

Uno de los usos primarios que se le da una computadora es el de almacenar datos en forma electrónica de manera tal que puedan ser recuperados cuando se los necesite. Normalmente los datos son almacenados en un archivo. Para que un archivo sea útil, debe ser asociado con una aplicación que comprenda tanto su estructura como los datos en el contenido. Desde este punto de vista una base de datos es un archivo cuya estructura y datos son administrados por una aplicación administradora de base de datos. Otros archivos, comúnmente llamados documentos, pueden contener texto, planillas de cálculos, presentaciones, video, audio, etc., son administrados cada uno de ellos por aplicaciones diseñadas para tal fin, por ejemplo los documentos de texto serán administrados por un procesador de palabras o por un editor de texto.

Cuando una copia del documento ha sido modificada y las modificaciones necesitan ser reintegradas al documento original, la solución comúnmente utilizada es la de sobrescribir la versión original en forma total, con la versión modificada. Con este nivel de reintegración no se hace necesario conocer la estructura interna del archivo y por lo tanto la aplicación que administra el documento no se ve involucrada en esta acción. Bajo esta óptica un documento es distinguido de una base de datos en la forma en que diferentes copias del archivo son reintegradas y sincronizadas. Un documento es definido como aquel archivo que es reintegrado en forma completa sin intervención de la aplicación que comprende su estructura.

Mientras que un documento es reintegrado a nivel de archivo, una base de datos, con la asistencia de la aplicación administradora de base de datos, puede ser reintegrada a un subnivel de archivo. Cuando dos diferentes versiones de una base de datos necesitan ser reintegrada, solamente la sección modificada es actualizada. Este subnivel de integración es posible porque la aplicación de bases de datos comprende la estructura interna de la base, y administra la actualización. Desde que el tamaño de las bases de datos a sido lo suficientemente grande, esta forma de reintegración es mas eficiente que la de sobrescribir todo el archivo.

Vale hacer notar que esta distinción entre documentos y bases de datos es algo arbitraria. En algunas ocasiones es requerido que la aplicación que administra documentos conozca la estructura interna del documento, especialmente en funcionalidades básicas para ambientes móviles. Por ejemplo adaptar un documento para un tipo particular de dispositivo móvil no podría lograrse sin este conocimiento.

El hecho de compartir documentos puede ser llevado a cabo de las siguientes maneras:

- SCDS (*Single Copy Document Sharing*) – De esta forma el usuario comparte el acceso al documento original. No son creadas múltiples copias del documento. Cualquier cambio al documento, incluyendo su destrucción, es absoluto.
  Una desventaja de este método es que al existir una sola copia del documento, solo un usuario puede acceder simultáneamente al mismo. Por otra parte una única copia del documento es un único punto de falla de la arquitectura. Si un usuario por descuido olvida dejar el documento disponible luego de haberlo usado, el mismo no estará disponible para el resto de los usuarios. Un ejemplo de esto lo constituye un File Server en ambiente Windows, donde los documentos alojados en el mismo, puede ser accedidos por todos los usuarios, pero solo uno puede abrirlo en forma de lectura / escritura, el resto al mismo tiempo, solo podrá acceder al documento en forma solo lectura y de realizar algún cambio y querer guardarlo deberá hacerlo en un nuevo documento. Dado el riego de no disponibilidad, y el resultado en la reducción de la productividad es alto, SCDS generalmente no es aceptado.

- MCDS (*Multiple Copy Document Sharing*) – Con este método cada usuario tiene una copia del documento. Ahora, la dificultad radica en mantener la información del documento actualizada. Si dos usuarios están utilizando un documento y uno de ellos realiza cambios en la información de su copia del documento, la información de la otra copia se vuelve incorrecta. Si los dos usuarios realizan cambios simultáneamente, ambas copias de vuelven incorrectas y deben ser sincronizadas o reintegradas.

Ahora bien, una arquitectura para compartir documentos debe llevar a cabo lo siguiente:

a. Permitir que la información en documentos sea compartida
b. Permitir que la información desde los documentos se encuentre disponible bajo demanda.
c. Mantener la concurrencia de la información en los documentos.

Entonces de los métodos descriptos anteriormente (SCDS & MCDS), y los requerimientos recién descriptos se puede concluir: SCDS previene el accedo por mas de un usuario al mismo tiempo, mientras que MCDS posee riesgo de que modificaciones realizadas en una copia del documento no se refleje en el resto de las copias. El limitado acceso a una simple copia tiene un serio impacto en la productividad. Sin embargo las desventajas de compartir múltiples copias son más fáciles de resolver, porque una arquitectura basada en este método puede proveer formas de sincronizar modificaciones entre múltiples copias del documento.

Para proveer la funcionalidad básica, una arquitectura de documentos compartidos debe contener los documentos originales, copia de estos documentos, almacenamiento de documentos compartidos, almacenamiento de imágenes de documentos (*shadow document*), y un sistema de administración que rastree la ubicación de los documentos y propague las modificaciones a cada copia en el sistema de archivos, resuelva conflicto de versiones automáticamente si es posible, y con la intervención del usuario de ser necesario.

Para proveer la disponibilidad de documentos mientras mantiene la coherencia entre copias, la arquitectura debe tener constante acceso a todas las copias de los documentos bajo su control. En otras palabras los documentos, las copias de los documentos y el sistema de administración deben estar constantemente conectados unos a otros. Si la conexión puede ser garantizada, el sistema puede permitir que los documentos sean compartidos, proveer continua disponibilidad de los documentos, y mantener la concurrencia.

Pero en un entorno móvil, una conexión constante entre documentos y el sistema de administración no puede ser garantizada por mucho tiempo. Entonces la arquitectura para compartir documentos debe ser modificada, para que aun con esta limitación permita que la información en los documentos pueda ser compartida manteniendo la concurrencia de los mismos. Es mas debe cumplir teniendo en cuenta los limitados recursos de hardware, como espacio de almacenamiento, típicos de un dispositivo móvil.

A continuación se analizará una posible Arquitectura para compartir documentos en ambientes móviles que abarque los requerimientos planteados al principio de este capitulo para caracterizar una aplicación móvil.

Clientes que soporten multiplataformas – La arquitectura debe proveer soporte a múltiples plataformas. Para que esto sea posible, debe ser capaz de identificar el dispositivo móvil y sus recursos disponibles, como ancho de banda, capacidad de almacenamiento, tipo de pantalla, etc., y en función de esto modificar el contenido del documento.

Capacidad de trabajar offline – Para permitir a los usuarios el trabajo *offline*, los documentos al ser accedidos debe ser prealmacenados en un archivo cache. Si el recurso de almacenamiento del dispositivo móvil así lo permitiera, todos los documentos a ser compartidos deben ser copiados del Server al sistema móvil. Sin embargo la mayoría de las plataformas tienen un espacio de almacenamiento limitado por lo que la arquitectura debe proveer la habilidad de un prealmacenado selectivo.

Además para lograr que los documentos compartidos este disponibles, la arquitectura debe ser capaz de reintegrar las modificaciones realizadas en la copia local, a la copia del Server y a todos las otras copias del sistema.

Uso optimizado de energía y performance – La arquitectura debe optimizar la capacidad del dispositivo en función de la energía de la batería, para extender los periodos de tiempo. Si la mejora de la performance viene aparejada a un mayor dejaste de energía, la arquitectura debe permitir al usuario el intercambio entre performance y uso de energía.

Conectividad Continua – Debe ser capaz de liberar al usuario de la preocupación de los problemas de conectividad. Debe sacar ventaja de la existencia de la conectividad lo más rápido posible y ser capaz de continuar sin ningún problema una vez que la misma se haya perdido. Por tal motivo la arquitectura debe implementar un sistema de mensajes confiable entre los componentes del sistema. Cuando múltiples caminos al Server estén disponibles, debe ser capaz de elegir la mejor conexión basada en confiabilidad, velocidad y calidad de servicio. Y finalmente, ante una conexión entre cliente y Server fluctuante, la arquitectura debe ser capaz de manejar una sesión basada en la seguridad.

Una posible estrategia de implementación para una Arquitectura que permita compartir documentos en un ambiente móvil, se presentará a continuación, distinguiendo los cambios y mejoras a realizar sobre el lado cliente, como el lado Server.

Los componentes del cliente se detallan a continuación junto con un grafico que muestra los mismos.

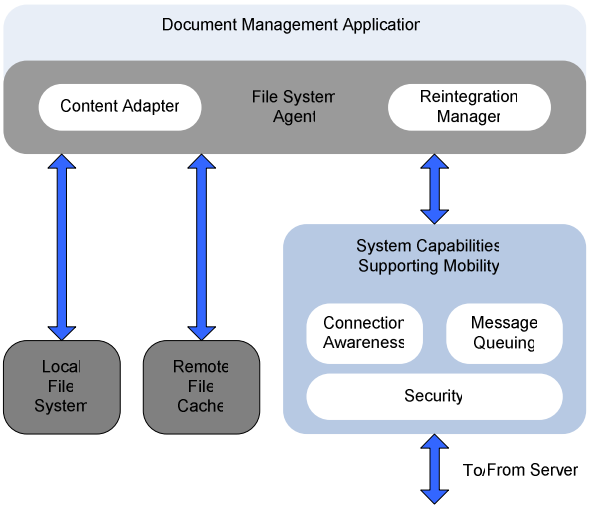
- *Document Management Application* – Se trata cualquier aplicación que haga uso de los documentos, como por ejemplo un editor de texto o un reproductor de mp3.
- *File System Agent* – Es el componente principal de la arquitectura, este administra todos los requerimientos de documentos realizados por el *Document Management Application*, y determina que documentos son locales cuales son remotos. Administra el contenido del cache, los documentos almacenados allí, en función de lo configurado por el usuario.

  El *Content Adapter* es parte del *File System Agent* y tiene la responsabilidad de adaptar el contenido de los documentos compartidos, a las características del dispositivo móvil. El contenido es adaptado en función de la configuración hecha por el usuario, las características del dispositivo y la estructura del documento.

  El *Reintegration Manager* es la parte del *File System Agent* responsable de resolver los conflictos de versiones. Siempre que la estructura del documento lo permita, tratará de reintegrar las versiones automáticamente. Cuando los conflictos no puedan ser resueltos automáticamente utilizará la aplicación asociada al documento y en última instancia recurrirá al usuario
- *Local File System* – Contiene todos los documentos locales. Generalmente tiene su propia aplicación de administración, la cual usualmente forma parte del sistema operativo.
- *Remote File Cache* – Contiene las copias de todos los documentos originados en los *file system*s remotos. Puede ser considerado una base

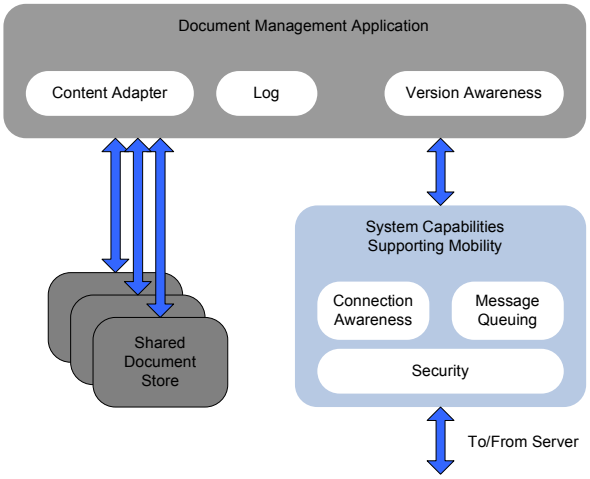
de datos para la cual el administrador de base de datos es el *File System Agent*. Puede estar almacenado en el *file system* local.

- *System Capabilities Supporting Mobility* – Comprende todas aquellas características relacionadas con la red y la energía. A su vez se lo puede considerar compuesto por los siguientes tres elementos.
  Connection Awareness, permite a la arquitectura conocer el estado de conexión de la red y la energía.
  Message queuing, es un elemento esencial para ocultar los estados de conexión de la red y asegurar la confiabilidad.
  Security, es un requerimiento para toda arquitectura distribuida. En una arquitectura de documentos compartidos para aplicaciones móviles, la seguridad esta relacionada con los nodos móviles, la variedad de protocolos inalámbricos, y lo poco confiable de la conexiones y sesiones.



Un Server hace que los archivos en su *file system* local estén disponibles para sus clientes. En una arquitectura de documentos compartidos, el Server puede ser a su vez cliente y el cliente puede también comportarse como Server. Por el lado del Server se tienen los siguientes componentes:

- *Document Sharing Application* – Los documentos que son compartidos deben ser administrados por este componente. Es responsable de comunicarse y manejar los mensajes de los *File System Agent* de cada cliente. Mantiene un log de todos los documentos cambiados y provee la funcionalidad de conocimiento de versiones. Este también incluye un *Content Adapter*. El log logrado por el seguimiento de todos los cambios realizados a los documentos, es usado en primera instancia cuando un cliente requiere una actualización del árbol de directorios de los documentos compartidos almacenados.
  Un *Content Adapter* en el Server provee la misma funcionalidad que el ubicado en el lado cliente, depende de los recursos disponible el hecho de en que lado será ubicado.
- *Shared Files* – Corresponde a los archivos compartidos disponibles para los usuarios
- *System Capabilities Supporting Mobility* – Características de conexión de red y energía, incluyendo conocimiento de conexión, manejo de cola de mensajes y seguridad. Este componente es similar al ubicado del lado cliente.



Del modelo de arquitectura planteado resulta interesante destacar algunos puntos que pueden variar en la implementación, y su análisis ayudará a comprender un poco más en profundidad la problemática entorno de las aplicaciones móviles:

Reintegración y Estructura de Documentos. Si el *Reintegration Manager* utiliza información provista por el *Document Managenement Application* sobre la estructura del documento, el mismo podría proveer una reintegración de sub-archivo, similar al provisto por una base de datos. Solamente los segmentos del documento que fueron modificados necesitarían reintegrarse. Esto permitiría un uso más eficiente de recursos como la red o la capacidad de almacenamiento. Sin embargo se necesitaría recursos de procesamiento más robustos.

La distinción entre documentos y bases de datos se diluiría aun mas si el propio *file system* fuese visto como una base de datos y los archivos en el *file system* como objetos en una base datos. Llegando aun más allá, un documento podría ser visto como una compleja colección de objetos en una base de datos. Si la aplicación de manejo de documentos se integrará con la aplicación de administración de la base de datos y esta a su vez con la de administración del *file system*, la distinción entre documentos y bases de datos virtualmente desaparecería. Une ejemplo de esto pero para ambientes no móviles lo constituyen herramientas como Panagon, donde los documentos son almacenados en una base de datos MS SQL y la herramienta se encuentra integrada al *file system* de un servidor MS Windows.

La complejidad de la estructura de los documentos debe ser considerada. La reintegración automática puede ser posible en documentos que tiene estructura rígida como los Adobe Acrobat Form. En el otro extremo documentos con muy pequeña estructura como los mp3 una reintegración a un nivel menor que archivo total resulta casi impracticable.

Ubicación del Adaptador de Contenido. El adaptador de contenido puede ser ubicado en el sistema Cliente o en el sistema Server. En cualquier de las dos posibilidades requiere acceso a lo siguiente:

- *Recursos del Sistema* – Desde el punto de vista utilización de recursos de red, la mejor ubicación es en el Server. Los documentos en el Server típicamente son de mayor contenido, que los que se transfieren a los clientes con dispositivos de recursos limitados. Para realizar esta adaptación se requiere de un poder de cómputo seguramente disponible con mayor frecuencia en el Server.
- *Conocimiento del formato del documento* – El mejor recurso para conocer el formato de un documento es la aplicación que maneja ese documento. En un Server no es necesario que la aplicación asociada a un documento este instalada en el mismo para que el documento pueda

ser almacenado. En cambio en el cliente, para un documento que fue requerido, seguramente se dispondrá de la aplicación para abrirlo. Por lo tanto ubicar en el adaptador de contenido en el cliente garantiza el acceso a la aplicación que lo administra y por ende a la información de la estructura del mismo.

- *Conocimiento de los recursos del sistema destino* – Para que el Server pueda tener conocimiento sobre los recursos disponibles en todos sus clientes, necesitará de una base de datos donde almacenar la información de los recursos de los clientes, o dicha información deberán ser entregada al Server, por el cliente, cada vez que requiera un documento. Sin embargo cuando el adaptador de contenido esta ubicado en el cliente, este hereda el acceso a la información de recursos del cliente.

La mejor solución puede ser la de tener una arquitectura lo suficientemente flexible como para ubicar el adaptador de contenido en el cliente o en el Server, en el tiempo que se comportan los archivos.

Sincronización del árbol de directorio completa o incremental. La arquitectura de documentos compartidos debe asegurar la coincidencia, entre la versión del documento en el Server y la versión en el cache del cliente remoto. Para esto el cliente regularmente requiere una instantánea del árbol del directorio del Server la que incluye la información de versiones de cada archivo.

Una sincronización completa garantiza la no disparidad entre los datos del Server y el cliente. Sin embargo un gran número de documentos puede ocasionar demoras inaceptables en el intercambio de información.

Una sincronización incremental contiene solo los cambios realizados desde la última sincronización. Este método si bien mas eficiente puede introducir posibles perdidas de coincidencia.

Server con conocimiento o sin conocimiento de sus clientes. Se refiere a si el Server debe mantener la información de estado referida a que clientes están utilizando cual archivo compartido. Mantener una base de datos de usuarios y los documentos que están compartiendo puede permitir enviar una actualización inmediatamente de ocurrido un cambio en un documento a todos los usuario que lo estén utilizando sin tener que esperar una requerimiento de actualización de cada uno de ellos. En un ambiente donde la conexión entre los clientes y el Server esta garantizada este esquema es ventajoso ya que los cambios son enviados a través de la red inmediatamente después de ocurridos. En un ambiente móvil, sin embargo, las actualizaciones solo podrán ser realizadas cuando el cliente este conectado a la red.

En una arquitectura de documentos compartidos para aplicaciones móviles, el proceso de *polling* por parte del cliente para detectar cambios parece ser lo suficientemente eficiente como para evitar la complejidad de un Server con conocimiento de todos sus clientes y archivos que los mismos comparten momento a momento.

A modo de resumen, una arquitectura de documentos compartidos, permite mantener la coincidencia entre las copias de un documento compartido en el sistema. En un ambiente en el que se dispone de conexión constantemente la coincidencia puede lograrse de varias maneras, por ejemplo a través del uso de permisos. En un ambiente donde los clientes no siempre están conectados, la arquitectura debe proveer mecanismos para sincronizar las copias modificadas por integración de los cambios en una versión común del documento.

2.6 ARQUITECTURA DE BASES DE DATOS PARA APLICACIONES MÓVILES

Los usuarios tradicionales de una base de datos acceden a los datos residentes en el Server de bases de datos desde sus equipos clientes conectados físicamente a la red. Los datos se presentan en la maquina cliente como una simple vista de los datos residentes en el Server. Esta particular arquitectura es segura pero al mismo tiempo limitada en el hecho de que los usuario no pueden ver o trabajar con los datos sin una conexión a la red. Todo procesamiento tiene lugar en el Server, construido específicamente para tal propósito.

En la forma más simple una base de datos es un archivo que contiene varios registros de datos. En un ambiente cliente/servidor tradicional, mas de un usuario puede utilizar la misma base de datos simultáneamente. RDBMS hace esto posible a través del uso de mecanismo interno de locking que previenen que más de un usuario modifique un registro al mismo tiempo.

Una arquitectura de base de datos preparada para un ambiente móvil, permite a los usuarios acceder a la información en cualquier momento y desde cualquier lugar.

En un ambiente móvil, copias de los datos pueden existir en distintos sistemas clientes. Dado que estos sistemas clientes no están continuamente conectados a la base de datos central, el RDBMS de dicha base no es capaz de prevenir cambios simultáneos a los datos por más de un usuario. Por otra parte, los datos locales en cada sistema cliente deben ser periódicamente sincronizados con los datos de la base master.

Algunos de los desafíos al diseñar una arquitectura de bases de datos que soporte aplicaciones móviles, son los siguientes:

- Los datos en los sistemas cliente se desactualizan durante los periodos en que el cliente no esta conectado. Los mensajes referentes a actualizaciones pendientes no estarán disponibles mientras el sistema este desconectado, esto introducirá mas dudas sobre la valides de los datos.
- La resolución de conflictos se volverán más desafiantes y ya no estarán bajo el control del RDBMS.
- El poder de procesamiento local en los clientes puede ser limitado en comparación al poder de procesamiento disponible en el Server.
- Los datos propietarios, deben mantenerse seguros en las ubicaciones remotas.

Un usuario móvil debe ser capaz de seleccionar los datos a replicar en el sistema cliente para su uso cuando el sistema este desconectado de la red. La replicación de la base de datos completa no debe ser permitida, se debe limitar al usuario aun arbitrario conjunto de datos. Teniendo en cuento lo expuesto en párrafos anteriores, la cantidad de datos a replicar solo debe ser limitada por el espacio de almacenamiento disponible en el sistema del usuario.

Las desconexiones cliente/servidor deben ser transparentes al usuario. La aplicación cliente debe continuar teniendo un buen comportamiento y los datos continuar disponibles para el usuario.

Un usuario necesita saber si los datos que va a utilizar en un ambiente *offline* son viejos, irrelevantes o transitorios. El usuario debe ser capaz de basar sus decisiones en estos datos, pero los mismos deben ser marcados de forma que la decisión resultante pueda ser actualizada cuando los datos vuelvan a estar disponibles *online* nuevamente.

Una arquitectura de base de datos para aplicaciones móviles debe garantizar que las transacciones serán trasmitidas confiablemente. Durante una transacción normal, una conexión de red es establecida entre el cliente y el Server y la transferencia de datos

es iniciada. Cuando la transferencia de datos se completa, una notificación sobre si la transferencia fue realizada con éxito o no es enviada al que la inició. La falla o el éxito de la transacción, no debe limitar el trabajo que el usuario puede hacer.

Por ejemplo, si el dispositivo esta conectado a la red y actualiza un campo de datos, la transacción será trasmitida al Server inmediatamente. Si la conexión se pierde durante la transmisión, la transacción será encolada para ser transmitida cuando la conexión sea reestablecida. Mientras tanto el usuario debe poder ser capaz de hacer referencia a la actualización aunque la transmisión no se haya completado.

La transmisión de los datos encolados debe ser llevada a cabo sin intervención del usuario. Sin embargo el usuario debe ser capaz de definir políticas sobre la disponibilidad de la red basada por ejemplo en costos, velocidad y recursos como tiempo de vida de la batería. El administrador de conexiones debe ser capaz de demorar las transmisiones en función a estas políticas.

Se trataran ahora algunas estrategias de implementación.

Visión global de la Arquitectura. El componente central de una arquitectura de bases de datos *offline* inteligente, es un *Database Proxy* que actúa como un interceptor entre la lógica de aplicación y el cliente local de base de datos. Toda actividad de base de datos es administrada por el *Database Proxy.*

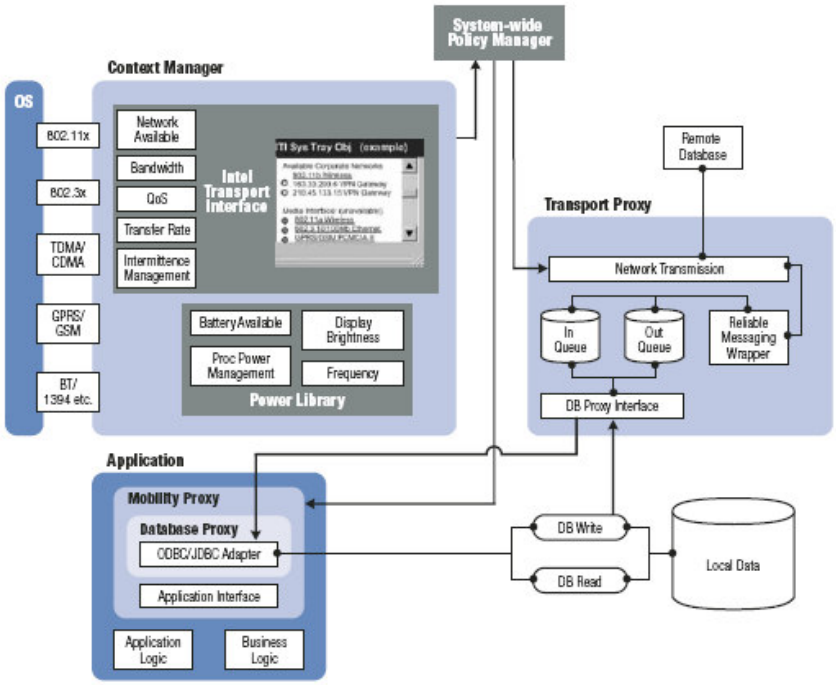
El *Database Proxy* es parte del *Mobility Proxy*, el cual conoce como comunicar el *Database Proxy* con el *Transport Proxy*. El *Transport Proxy*, el *System-Wide Policy Manager* y el *Context Manager* están activos aun cuando la aplicación no esta ejecutándose.

El *Transport Proxy* es responsable por la confiabilidad de la mensajería desde y hacia la base de datos maestra. Tiene conocimiento de los recursos del sistema cliente, como conectividad, vida de la batería, entre otros, basado en la información suministrada por el *Context Manager*.

El *Context Manager* detecta cambios en el sistema y notifica al *Transport Proxy* de acuerdo a como el usuario ha definido las acciones a llevarse a cabo.

Las políticas son almacenadas en el *System-Wide User Policy Manager*, el que es usado por cualquier aplicación en el cliente capaz de funcionar en un ambiente móvil.

Estos componentes pueden verse en el siguiente grafico y serán descriptos más en detalle, en función de su ubicación del lado del cliente o del lado del Server.



Componentes del Cliente. Los siguientes componentes son considerados necesarios para lograr confiabilidad en un sistema *offline* de bases de datos para aplicaciones móviles:

- *Transport Proxy* – Consiste en un sistema confiable de mensajes y un repositorio para mensajes entrantes enviados desde fuentes externas. El *Transport Proxy* se encuentra accesible para todas las aplicaciones y actúa como un servicio siempre disponible para enviar y recibir mensajes en función del estado de la red. Cada aplicación puede registrarse a través del *Database Proxy*, y esto le permite usarlo como un servicio común, manteniendo la propiedad del mensaje.
- *Policy Manager* – Utiliza el *Context Manager* para determinar el estado de conectividad de la red. Recibe mensajes de notificación referidos al contexto, como que adaptador de red esta disponible, el estado y calidad de una conexión, o la energía disponible en la batería. El *Policy Manager* utiliza esta información para tomar decisiones en función a un juego de políticas establecidas por el usuario basadas en factores externos como ser costos y ubicación. Por ejemplo un usuario puede tener una política que indique un mensaje con prioridad alta sea enviado ni bien este disponible la red, sin importar el costo o el consumo de energía que este implique.
- *Context Manager* – Monitorea constantemente el estado de los distintos recursos a administrar por el sistema. Por ejemplo para la conectividad de red el *Context Manager*, monitorea y genera datos en relación a la calidad de servicio, ancho de banda de todos los dispositivos de red, para que luego puedan ser usados por el *Policy Manager*.
- *Database Proxy* – Es una capa que interactúa entre la aplicación y la base de datos, y entre la base de datos y las colas de mensajes del *Transport Proxy*. Cuando la aplicación requiere datos de la base, el requerimiento pasa a través del *Database Proxy* el cual lo adapta vía las interfaces ODBC o JDBC, la información luego es recuperada de la base local. El *Database Proxy* esta constantemente comunicado con la

interfase de la aplicación la que le informa que datos están siendo usados. Cuando el sistema esta *online*, es el encargado de registrar los métodos de comunicación con el *Transport Proxy*, los que este ultimo utiliza para guardar datos en el almacenamiento local. Esto permite al *Database Proxy*, conocer en todo momento cuales son los datos nuevos en la base local.
Dado que le *Database Proxy* conoce cuales son los datos que se encuentran actualmente en uso, y cuales datos han sido cambiados, puede enviar mensajes en base a las políticas definidas por el usuario, a la aplicación. Estos mensajes permiten a la misma tomar acciones como actualizar los datos actuales.

- *Client Mobility Proxy* – Es un comunicador centralizado entre la aplicación y los distintos Proxys. A través de la interfase con la aplicación es notificado de los distintos eventos, por ejemplo de una actualización de la base de datos. El usuario tiene la opción de forzar la sincronización a través de la descarga de toda la información a la que esta subscripto o a parte de ella.

Componentes del Server. El lado del Server de la arquitectura de base de datos para aplicaciones móviles es similar al del lado cliente, con algunas excepciones, a sabe:

- *Transport Proxy* – Del lado del Server, el *Transport Proxy*, mantiene un seguimiento de las direcciones de destino de cada cliente. Se comunica con todo los servicios *sing-on* y verifica la identidad del cliente, mantiene un mapeo entre el cliente y su dirección IP. Cuando la dirección IP cambia, todos los paquetes en la cola de salida son detenidos de entrega, y luego redirigidos a la dirección de red correcta.
- *Database Proxy* – En el Server, el *Database Proxy* utiliza ODBC, JDBC o cualquier otro adaptador para administra las actualizaciones desde la cola de entrada, a la base de datos maestra. Los ítems en la cola tienen variadas prioridades basadas en circunstancias definidas por la lógica del negocio y/o políticas del usuario. Por ejemplo un usuario puede decidir que todo dato que transmita al Server resulte en una sincronización forzada, ya que el mensaje fue marcado como de máxima prioridad, o cuando información crucial deba ser enviada desde el Server, El Server fuerce máxima prioridad en los paquetes de datos que serán enviados.
Un proceso típico para envío de datos es el siguiente: Cuando los datos a los que esta suscripto el cliente sufren un cambio, un mensaje es enviado al cliente con una alerta de los cambios. En función de las políticas definidas por el usuario, esto resultará en una acción que detendrá o permitirá la transacción de actualización empujada desde el Server.

Cabe destacar las limitaciones de la solución que la arquitectura propuesta presenta. No incluye una formal definición de la resolución de conflictos, ni de habilidades de seguridad. Solo se hace mención las mismas pero no se entra en detalles. Por otra parte no provee detalles sobre como manejar los puntos mas conflictivos relacionados a sincronización, como operaciones que dependen de datos que ya no son validos o están en conflicto.

A modo de síntesis. Una base de datos provee a una colección de datos de una organización para que pueda se fácilmente accedida, administrada y actualizada. En un ambiente donde la interconexión esta garantizada, los datos son mantenidos en una base de datos central controlada por un RDBMS. En un ambiente móvil los datos son replicados en el sistema cliente para que puedan estar disponibles cuando el usuario

este desconectado. La integridad de los datos es mantenida a través de actualizaciones automáticas realizadas cada vez que el cliente vuelve a estar conectado.

2.7 RESUMEN CAPITULO

En el presente capitulo se realizó una introducción en forma genérica de las aplicaciones móviles. Se planteó una serie de requerimientos que un aplicación móvil debería cumplir, con el solo objetivo de caracterizar las mismas y analizar la problemática entorno de la movilidad. Se hizo mención a los modelos de adaptabilidad conocidos actualmente. Y Por ultimo se trataron las tres áreas de aplicación comúnmente utilizadas en movilidad, describiendo una arquitectura ejemplo para cada caso y detalles de implementación para cada una de las mismas.

CAPITULO 3

CONSIDERACIONES DE DISEÑO EN APLICACIONES MÓVILES

Las aplicaciones móviles difieren ampliamente de las aplicaciones de escritorio. Los usuarios no las utilizan de la misma manera, no esperan las mismas capacidades y hasta en algunos casos buscan utilizarlas en forma complementaria. Poseen características propias que la hacen diferentes del resto de las aplicaciones, comúnmente relacionadas con el entorno de ejecución. Todas estas cuestiones se verán reflejadas al momento de tomar decisiones sobre la arquitectura de las mismas.

3.1 INTRODUCCIÓN

La experiencia del usuario de aplicaciones móviles es significativamente diferente a la de un usuario de aplicaciones de escritorio, por razones obvias como las restricciones impuestas por el dispositivo y la red, y otras no tan obvias tales como y donde la gente utiliza las aplicaciones móviles y que expectativas tienen de las mismas.

En el capitulo 2 se analizaron las aplicaciones móviles dentro de un alto nivel de abstracción. En este capitulo se verán cuestiones mas relacionadas con el uso de este tipo de aplicaciones y las soluciones que se han desarrollado para cubrir las necesidades, pero desde un punto de vista menos abstracto, donde en algunos casos se presentaran ejemplos concretos.

Los tópicos fundamentales que se trataran son:

- ***Patrones de Uso***
  - ***Tiempo de Inicio.*** Comúnmente una aplicación móvil es utilizada en periodos cortos, por lo que el tiempo que la misma se tarde para estar operativa se vuelve crucial.
  - ***Foco en objetivo.*** La mayoría de las aplicaciones móviles son orientadas a cumplir un objetivo simple y específico a diferencia de las de escritorio que son más de carácter amplio y exploratorio.

- ***Interfase de Usuario***
  - ***Interacción con datos.*** El uso de las aplicaciones móviles ha evolucionado y las personas quieren utilizarlas en algunos casos como reemplazo de aplicaciones de escritorio, pero estas tienen otro tipo de restricciones que deben ser consideradas.
  - ***Influencia del contexto en el uso de la aplicación.*** Se refiere a como el contexto influye en la forma de interactuar con una aplicación

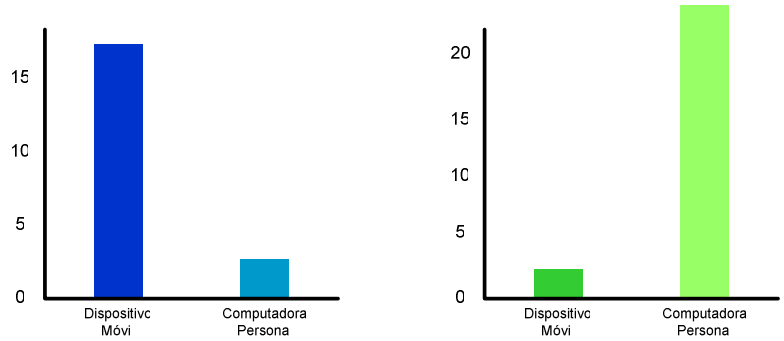
- ***Interconexión de dispositivos***
  - ***Selección de métodos de transferencia.*** Actualmente existen varios métodos de transferencia de información desde y hasta el dispositivo móvil, la selección del adecuado contribuye a alcanzar el objetivo de la aplicación móvil
  - ***Acceso a datos críticos.*** Existen datos que deben permanecer accesibles aun en situaciones en que no se cuente con acceso a la red.

- ***Seguridad.*** La seguridad en ambientes móviles no es un tema menor, que debe ser correctamente tratado. Actualmente los sistemas operativos utilizados dispositivos móviles proveen una serie de características que pueden ser incorporadas en las aplicaciones que corren sobre los mismos.

- ***Requerimientos de Confiabilidad.*** Desde el punto de vista de la confiabilidad, paradójicamente los dispositivos móviles se parecen mas a servidores que a computadoras de escritorio. Las aplicaciones que corren sobre este tipo de dispositivos, habitualmente lo hacen 24 horas al día, 7 días a la semana; deben manejar errores inesperados como intermitencia en la conexión de red; disminución de la carga de la batería, etc.

3.2 PATRONES DE USO

Los patrones de uso difieren fundamentalmente entre dispositivos móviles y PCs. Los usuarios tienden estar sentados frente a sus computadoras de escritorio en sesiones de larga duración, utilizando un teclado, pantallas de gran tamaño, un disco rígido, y otros periféricos, para crear y editar grandes cantidades de información. En contraste, las personas utilizan los dispositivos móviles con mayor frecuencia pero en periodos de corta duración, comúnmente para revisar o actualizar pequeñas partes de información. Por ejemplo buscar un número telefónico o revisar su próximo compromiso [ZoP03]. El siguiente grafico ilustra lo descripto.



Accesos por día

Tiempo promedio de sesión en minutos

3.2.1 Tiempo de Inicio

Un tiempo de inicio corto es una característica importante para las aplicaciones móviles. Dado que los usuarios tienden a utilizar frecuentemente sus dispositivos móviles en sesiones de corta duración, se hace imperativo que la aplicación se encuentre lista para ser empleada en el menor tiempo posible. Puede ser algo incomodo el hecho de esperar que un procesador de texto o cualquier otra aplicación se tome algunos segundos para iniciar en un computadora de escritorio, pero comparado con el tempo que se estará utilizando la misma, estos segundos de inicio no son significativos. Para una aplicación móvil donde el usuario quiere usarla por 20 segundos para verificar o actualizar una pieza de información, esperar un inicio de 6 segundos puede ser un tiempo completamente inaceptable. Por lo que podría concluir que una buena práctica es hacer que el tiempo de inicio de la aplicación sea despreciable frente al tiempo que tomara una sesión de uso. Las aplicaciones móviles tienen una

sesión de uso corta por lo que proporcionalmente se requiere un tiempo de inicio corto.

Una técnica habitual para de acelerar el tiempo de inicio es la de guardar la ultima sesión del usuario para luego devolverlo al mismo punto donde la dejo. Esto puede ser valido para algunas aplicaciones no todas. Por ejemplo algunos juegos para dispositivos móviles, que también son de corta duración a diferencia de los juegos para PCs, permiten guardar la sesión para continuarla mas tarde.

Otras aplicaciones como la agenda de contactos de un teléfono celular esta siempre corriendo para permitir el acceso a los datos en forma inmediata sin ningún tiempo de inicio de por medio.

3.2.2 Foco en objetivo

Enfocar en el objetivo es otra característica común de las aplicaciones móviles. La aplicación debe tener bien defino cuales es el conjunto de tareas que realiza; las cuales deben poder hacerse con el menor numero de ingreso de comandos por parte del usuario, y lo mas rápido posible. La importancia del foco en el objetivo es puesta de manifiesto, en los dispositivos móviles, con la existencia de botones especiales que realizan tareas específicas, por ejemplo el botón de acceso a la aplicación de mensajería de un teléfono celular.

Durante el desarrollo de una aplicación móvil, es necesario esforzarse en identificar las tareas comunes a realizar para que las mismas sean sencillas y absolutamente posibles de llevar a cabo por el usuario. Por ejemplo, si la aplicación móvil debe permitir al usuario el ingreso de fechas rápidamente, se debe asegurar que la selección de las mismas sea fácil de hacer y medir la efectividad de este proceso. Otro ejemplo seria una aplicación que permita a los usuarios escoger una ubicación en una mapa regional para orientarlos en como llegar a un lugar especifico, se debería asegurar que el usuario pueda hacer esto lo mas rápido y fácil posible, por ejemplo minimizando los datos de opciones que deba completar.

3.3 INTERFASE DE USUARIO

Ha ocurrido un cambio significativo en la forma que las personas utilizan sus dispositivos móviles, y esto se debe en gran parte al incremento en la prestaciones que han experimentados los distintos dispositivos móviles. Por ejemplo, el rol mas común de una PDAs era el de ofrecer aplicaciones de agenda, hoy se tienen aplicaciones que permiten tareas como manejo de e-mails, edición de documentos, entre otras.

Los usuarios ya no realizan la mayoría de las tareas sus computadoras de escritorio, para luego al estar en movimiento, conformarse con tener acceso a una simple vista de la información. Hoy en día esperan dispositivos móviles que los ayuden a administrar la información, analizar y seleccionar datos, y realizar transacciones, sin tener que espera a regresar a sus computadoras de escritorio. Por estos motivos, las interfases con las que el usuario debe interactuar juegan un papel importante al momento de tomar decisiones de diseño.

3.3.1 Interacción con datos

Existen dos aspectos a tener en cuenta en los dispositivos móviles, referidos a interacción con datos:

- ***Características de la pantalla.*** Debe presentar la información en un formato legible. Esto es en parte función del tamaño de la pantalla, este debe ser lo suficientemente grande para permitir el uso de menús, check boxes, botones, gráficos y otras técnicas de información. Y por otra parte es función de cuan eficientemente es utilizada la pantalla. Solamente los controles relativamente inmediatos a la información que se esta mostrando, deben estar visibles, el resto deberían permanecer ocultos hasta que sean necesarios.
- ***Modos de ingreso de datos disponibles.*** Para un trabajo eficiente de la información que se esta mostrando los usuarios deben disponer de una variedad de modos de ingreso. Actualmente se puede optar por plataformas que soportan entradas infrarrojas, *note pad*, teclados en pantalla, *touch screen*, etc.

La aplicación en si necesita ser diseñada para trabajar rápidamente en una pantalla de dimensiones reducidas. Por ejemplo el sitio de noticias clarín.com cuando se lo accede desde una computadora de escritorio, ofrece un gran despliegue de gráficos, vínculos, posibilidad de acceder a contenido multimedia, entre otras funcionalidades. Mientras que se si lo accede desde un dispositivo móvil, como ser un teléfono celular, el mismo presenta solo un grupo de titulares en forma de links que luego el usuario puede navegar para acceder al contenido de la noticia. El desarrollo de una aplicación para un fin específico puede utilizar un enfoque similar.

T9 es un ejemplo de una buena solución a una restricción de un dispositivo móvil. Permite el ingreso rápido de texto en un teléfono celular el cual tiene las 12 teclas características de un teclado telefónico. Anterior a T9 los usuarios ingresaban sentencias de texto letra por letra presionando cada tecla las veces que fuese necesario para conseguir la letra requerida, por ejemplo una “s” requería presionar cuatro veces la tecla “7”. Por otra parte, si la palabra a ingresar tenia letras que se encontraban en la misma tecla, una vez seleccionada la primera se debía esperar un momento para comenzar la selección de la segunda letra. T9 predice que quiere escribir el usuario en base a un diccionario y sugiere la siguiente letra o letras. Produce un ahorro de tiempo, ya que no se debe esperar para ingresar otra letra, aun en el caso de que esté en la misma tecla. Así mismo tiene la posibilidad de aprender aquellas expresiones que el usuario utiliza con frecuencia e incorporarlas al diccionario. Es una solución pensada exclusivamente a un problema propio de una aplicación móvil, que si bien no es adecuada para escribir un libro, lo es para el ingreso de mensajes de texto.

En la siguiente tabla se muestra el ejemplo de ingresar el texto “*estoy llegando*”. Obsérvese la cantidad de veces que se presionan las distintas teclas con y sin T9

Letra deseada	*Tecla presionada sin T9*	*Tecla presionada con T9*
E	3,3 = e	3 = e
S	7,7,7,7 = s	7 = s
T	8 = t	8 = t
O	6,6,6 = o	6 = o
Y	9,9,9 = y	9 = y
"espacio"	# = espacio	# = espacio
L	5,5,5 = l	5 = l
L	5,5,5 = l	5 = l
E	3,3 = e	3 = e
G	4 = g	4 = g
A	2 = a	2 = a
N	6,6 = n	6 = n
D	3 = d	3 = d
O	6,6,6 = o	6 = o

Una descripción completa de T9 puede encontrarse en http://www.t9.com/

3.3.2 Influencia del contexto en el uso de la aplicación

Es de destacar como el contexto en el que son utilizadas las aplicaciones móviles influye en la forma que los usuarios interactúan con la misma.

Supóngase la situación donde el usuario con su dispositivo móvil se encuentran en un ambiente público o ruidoso. La alternativa de dotar a la aplicación con la capacidad de ser operada a través del reconocimiento de voz no siempre es una buena idea aun en los casos donde tecnológicamente sea posible. Por otra parte es también importante que el uso del dispositivo móvil no moleste a las personas de su entorno. Un tren lleno de personas gritándole a sus dispositivos móviles no es precisamente un buen escenario.

Una pantalla "*touch-screen*" ofrece un rico entorno para aplicaciones de navegación, pero si desarrollan interfaces pequeñas con gran cantidad de opciones, las mismas serán difíciles de utilizar en un entorno móvil, donde existen saltos y vibraciones como por ejemplo dentro un vehiculo en movimiento

Otro punto a tener en cuenta es la forma de operación habitual del dispositivo. Varios dispositivos fueron pensados para ser operados con una sola mano. La mayoría de los teléfonos celulares adhieren a este criterio. Algunos otros fueron pensados para que sean sostenidos con una mano y operados con la otra, tal es el caso de la mayoría de las PDAs. Las laptops generalmente requieren dos manos y una superficie plana para un uso eficiente. Una aplicación móvil debe seguir el paradigma del dispositivo para el que será desarrollada, por ejemplo no construir una aplicación que requiera dos manos para su uso si la misma correrá en un teléfono celular, el cual las personas típicamente lo operan con solo una mano.

Nuevamente un ejemplo lo constituye T9, la cual provee una solución que se ajusta al paradigma del tipo de dispositivo para el cual fue diseñada.

3.4 INTERCONEXIÓN DE DISPOSITIVOS

El método de conexión que el dispositivo móvil pueda establecer con la red y con otros dispositivos, tendrá una influencia directa sobre las consideraciones de diseño. Dentro de las posibilidades de conexión a través de cables o sin ellos, se tiende al uso

de redes inalámbricas, especialmente dado la proliferación que estas han tenido en los últimos tiempos.

En la actualidad los métodos comúnmente disponibles son:

1. Infrarrojo (IR) o conexión por cable entre dos dispositivos móviles o entre el dispositivo móvil y el teléfono celular.
2. Cuna de sincronización (*Synchronization Cradle*) conectada a un computadora de escritorio o directamente a la red corporativa
3. Conexión Dial-Up a la red corporativa
4. Conexión inalámbrica a la red corporativa
5. Conexión de red inalámbrica a través de un teléfono celular u otro dispositivo con capacidades inalámbricas
6. LAN inalámbrica (IEEE802.11b) dentro de edificios de oficinas, aeropuertos, hoteles, etc.
7. Redes inalámbricas de área personal (PAN: *Personal Area Network*) basadas en el estándar *Bluetooth*

3.4.1 Selección de métodos de transferencia

Como ya se mencionó, un usuario móvil trabaja diferente a como que lo hace un usuario en su computadora de escritorio. Mientras que los últimos utilizan sus PCs dos a tres horas un par de veces al día, los usuarios móviles los hacen en sesiones de corta duración en varias oportunidades y en la mayoría de las veces con cierto apuro.

Se vuelve imperativo que, de necesitarse una transferencia de información, la misma se haga en el menor tiempo posible, se querrá entonces poder seleccionar la manera más eficiente para mover la información hacia y desde el dispositivo móvil.

De los métodos disponibles, algunos pueden ser más apropiados que otros para la transferencia de información, dependiendo del tamaño de los datos que se quieran transferir, la validez en el tiempo de los mismos y el valor que tengas los cambios para el usuario. Por ejemplo supóngase una aplicación de soporte a vendedores, el cambio en el stock disponible de un determinado producto es una pieza de información pequeña que puede ser transmitida instantáneamente y que tiene valor si el usuario la recibe en tiempo. Lo ideal para este caso sería la transmisión a través de una red inalámbrica. Otro ejemplo es el de una aplicación que permite en un mapa de las cercanías del usuario ubicar un servicio determinado como ser un banco. El tamaño de la información del plano es grande y el dato de la dirección del banco es pequeño pero debe ser enviado rápidamente ya que si el usuario se encuentra en movimiento, un retraso puede significar que la misma ya no tenga validez. Entonces, se podría transferir al dispositivo móvil, el plano por algún método de alta velocidad (p. ej. 2 ó 6) antes de que el usuario salga de su casa u oficina, y el dato de la dirección, a través de una conexión de red inalámbrica (p. ej. 5)

3.4.2 Acceso a datos críticos

Si importar cuantos métodos de acceso a datos se utilicen, habrá momentos en los que no estarán disponibles. Los usuarios podrán estar fueran del área de cobertura de la red inalámbrica o de la red celular y sin posibilidad de utilizar un medio de conexión.

Una buena regla para estos casos es la de asegurar que todos los datos “críticos” sean almacenados en el dispositivo móvil y viajen con el usuario. Por

datos críticos se entiende aquella información o herramienta, que en ausencia de los mismos el usuario no puede continuar su actividad. Generalmente es mas importante que esta información sea portable, a que este accesible en tiempo real. Por ejemplo, un técnico de campo reparando un equipo ubicado en el subsuelo de una edificio de oficinas que intenta acceder en forma online al manual de reparación que reside en un servidor Web. Si en ese lugar no existe cobertura de la red inalámbrica, para realizar la consulta deberá subir hasta donde la haya y luego retornar a su trabajo. Una alternativa seria descargar la parte pertinente del manual en el dispositivo móvil del técnico antes de que concurra a realizar el trabajo. El dispositivo móvil debe incorporar un almacenamiento de datos y disponer de un poder de procesamiento tal que le permita al usuario acceder a la información donde quiera que se encuentre sin interrumpir su actividad. Una vez que la conexión de red este disponible los recursos locales podrán acceder al lado Server de la aplicación y actualizar la información que sea necesaria.

El ejemplo anterior plantea un escenario donde la información siempre fluye hacia el dispositivo móvil. Esto si bien es valido tampoco se debe descartar el caso donde la información debe ser transferida en ambos sentidos, como ser el conocido caso de la aplicación para soporte de vendedores. Esto deriva, entre otras cosas, en la necesidad de disponer de un proceso de sincronización de la información local con la información del repositorio central. Es un punto ampliamente estudiado en la comunidad móvil que ha derivado en proyectos impulsados por varios proveedores tratando de llegar un Standard para la sincronización de datos. Para mas detalles ver en Glosario “SyncML”

3.4.3 Seguridad

La seguridad es una de las mayores preocupaciones tanto para entornos móviles como no móviles.

Una solución de seguridad para aplicaciones móviles debe tratar todo el trayecto móvil, es decir desde los sistemas back-end hasta el dispositivo del cliente. Bajo este concepto puede decirse que la seguridad en una aplicación móvil tiene tres preocupaciones principales [Arc01]:

- ***Privacidad de la mensajería.*** Asegurar que los datos sensibles no estén comprometidos durante su transmisión. Esto comúnmente es realizado con el uso de criptografía.
- ***Autentificación.*** Asegurar que la identidad de todos los usuarios involucrados en la comunicación es correcta. Usualmente alcanzado con el uso de certificados digitales o técnicas de usuario / password
- ***Seguridad en el dispositivo.*** Asegurar la protección de los datos almacenados en el dispositivo móvil, usualmente logrado a través del uso de password de protección y encriptación.

Según la arquitectura de la aplicación móvil, se dispone de distintos mecanismos de seguridad ya provistos por sistemas comerciales, a los cuales se puede adherir.

Por otra parte, sistemas operativos como Palm OS ofrecen una serie de servicios que pueden ser incorporados a las aplicaciones y políticas de seguridad. A modo de ejemplo se listan algunos de ellos:

- ***System-level authorization and authentication.*** Permite especificar una serie de reglas para permitir el acceso de datos

almacenados en el dispositivo. Estas incluyen el uso de passwords, PINs, y una variedad de opciones biométricas.

- ***System-wide encryption.*** Permite la utilización de algoritmos de encriptación actualmente considerados más seguros como RC4 y SHA-1.
- ***Secure Sockets Layer (SSL) Encryption.*** Incorpora SSL 3.0 para proveer protección "*end-to-end*" en as comunicaciones basadas en Internet.
- ***Support for Signed Code.*** Utiliza un esquema de firma digital para permitir el accedo a datos o recursos importantes.
- ***Unique Device Identification.*** Permite autenticar dispositivos a través de sus Flash ID, MAN (*Mobile Access Number*) y ESN (*Electronic Serial Number*)

3.5 REQUERIMIENTOS DE CONFIABILIDAD

Como ya se dijo, en algunos aspectos referidos a confiabilidad, los dispositivos móviles se asemejan más a servidores que a computadoras de escritorio. A continuación se analizaran algunos de ellos.

3.5.1 Always Running

Los dispositivos móviles y sus aplicaciones, comúnmente se encuentran activos 24 horas al día, 7 días a la semana. Teléfonos celulares y PDAs son un ejemplo de esto, los mismos siempre se encuentran activos o en algún modo *standby* que les asegura estar disponibles casi instantáneamente cuando se los requiera. A pesar de que se ha incrementado el hecho de dejar las computadoras de escritorio encendidas por mas tiempo, es normal que los usuarios reinicien las mismas, inicien y cierren sesiones, abran y cierren aplicaciones en reiteradas oportunidades en la misma sesión. En contraste las aplicaciones móviles con el objetivo de mantener un acceso instantáneo, se encuentran corriendo en *background*.

Por estas razones, los dispositivos móviles se asemejan a los servidores en el hecho que los mismos necesitan estar listo para proveer un servicio instantáneo a sus clientes.

3.5.2 Manejo de fallas

Las aplicaciones corriendo en un dispositivo móvil, operan en un entorno altamente demandante, donde son comunes las interrupciones en la comunicación, la disminución del ancho de banda, la capacidad limitada de las batería, entre otros. El mismo sistema operativo puede detener una aplicación corriendo en *background* si los recursos comienzan a disminuir. Y peor aun, el dispositivo puede ser extraviado, robado o sufrir cualquier tipo de daño. Ante cualquiera de estas circunstancias, el usuario necesita tener la seguridad que la integridad de los datos residentes en el dispositivo móvil no se verá afectada.

Por estas razones, las aplicaciones móviles, como los servidores de misión crítica, deben asegurar que los datos importantes para el usuario y su administración puedan superar la ocurrencia de daños inesperados o fallas.

3.5.3 Uso de la memoria

Una computadora de escritorio tiene comúnmente asignada un gran espacio en disco que le permita bajar allí memoria que no esta siendo usada en forma de archivo. Este archivo es comúnmente llamado Archivo de Intercambio (*swap file*). Si nueva memoria es requerida por una aplicación porque se esta iniciando o porque tiene la necesidad de memoria adicional, y el sistema comienza a quedarse sin memoria física, el sistema operativo tomara secciones de memoria que no están siendo usadas y las enviara al archivo de intercambio. De esta manera una computadora de escritorio podrá funcionar como si tuviera mucha mas cantidad de memoria RAM que la que realmente tiene. Esto permite a los usuarios correr varias aplicaciones simultáneamente. En los servidores se tiende a no usar esta estrategia porque los mismos fueron diseñados para un máximo rendimiento. En un servidor se quiere mantener todo en la memoria física donde el acceso es mucho más veloz. En los dispositivos móviles comúnmente no se cuenta con grandes discos como para alocar un Archivo de Intercambio. Disponer de esta capacidad en un dispositivo móvil puede ser prohibitivo por costos, tamaño, velocidad y consumo de energía. Nuevamente, como los servidores, los sistemas operativos y las aplicaciones de un dispositivo móvil, comúnmente no utilizan la técnica de archivos de intercambio.

3.5.4 Servicios Críticos

Varios dispositivos móviles sirven a otro propósito crítico mientras corren aplicaciones prioritarias. Si un teléfono móvil deja de funcionar como teléfono móvil porque una aplicación colapso, se vuelve drásticamente lento en su respuesta, bloquea la interfase del usuario, o presenta cualquier otro mal comportamiento, el usuario final no se encontrará muy conforme. La mayoría de los sistemas operativos móviles posee niveles de protección para salvaguarda las funciones críticas, pero de todas formas una aplicación móvil debe ser capaz de hacer que el dispositivo siga siendo útil bajo cualquier circunstancia. Como los servidores, los dispositivos móviles necesitan manejar una serie de servicios críticos que deben estar disponibles para el usuario en todo momento.

3.6 RESUMEN CAPITULO

En el presente capitulo se presentaron características propias de las aplicaciones móviles que de una forma u otra afectaran las decisiones de diseño de las mismas. El enfoque del capitulo trato de ser practico, con el objetivo de presentar una serie de consideraciones a tener en cuenta cuando se trata con aplicaciones dentro de un entorno móvil. Se hizo hincapié en las diferencias en el uso de una aplicación móvil a una aplicación de escritorio. Se trataron cuestiones como seguridad, el foco en el objetivo y requerimientos de confiabilidad.

CAPITULO 4

PATRONES EN APLICACIONES MÓVILES

Los patrones de diseño han sido propuestos en varios dominios como una forma de capturar y compartir conocimiento de diseño entre los diferentes participantes del proceso de desarrollo. Los patrones comunican en una forma compacta el problema recurrente, capturando su esencia y la de su solución. Según los distintos autores el formato del patrón puede varias pero comúnmente, describen el problema en profundidad, el racional para la solución, como aplicar la solución, y algunos detalles a tener en cuenta al momento de aplicar dicha solución. Un conjunto de patrones interrelacionados para un dominio específico es conocido como un lenguaje de patrones.

4.1 INTRODUCCIÓN

El lenguaje de patrones tiene sus orígenes en el campo de la arquitectura con el trabajo de Christopher Alexander (ver Anexo 1), *A Patterns Lenguaje: Towns, Building, Construction*, quien propuso un conjunto de 253 patrones, no solo como una herramienta para los arquitectos, sino como una manera en que los *stackholders* del proceso de diseño, incluido el cliente, se comunicaran entre si en referencia al mismo. La idea fue replicada al mundo informático y tuvo su punto de mayor difusión con el conocido trabajo de GoF (ver Anexo 1), *Design Patterns: Elements of Reusable Object-Oriented Software* [GHJV94].

A partir de allí, y dado el éxito obtenido, un gran cantidad de investigaciones han llevado el concepto de patrones a los diferentes dominios del desarrollo de software.

Para el caso de dominios nuevos, como el de las aplicaciones móviles, el hecho de disponer de un lenguaje de patrones, tiene una serie de beneficios potenciales, por ejemplo se tendrá una forma “estándar” de documentar las lecciones aprendidas en los primeros diseños. Sin embargo, puesto que los patrones capturan experiencia, algunos importantes patrones para un nuevo dominio, no estarán allí para soportar el desarrollo del primer sistema. Solo los patrones “importantes” pueden trascender de dominio en dominio y entonces colaborar para el desarrollo del primer sistema. Los nuevos patrones solo saldrán a la luz luego de adquirir experiencia en el dominio.

4.2 TRABAJOS RELACIONADOS

A la fecha existen varios trabajos relacionados al tema de patrones entorno a las aplicaciones móviles, resulta interesante analizar algunos de ellos con el objetivo de profundizar en el tema.

Los trabajos que se presentaran son los siguientes:

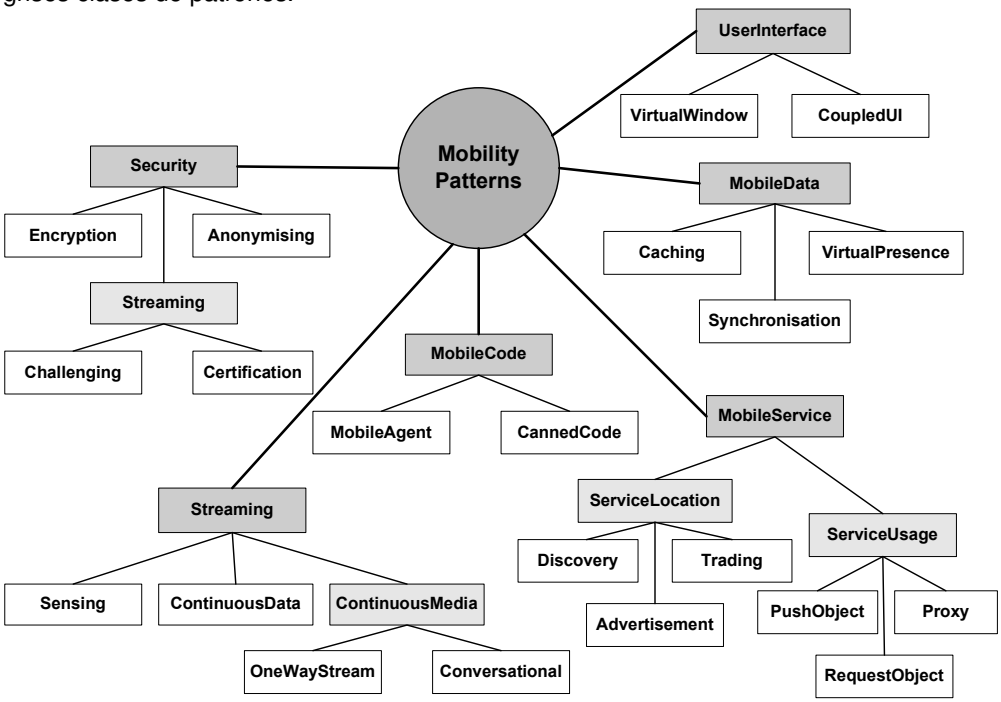
- *Patterns of Mobile Interaction*. [Rot01] Jörg Roth, University of Hagen, Alemania
- *Design Patterns for Ubiquitous Computing*. [LB03] Group for User Interface Research, Computer Science Division. University of California y DUB Group, Computer Science and Engineering, University of Washington
- *An architectural pattern catalogue for mobile web information systems,* [RR04] Walter A. Risi y Gustavo Rossi, LIFIA, Universidad Nacional de La Plata, Argentina

4.2.1 Patterns of Mobile Interaction

Este trabajo esta enfocado teniendo en cuenta las condiciones ampliamente restrictivas de los ambientes móviles, especialmente en aquellas relacionadas al hardware. El autor sostiene que al momento del diseño se deben considerar diversas áreas de problemas y comúnmente cuando se hace foco en un área específica se descuida otra, especialmente en un campo tan complejo como el de la computación móvil, descuidar un área puede llegar a significar el fracaso de todo el diseño. Ante esto propone una posible solución, la cual ha sido usada exitosamente en otros dominios, estos son los Patrones de Diseño.

La propuesta es utilizar un tipo especial de patrones que el autor pasa a denominar *Mobility Patterns*.

Los *Mobility Patterns* cubren determinadas áreas de problemas muy comunes en escenarios móviles. Patrones relacionados son agrupados en *Pattern Class*, usando una jerarquía (ver figura siguiente). Las cajas blancas representan patrones, mientras que las grises clases de patrones.



Con el objeto de describir los Patrones el autor estableció un formato que contiene las siguientes secciones: *Name, Synopsis, Context, Forces, Solution, Concequences, Examples, Related Patterns* y *Classes*. La sección *Implementation* del modelo de patrones utilizado en el desarrollo orientado a objetos, fue reemplazada por *Examples*. Explica que es dificultoso dar una implementación tangible para un específico *Mobility Pattern*, y resulta más significativo presentar una lista de buenos ejemplos de uso.

En adición se incorpora la sección *Classes*, la que indica como el patrón es integrado dentro de la jerarquía. La clase del patrón da al diseñador información adicional en cuanto estructura, de manera que los problemas y soluciones relacionadas con un patrón específico puedan ser clasificados más fácilmente.

Comparado con las descripciones formales de patrones de diseño, los *Mobility Patterns* presentan una característica sumamente informal.

Se presentan dos ejemplos: *Synchronisation Pattern* y *Proxy Pattern*

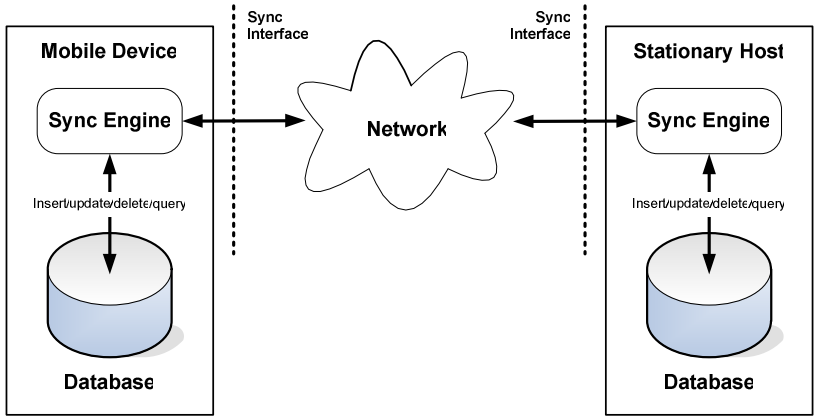
Synchronisation Pattern

Synopsis: Información idéntica es almacenada en diferentes dispositivos o computadoras los cuales se encuentran débilmente conectados (entiéndase dispositivos con conexión intermitente o nula). Diferentes usuarios aplican cambios a la información a menudo simultáneamente.

Context: Dos usuarios con un copia en sus PDAs, de la información originalmente almacenada en un Server. Mientras se encuentran en movimiento se conectan en forma esporádica a dicho Server.

Forces: Información idéntica almacenada en diferentes locaciones tiende a perder el sincronismo cuando el usuario aplica cambios a su copia local y los dispositivos se conectan esporádicamente.

Solution: Cada dispositivo o computadora es provisto de un *Sync Engine*. Ver la siguiente figura



Cada *Sync Engine*

- Mantiene un seguimiento de las modificaciones que se aplican a la información local.
- Intercambia modificaciones con otros dispositivos cada vez que se reconectan
- Detecta conflicto e implementa una estrategia de resolución

Concequences: Este patrón puede ser utilizado cuando la información almacenada en diferentes dispositivos no tiene que estar fuertemente actualizada. Los usuarios deben tener presente que la información puede ser modificada simultáneamente por otros usuarios y luego pueden presentarse conflictos.

Para poder comparar efectivamente los cambios, la información debe ser almacenada en tablas o listas. Idealmente, las modificaciones son “logueadas” para cada campo en un registro. La información poco estructurada debe ser comparada registro por registro lo que incrementa la probabilidad de conflictos no deseados. Este patrón no es aplicable a información de video o audio.

Examples: SyncML (http://www.syncml.org) es un Framework para sincronización de datos en escenarios móviles. SyncML fue recientemente consolidado al proyecto OMA (*Open Mobile Alliance*), el cual es impulsado Ericsson, IBM, Lotus, Motorola, Nokia, entre otros.

Palms HotSync permite a los usuarios sincronizar su Palm con una o más PCs.
QuickStep plataform, la cual será brevemente explicada mas adelante.

Related Patterns: VirtualPresence

Classes: MobileData

Remote Proxy Pattern

Synopsis: Un dispositivo no tiene la capacidad de realizar una determinada tarea. Se conecta a otro dispositivo de mayor poder computacional, el cual actúa como delegado.

Context: Considérese un usuario navegando en la Web con un dispositivo HandHelp. La resolución de la pantalla del mismo es muy pobre, por lo que los elementos gráficos y las tablas son difíciles de mostrar. Adicionalmente el *rendering* de elementos complejos requiere de un poder computacional que no dispone el dispositivo.

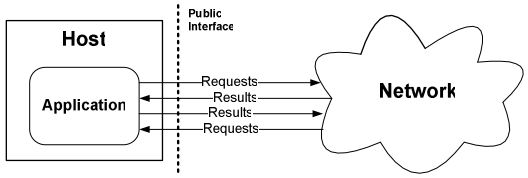
Forces: el usuario que requiere una tarea específica,

- Necesita minimizar el uso de ancho de banda
- Necesita minimizar el uso de los recursos computacionales
- Espera un apropiado comportamiento de Input/Output, acorde a las capacidades disponibles localmente.

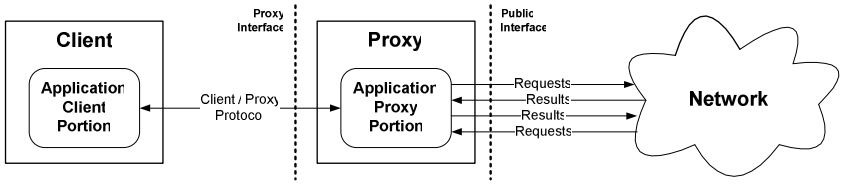
Solution: el dispositivo no se conecta directamente a la red, pero pregunta por otros dispositivos o computadoras para realizar esta tarea. Esta otra computadora, llamada *Proxy,*

- Acepta requerimientos de servicios de otros dispositivos
- Se conecta al actual *Service Provider* y realiza la tarea requerida
- Procesa el resultado
- Envía la respuesta al dispositivo inicial

La siguiente figura muestra la comunicación tradicional entre una computadora y la red.



Esta otra figura representa la configuración acorde al *Remote Proxy Pattern*



La aplicación se encuentra esparcida sobre dos dispositivos llamados Cliente y Proxy.

Se identifican las siguientes entidades:

- El cliente es el dispositivo del usuario final, el que contiene al menos la interfase del usuario de la aplicación.
- El Proxy corre las partes más demandantes de la aplicación, por ejemplo aquellas que requieren mayor poder computacional. Usualmente el Proxy es un *host* estacionario que corre sin interacción con el usuario. La interfase del usuario es solo necesaria para configuración o administración. La relación entre el Cliente y el Proxy cambia dinámicamente.
- La interfase pública establece la comunicación entre el Proxy y la red. Dicha interfase es idéntica a la presentada en una conexión tradicional, de hecho desde el punto de vista de la red no es posible distinguir una arquitectura de Proxy de una correspondiente a un simple dispositivo.
- El cliente accede al Proxy a través de la interfase del Proxy. Esta interfase no tiene que ser publica para que el resto de los *host* de la red no puedan accederla. Usualmente el dispositivo del usuario es móvil y el Proxy es estacionario y la comunicación se establece vía inalámbrica.

Consequences: Este patrón es en general muy útil en escenarios móviles. Es muy común que los dispositivos de los usuarios tengan pobres capacidades y además quieran realizar tareas muy demandantes.

Como gran beneficio es que no se tenga que cambiar la interfase publica, el resto de la red permanece sin cambios y se puede continuar usando la misma infraestructura y protocolos. Sin embargo este patrón tiene dos puntos cruciales:

- El Proxy en si mismo: en caso de falla, la ejecución de tareas no es posible aun estando el dispositivo que requiere y el *Service Provider* corriendo sin inconvenientes.
- El link de comunicación entre el Proxy y el dispositivo: si el link se rompe, la tarea no puede ser realizada, aun en el caso que el Proxy haya ejecutado exitosamente la misma. Mecanismos de cache deben ser integrados en el Cliente para prevenir problemas con conexiones débiles.

Obviamente el Proxy debe contar con mejores capacidades que las del dispositivo del usuario final. Como el Proxy provee un servicio especifico, el dispositivo móvil debe ser capaz de ubicar al Proxy dentro de la red. Esto implica lo siguiente:

- El Proxy debe tener una dirección IP fija o el dispositivo móvil, con la ayuda de algún servicio, resolver la dirección en la red
- El Proxy debe estar on-line, siempre que un dispositivo móvil requiera un servicio
- Actualmente, el Proxy es una tradicional Workstation, no un dispositivo móvil.

Examples: Browse-it permite a los usuarios de handhelp navegar por la Web sin las limitaciones impuestas por el dispositivo móvil, como por ejemplo resolución de la pantalla. El Proxy preprocesa las paginas bajando las escalas de los gráficos. Como resultado la cantidad de información transmitida al dispositivo se ve drásticamente reducida, permitiendo el mismo se encuentre mas liberado para otras operaciones por ejemplo de *rendering*.

Related Patterns: LocalProxy, PushObject, RequestObject

Classes: ServiceUsage, MobileService

La idea de Patrones para el autor, se vio influenciada por sus plataformas de investigación *QuickStep* y *Pocket DreamTeam*. Ambas permiten a los usuario compartir información en grupos, y hasta editar en forma colaborativa un bosquejo hecho a mano alzada.

QuickStep utiliza el patrón *Synchronisation*. Es una plataforma que brinda soporte al desarrollo de aplicaciones para handhelp del tipo "*movile-aware collaborative*". Ofrece primitivas de comunicación, colaboración y dialogo, que pueden integrarse a una aplicación como "*awareness widgets*" predefinidos.

Pocket DreamTeam esta basado en el patrón *Remote Proxy*. Es la versión para PalmOS de la plataforma desarrollada por el autor, *DreamTeam*. El ambiente DreamTeam permite desarrollar aplicaciones sincrónicas colaborativas, sin preocuparse sobre detalles de red o algoritmos de sincronización. *DreamTeam* fue originalmente desarrollado para PC o Workstation corriendo Java. Esta basado en una arquitectura completamente descentralizada sin la necesidad de un Server central. La información es compartida entre los miembros del grupo por medio de mecanismos automáticos de replicación.

Por ultimo en su trabajo el autor presenta una lista de patrones de los cuales solo da una pequeña descripción y cita ejemplos de aplicaciones que utilizan los mismos. A continuación se transcribe la misma:

Pattern	**Sinopsis**	**Ejemplos** (*)
VirtualPresence	Una pieza de información se encuentra virtualmente en otro dispositivo y puede ser accedida o modificada de acuerdo con un conjunto de reglas del host	Windows CE remote file access, CODA
RequestObject	Un dispositivo requiere un objeto especifico de otro dispositivo	WAP
PushObject	Un dispositivo envía un objeto especifico sin que se lo requieran	SMS. OBEX
LocalProxy	Una instancia local provee una interfase a un servicio local o remoto	Corriendo localmente un Web Proxy
OneWayStream	Transmisión one-way de datos de audio o video	Video on Demand, UMTS
Conversational	Transmisión two-way de datos de audio o video	GSM, DECT, Bluetooth audio
VirtualWindow	Un dispositivo presenta una ventana o desktop a otro dispositivo o computadora	Pebbles, PalmVNC
CannedCode	Un dispositivo envía código el cual se ejecuta en otro dispositivo. El código no es ejecutable en el dispositivo que envía.	WMLscript,
Sensing	Un dispositivo recibe información continua de censado desde otro dispositivo	GPS

Síntesis

El trabajo *Patterns of Mobile Interaction*, esta basado en dos plataformas desarrolladas por el autor, las cuales son fuertemente orientadas a dispositivos tipo HandHelp, lo que se traduce en claras restricciones de hardware. Esto si bien es importante, puede llegar a quitar foco a otros problemas también propios de los ambientes móviles.

Los dos patrones ejemplos se ajustan a las áreas de aplicación planteadas en el capitulo 1, es decir para el caso de *Synchronisation Pattern*, este aplica al área de aplicaciones de bases de datos compartidas, mientras que el *Remote Proxy Pattern* lo hace para el área de portales.

La idea de organizar los patrones en clases, dentro de una estructura, sin bien no es nueva permite tener una visión global y una organización que facilitan la compresión y el uso de los mismos.

De la lista de patrones que el autor presenta, no se ha podido encontrar más información que permita comprender los patrones que no fueron descriptos en detalle.

4.2.2 Design Patterns for Ubiquitous Computing

El trabajo realizado por estos grupos de investigación, extiende la idea del uso de patrones de diseño para asistir a los diseñadores, al campo de *Ubiquitous computing* (ver anexo 1). Al ser esta un área en surgimiento, plantean que desarrollar patrones en esta etapa, traerá beneficios como:

- Acelerar la difusión de nuevas técnicas de interacción y evaluación de resultados, al presentarlas en una forma mas practica para los diseñadores.
- Un lenguaje de patrones puede ayudar a integrar ideas y ver mas claramente cuales son los puntos que restan atacarse.
- Puede influenciar positivamente en el diseño de aplicaciones ayudando a encontrar buenas soluciones y evitando los pobres diseños.

Con este objetivo desarrollan un lenguaje inicial de patrones para *Ubiquitous computing* integrado por 45 pre-patrones, agrupados por categorías que tratan temas como espacios físico-virtuales, técnicas de interacción, administración de la privacidad, entre otros.

Los denominan pre-patrones, argumentando que recién comienzan a surgir y aun no son usados por la comunidad de diseñadores, ni por los usuarios. Sin embargo adoptan el formato de patrón para valerse de la característica de los mismos para comunicar conocimiento de diseño. Esperan que estos pre-patrones evolucionen con el tiempo donde algunos serian remplazados por nuevos patrones.

Realizaron un estudio controlado sobre como influenciaban estos pre-patrones con un grupo de diseñadores. De manera resumida, el estudio se dividió en dos rondas, en la primera se contó con nueve pares de diseñadores a los que se le dio el objetivo de diseñar un aplicación "*location-enhanced*", cuatro tuvieron acceso a los pre-patrones y cinco no. En la segunda ronda, se modificaron los pre-patrones, en función del feedback obtenido en la primera ronda, y nuevamente se dividió el grupo de diseñadores en seis pares que los usaron y un para que no. Luego se analizaron los resultados en función de una serie de características prefijadas para los diseños, como creatividad, calidad y completitud. En la evolución de la primera ronda no hubo estadísticamente diferencias significativas en términos de calidad, completitud o creatividad, respecto del grupo que tuvo acceso a los pre-patrones, del grupo que no lo tuvo. En la segunda ronda, ya hubo

algunas diferencias como por ejemplo en términos de rapidez en completar la tarea, entre el grupo que los uso y el grupo que no los uso. Más allá de eso, el estudio confirmo la idea original de la utilidad que podía tener el uso de patrones, colaborando en la transmisión de ideas, ayudando a aprender sobre el nuevo dominio y evitando problemas de diseño que ya fueron detectados anteriormente.

La forma en que desarrollaron el lenguaje de patrones fue a través de un proceso interactivo de varios meses de duración. Comenzaron con una *brainstroming* de los patrones candidatos basados en una revisión de la literatura existente. Inicialmente trataron de extraer patrones de alto nivel, aquellos que de forma los mas abstracta posible, abarcaran por ejemplo un grupo de aplicaciones. Resulto en una tarea altamente compleja, es decir la de encontrar los patrones y al mismo tiempo crear una jerarquía. Esto los llevo a trabajar en identificar primero los patrones de bajo y medio nivel y luego encontrar los temas que conectaban unos con otros. Comenzaron con un conjunto de 80 pre-patrones que luego del estudio descrito en el párrafo anterior, fueron depurados a 45 pre-patrones. Finalmente los catalogaron en 4 grupos, a saber:

- *Ubiquitous Computing Genres* – describen ampliamente clases de aplicaciones de computación ubicua.
- *Phisical-Virtual Spaces* – trata sobre como objetos y espacios pueden ser combinados con lo virtual
- *Developing Successful Privacy* – describe políticas y mecanismos para administrar la privacidad del usuario final
- *Designing Fluid Interactions* – detalles de técnicas de interacción con sensores y dispositivos.

La siguiente tabla muestra los diferentes pre-patrones y su clasificación

A – Ubiquitous Computing Genres	**B – Physical-Virtual Spaces**	**C – Developing Successful Privacy**	**D – Designing Fuild Interactions**
Describe amplias clases de aplicaciones emergentes, proveyendo varios ejemplos e ideas.	Asocian objetos físicos y espacios, con información, por ejemplo servicios basados en ubicación.	Políticas, sistemas, y cuestiones de interacción en diseños de sistemas sensibles en la privacidad	Como diseñar interacciones que involucran varios sensores y dispositivos.
A1 - Upfront Value Proposition	B1 - Active Map	C1 - Fair Information Practices	D1 - Scale of Interaction
A2 - Personal Ubiquitous Computing	B2 - Topical Information	C2 - Respecting Social Organizations	D2 - Sensemaking of Services and Devices
A3 - Ubiquitous Computing for Groups	B3 - Successful Experience Capture	C3 - Building Trust and Credibility	D3 - Streamlining Repetitive Tasks
A4 - Ubiquitous Computing for Places	B4 - User-Created Content	C4 - Reasonable Level of Control	D4 - Keeping Users in Control
A5 - Guides for Exploration and Navigation	B5 - Find a Place	C5 - Appropriate Privacy Feedback	D5 - Serendipity in Exploration
A6 - Enhanced Emergency Response	B6 - Find a Friend	C6 - Privacy-Sensitive Architectures	D6 - Context-Sensitive I/O
A7 - Personal Memory Aids	B7 - Notifier	C7 - Partial Identification	D7 - Active Teaching
A8 - Smart Homes		C8 - Physical Privacy Zones	D8 - Resolving Ambiguity
A9 - Enhanced Educational		C9 - Blurred Personal Data	D9 - Ambient Displays

Experiences			
A10 - Augmented Reality Games		C10 - Limited Access to Personal Data	D10 - Follow-me Displays
A11 - Streamlining Business Operations		C11 - Invisible Mode	D11 - Pick and Drop
A12 - Enabling Mobile Comerse		C12 - Limited Data Retention	
		C13 - Notification on Access of Personal Data	
		C14 - Privacy Mirrors	
		C15 - Keeping Personal Data on Personal Devices	

Los pre-patrones adoptan un formato similar a los patrones de Alexander, donde cada uno de ellos esta compuesto por las siguientes secciones:

- ***Name*** - un nombre y par letra-numero donde la letra indica a que grupo pertenece el patrón. Por ejemplo el A10 corresponde al patrón numero diez, del grupo A – *Ubiquitous Computing Genres*.
- ***Background*** – provee el contexto y el alcance del patrón, y además describe cualquier otro patrón que este relacionado con este.
- ***Problem*** – describe el problema que el patrón ataca.
- ***Solution*** – la solución al problema planteado en la sección anterior, también hace referencia a cualquier otro patrón de bajo nivel que pueda ayudar en la solución del problema.
- ***References*** – referencias a trabajos relacionados con el patrón.

Según los autores, sus patrones tienden a ser más prescriptitos que descriptivos, esto se debe principalmente a que existen pocas aplicaciones de *Ubiquitous Computing* en práctica. Tratan de poner foco en asuntos de alto nivel, como necesidades del usuario, versus especificaciones de interfaces o técnicas de interacción. Argumentan que las abstracciones de alto nivel son mucho más fáciles de comprender e implementar que las de bajo nivel.

A continuación se presentan cuatro patrones ejemplo, uno de cada uno de los grupos. El catalogo completo de estos pre-patrones puede ser ubicado en: http://guir.berkeley.edu/projects/patterns

A5 – Guides for Exploration and Navigation

Background: Este patrón describe guías de viaje portátiles, las que pueden ayudar, a las personas, a explorar lugares que no le son familiares. Este patrón es un ejemplo de A2 *Personal Ubiquitous Computing* y puede ser usado junto con *A1 Up-Front Value Proposition* para crear guías más útiles.

Problem: A las personas que exploran nuevas áreas geográficas siempre se le presentan las mismas cuestiones, ¿Donde estoy ahora?, ¿Donde están mis amigos?, ¿Que hay de interesante en el vecindario?,

¿Como llego hasta allí?, ¿Cual es el horario de los medios de transporte?

Solution: Guías para exploración y navegación son típicamente construidas usando dispositivos móviles como PDAs y teléfonos celulares. A continuación se presentan una serie de características que los usuarios encuentran útiles en una guía de viaje:

- *Proveer un mapa activo del área* – Los mapas activos (*B1 Active Maps*) son pantallas de actualización dinámica que muestran donde el usuario se encuentra actualmente, y en forma opcional, que amigos y puntos de información ordenada por tópicos (*B2 Topical Information*) se encuentran en las cercanías.
- *Considerar búsquedas por lugares y por amigos* – Con el surgimiento de las capacidades basadas en ubicación, las guías pueden ofrecer el servicio de ubicación se sitios (*B5 Find a Place*), permitiendo a los usuarios búsquedas por restaurantes cercanos, hoteles, estaciones de servicio, o atracciones turísticas. Una variación es la de ubicar un amigo (*B6 Find a Friend*), la cual permite realizar búsquedas de amigos en la cercanía.
- *Permitir a las personas explorar* – Las personas tiene aversión a los tours preplaneados. Una alternativa es la de proveer exploración casual (*D5 Serendipity in Exploration*) permitiendo a los usuarios una exploración mas amigable. Por ejemplo, en vez de proveer un mapa con detalles, dejar a las personas descubrir por sus propios medios.
- *Considerar permitir a los usuarios crear contenido* – En vez de solo consumidores de contenido, considerar la posibilidad de soportar a usuarios que crean contenido (*B4 User-Created Content*). Por ejemplo una idea es la de permitir a los usuarios asociar a lugares “post-it” virtuales para que puedan ser recuperados en otro momento por ellos mismos o por otros usuarios.
- *Considerar Servicios de Traducción* – Las personas que visitan países extranjeros, pueden no estar familiarizados con el lenguaje local. Una idea es la proveer traducciones de señales las cuales son previamente fotografiadas. Otra idea es la de proveer frases básicas utilizadas en situaciones comunes.

References:

Pfeiffer, Eric, “WhereWare.” Technology Review. 106(7): 46–52.
Benefon Esc! NT2002 Personal Navigation Phone. http://www.benefon.com/products/esc/
Kaasinen, E. (2003). “User needs for location-aware mobile services.” Personal Ubiquitous Computing 7: 70–79.
Abowd, G. D., C. G. Atkeson, et al. (1997). “Cyberguide: A Mobile Context-Aware Tour Guide.” Baltzer/ACM Wireless Networks (3): 421–433.
Interactive Systems Laboratories, Carnegie Mellon University, Sign Translation. http://www.is.cs.cmu.edu/signtranslation/

B4 – User-Create Content

Background: Este patrón trata sobre el contenido que la gente puede crear y proveer como un servicio para otros usuarios. Este patrón puede ser incorporado en *A5 Guides for Exploration and Navigation*, *B3*

Successful Experience Capture, *A10 Augmented Reality Games* y *D5 Srendipity in Exploration.*

Problem: La gente no siempre quiere ser solamente consumidores de contenido.

Solution: Permitir a los usuarios crear contenido que otros puedan ver. Una idea es la de permitir a las personas dejar notas virtuales asociadas a lugares físicos, dejando mensajes a otras personas que visiten el mismo lugar. Usuarios creadores de contenido son una parte integral de varios juegos de realidad aumentada (*A10 Augmented Reality Games*). Recientemente la gente ha comenzado a crear moblogs (*mobile web logs*). Esta forma de capturar experiencia (*B3 Successful Experience Capture*), apunta a hacer más sencillo el hecho de tomar una fotografía digital y escribir en ella sus pensamientos mientras se encuentra en movimiento.
Un peligro potencial es la generación de contenido impreciso, inapropiado o hasta calumnioso hacia algunos individuos u organizaciones. Una manera de administrar este tema es el hecho de facilitar los medios para que los usuarios puedan reportar este tipo de contenido.

References:
Abowd, G. D., C. G. Atkeson, et al. (1997). "Cyberguide: A Mobile Context-Aware Tour Guide." Baltzer/ACM Wireless Networks 3: 421-433.
Jenna Burrell, G. K. G., Kiyo Kubo, Nick Farina (2002). Context-Aware Computing: A Test Case. Proc. 4th International Conference, Goteborg, Sweden, Springer.
Salil Pradhan, C. B., Jun-Hong Cui, Alan McReynolds, Mark Smith (2001). Websign: hyperlinks from a physical location to the web. Palo Alto, HP Laboratories.

C1 – Fair Information Practices

Background: No siempre esta claro como una compañía o gran organización maneja la información personal de sus usuarios. Este patrón en conjunción con *C2 Respecting Social Organizations* y *C3 Building Trust and Credibility*, forman el núcleo de las políticas para privacidad en los sistemas de computación ubicua.

Problem: La privacidad es una seria preocupación para muchas personas, especialmente dada la naturaleza sensible de los datos colectados por un sistema de computación ubicua. Sin embargo no siempre queda claro que políticas o procedimientos se utilizan para colectar o administrar la información personal de manera de hacerlo en forma segura.

Solution: La privacidad es la crítica más citada para la computación ubicua, principalmente porque tales sistemas tienden a colectar datos personales sensitivos como por ejemplo la ubicación y actividad del usuario. Hay una variedad de asuntos complejos que necesitan ser manejados, tales como que información puede ser colectada, así como de quien y para quien. Además las leyes sobre privacidad y telecomunicaciones varían de país en país.
Este patrón trata de mostrar que es lo requerido en materia de privacidad.

Las *Fair Information Practices*, son un *framework* generalmente acordado por grandes organizaciones que manejan información personal de individuos. Las prácticas fueron creadas a principios de los ’70 y son las bases de varias leyes actuales sobre privacidad. A continuación se listan las mismas:

1. **Notice.** Las organizaciones deben notificar a los individuos sobre los propósitos por los cuales se colecta y usa información referida a los mismos. Esto incluye información de contactos por investigación, o querellas, así como información personal de terceras partes que es compartida.
2. **Choice.** Las organizaciones deben dar a los individuos la oportunidad de elegir que información personal puede ser divulgada a terceras partes o utilizada para otro propósito distinto para el que fue colectada.
3. **Onward transfer.** Las organizaciones deben notificar y solicitar la selección de la información personal antes de divulgarla a terceras partes.
4. **Access.** Los individuos deben tener acceso a su información personal y estar habilitados a realizar cambios sobre la misma cuando sea inexacta.
5. **Security.** Las organizaciones debe tomar precauciones razonables para proteger la información personal de pérdida, mal uso y accesos no autorizados, divulgación, alteración y destrucción.
6. **Data Integrity.** Las organizaciones deben tomar razonables medidas para asegurar que la información personal es relevante, confiable y correcta para su intención de uso.
7. **Enforcement.** Deben existir, a) procedimientos para verificar que las organizaciones se adhieren a estas practicas, y b) mecanismos independientes para recurrir en casos de quejas realizadas por individuos y disputas, deban ser resueltas

Related Patterns: Las *Fair Information Practices* tiene una visión de la privacidad desde el punto de vista de las organizaciones. No deja de ser importante tener una perspectiva de la privacidad, desde el lado individuos. Una forma de hacer esto es limitando el acceso a la información personal (*C10 Limited Access to Personal Data*), o autorizando a los usuarios a seleccionar que y con quien comparten su información. Otra manera de hacer esto seria utilizar una arquitectura donde la información del usuario se almacene solo en el dispositivo del usuario (*C6 Privacy-Sensitive Architectures*)

References:

Langheinrich, M. *Privacy by Design - Principles of Privacy-Aware Ubiquitous Systems*. In the Proceedings of Ubicomp 2001, pp. 273-291, 2001. http://www.inf.ethz.ch/~langhein/
Location Privacy Protection Act.
The original draft of this act can be downloaded at
http://www.techlawjournal.com/cong107/privacy/location/s1164is.asp

D6 – Context-Sensitive I/O

Background: Este patrón considera como las modalidades de entrada y salida pueden ser adaptadas en función del contexto del usuario. Este patrón se aplica a todos los niveles de la escala de interacción (*D1 Scale*

of Interaction) y es una de las ideas potenciales para el descubrimiento de servicios y dispositivos (*D2 Sensemaking of Services and Devices*)

Problem: Los dispositivos son usados en una variedad de ubicaciones y situaciones, desde ambientes silenciosos a ruidosos, en forma individual o grupal, estacionario en un lugar o en movimiento. ¿Cómo pueden ser diseñados los dispositivos tal que su modalidad entrada salida sea la apropiada para cada circunstancia, y no interrumpa a otras personas o usuarios cercanos?

Solution: Las modalidades de entrada salida deben ser adaptadas al contexto corriente del usuario. A continuación se describen algunos escenarios que muestran las posibilidades existentes:

- Un usuario entra a una sala de cine, su teléfono celular se reconfigura a si mismo para solamente mostrar mensajes de texto y vibrar cuando recibe una llamada.
- El dispositivo de un usuario se cambia a una interacción basada en su voz, cuando sus manos están ocupadas, por ejemplo cuando esta manejando un vehiculo.
- La modalidad de salida es restringida a utilizar solamente el display cuando se esta en largas reuniones, con el objeto de minimizar las distracciones.
- Una PDA ajusta automáticamente su backlight para maximizar la visibilidad mientras minimiza el consumo de energía.

Mientras context-sensitive I/O en la teoría es posible, se encuentra aun en investigación, problemas como:

1. Cual es la mejor manera de implementarlo
2. Como hacerlo predecible
3. Como hacerlo y a la vez permitir al usuario revocar una decisión equivocada

Por “predecible”, se refiere a como hacer que el dispositivo se ajuste a las expectativas del usuario. Esto es en parte un problema de aprender de las peculiaridades del sistema, pero aun se encuentra en investigación que clase de feedback es útil como señal para realizar el cambio.

Por “revocar”, la clave es que el usuario mantenga el control (*D4 Keeping User in Control*). Por ejemplo, algunos automóviles bajan automáticamente el volumen del estero cuando esta entrando una llamada al teléfono celular. Esto puede ser lo correcto en la mayoría de las situaciones, pero puede ocurrir que el usuario este escuchando una noticia por demás importante para él. Una manera simple de que el usuario mantenga el control, es haciendo la mayoría de las veces lo correcto automáticamente, pero permitiéndole revocar manualmente las acciones incorrectas, en el caso del ejemplo subir el volumen.

References:

van Duyne, D. K., J. A. Landay, et al. (2002). The Design of Sites: Principles, Processes, and Patterns for Crafting a Customer-Centered Web Experience. Reading, MA, Addison-Wesley.

Síntesis

El trabajo realizado por estos grupos de investigación resulta interesante visto como un compendio de ideas entorno de la problemática asociada a la computación ubicua. El termino pre-patrón se corresponde con lo expuesto, ya que como se puede ver los mismos distan de lo que la comunidad de patrones entiende por patrón. No se presentan diagramas constructivos, ni bloques que indiquen como se conforman las soluciones que cada pre-patrón prescribe, entre otras consideraciones.

De todas maneras puede ser visto como un muy buen resumen para quienes comienzan a avanzar en el terreno de la computación ubicua.

Por otra parte, el estudio llevado a cabo con diseñadores de diferente experiencia, con el fin de determinar en que forma este catalogo ayuda en la tarea de diseñar una aplicación de este tipo, si bien a esta altura de los hechos no esta en discusión el aporte realizado por los patrones al desarrollo de software, resulta interesante ya que no existe en la literatura, muchos de este tipo.

4.2.3 An architectural pattern catalogue for mobile web information systems

En este trabajo sus autores sostienen que los Sistemas Web de Información Móvil (MWIS - *Mobile Web Information Systems*), se han convertido en los sistemas de información comerciales y corporativos en los que se ha puesto mayor empeño, y de allí surgen nuevos desafíos para la Ingeniería de Software. Mientras los problemas mas visibles en la construcción de NWIS son las infraestructuras de comunicaciones y las restricciones de los dispositivos móviles, existen algunos puntos referidos al diseño arquitectónico que deben ser correctamente tratados en un nivel conceptual.

Su enfoque se basa en que las Investigaciones actuales sobre artefactos de software referidas a movilidad, se concentran principalmente en puntos tecnológicos como *wireless networking* y restricciones de hardware. Dejando de lado el hecho de que las NWIS son altamente inestables y deben ser continuamente evolucionadas, y ante esto es ampliamente conocida la importancia de enfatizar consideraciones de diseño de aplicaciones que evolucionan rápidamente, y por lo tanto cierto grado de atención debe ser puesto en el análisis de la arquitectura conceptual cuando se construye una NWIS.

A pesar de los avances tecnológicos, ellos afirman que el diseño de la arquitectura de una NWIS debe ser un desafió mas allá que simplemente las restricciones tecnológicas. Por ejemplo una NWIS puede requerir configurar sus servicios para diferentes tipos de usuarios (móviles o no). Por otra parte, un sistema de información móvil puede necesitar ofrecer servicios a diferentes tipos de usuarios móviles (*wired* o *wireless*).

Para hacer frente a esta necesidad, han definido un catalogo de patrones arquitecturales para este dominio, y han extraído algunos patrones, desde sistemas bien conocidos y actualmente en uso. No se concentran en consideraciones técnicas, sino más bien en identificar las arquitecturas que mejor se ajusten a los objetivos de una aplicación móvil. Sostienen que este catalogo debería ser la base para un lenguaje común de diseño de alto nivel, que describa soluciones arquitecturales para aplicaciones móviles. Y que este lenguaje podría ser usado en el proceso de modelado arquitectónico de este tipo de aplicaciones, con el objeto de resolver problemas en un alto nivel de abstracción, por medio de la reutilización de soluciones exitosas.

Realizan una clasificación de los patrones extraídos, siguiendo un enfoque *bottom-up*. Afirman que la misma no intenta ser definitiva pero sin embargo podría ser vista como un conjunto de ejes organizacionales, para reflejar la naturaleza de los problemas arquitecturales involucrados. A continuación se transcriben las clases propuestas.

Web accessing. Esta categoría trata cuestiones sobre como el acceso físico de una MWIS desde el cliente móvil, depende de las restricciones impuestas por los requerimientos y por el dispositivo. Las limitaciones de hardware tienen un fuerte impacto sobre la manera que el dispositivo móvil accede a la información. Por ejemplo un teléfono móvil tiene un mejor acceso a facilidades, que una PDA standalone (considerándola sin modem). Una PDA con modem dial-up tiene mejor acceso a facilidades que una PDA standalone, pero menos autonomía que una PDA *wireless-enable*. Sin embargo una conexión dial-up se encuentra comúnmente disponible en cada ciudad del mundo, mientras que hoy en día las áreas con redes *wireless* son reducidas. Mientras que algunas alternativas pueden ser preferidas a otras en ciertos contextos, el diseñador de la arquitectura para una NWIS, puede que tenga que proveer una solución que trabaje tanto para clientes con gran autonomía como para cliente muy limitados en términos de acceso a la información. Una razón común para esto es que el dispositivo mas adecuado es también el más caro, y la mayoría de los usuarios poseen dispositivos que se encuentran el medio del espectro de costos, o en ocasiones en la parte inferior.

Web navigation. Esta categoría es dirigida a cuestiones sobre la situación en que el cliente móvil pretende navegar en la Web bajo un contexto de NWIS. Mientras que un cliente desktop tiene un contexto adecuado para la amistosa navegación de la totalidad de la Web, la selección de un cliente móvil es generalmente mas reducida. Un cliente móvil tal vez sea capaz de procesar solo un subconjunto de la totalidad de la Web. Y aun siendo capaz, el usuario móvil puede estar interesado en navegar un subconjunto que le es relevante según su contexto. Notar que esta categoría no intenta atacar el diseño según la perspectiva de la navegación hypermedia, más bien ataca a los asuntos relacionados a la arquitectura para soportar la navegación.

Por ejemplo, la mayoría de los dispositivos PDA tienen Browsers especializados para acceder al contenido de la Web. Sin embargo la mayoría de estos Browsers no trabajan correctamente con cualquier página, solamente lo hacen con un subconjunto, codificadas en una manera tal que son adecuadas para un dispositivo móvil (Por ejemplo pantallas más pequeñas y una limitada sintaxis HTML). Un usuario móvil posiblemente no quiera navegar por la totalidad de la Web en su dispositivo, pero si en un subconjunto adecuado para ese dispositivo y sus necesidades. Considerarse también que el usuario desktop puede sentirse cómodo ingresando el URL o lanzando una búsqueda para encontrar alguna información, pero un urgido usuario móvil puede ver esto como algo no apropiado en su contexto.

La selección de sitios Web disponibles a través de los cuales el usuario móvil puede navegar es generalmente un subconjunto de la totalidad del universo de selecciones, el cual surge no solamente por las limitaciones del hardware, sino también por los intereses especiales del usuario móvil. Un cliente móvil en esas condiciones seguramente no quiere tener que ver con la complejidad de direccionar la totalidad de la Web, pero si a un subconjunto de su interés.

Customisation. Esta categoría incluye a aquellos patrones arquitecturales que tratan los problemas de personalizar/configurar una NWIS. Nuevamente, la naturaleza restrictiva de los dispositivos móviles plantea restricciones físicas y conceptúales, a la información que puede ser presentada en tal dispositivo. Un dispositivo móvil posiblemente no sea capaz de interpretar cada codificación de cada sitio Web. A pesar de esto, el usuario del dispositivo móvil puede no estar interesado en la misma vista de la información que lo esta el usuario de un desktop.

Por ejemplo, mientras que la mayoría de los dispositivos PDA tienen un Browser, las limitaciones de pantalla hacen que algunas páginas sean imposibles

de presentar en estos dispositivos. Algunos Browsers tampoco soportan completamente la sintaxis HTML, y por lo tanto no pueden acceder arbitrariamente a cualquier página en la Web.

Finalmente un usuario móvil accediendo a un sitio Web puede estar interesado solamente en información relevante a su contexto (Por ejemplo un hombre de negocios accediendo al sitio Web de una compañía financiera este más interesado en encontrar información del mercado que la historia de la compañía). Un usuario móvil puede estar interesado en alguna ubicación Web tanto como un usuario no móvil, pero no exactamente en la misma vista de la información. Los dispositivos móviles pueden no ser los suficientemente poderosos como para procesar un sitio Web. Aun siendo así, los usuarios móviles están interesados en una vista de la información la cual es relevante a su contexto móvil.

Los patrones extraídos se presentan con un formato similar al utilizado en el libro de GoF, ligeramente ajustado para considerar la naturaleza de alto nivel que poseen los patrones arquitecturales. Incluyen un diagrama UML simplificado, para explicar la estructura de la solución. En estos diagramas las entidades de primera clase se corresponden a componentes y conectores. Los autores consideran importante notar que conceptualmente estos componentes y conectores arquitecturales no representan necesariamente el mapa exacto de componentes y conectores para una implementación. Mejor dicho las entidades conceptúales muestran aquellos bloques funcionales (componentes) y sus interacciones (conectores) que deben estar presentes para proveer una cierta funcionalidad, pero la implementación puede implicar que algunas entidades sean mapeadas en un simple bloque de implementación o viceversa.

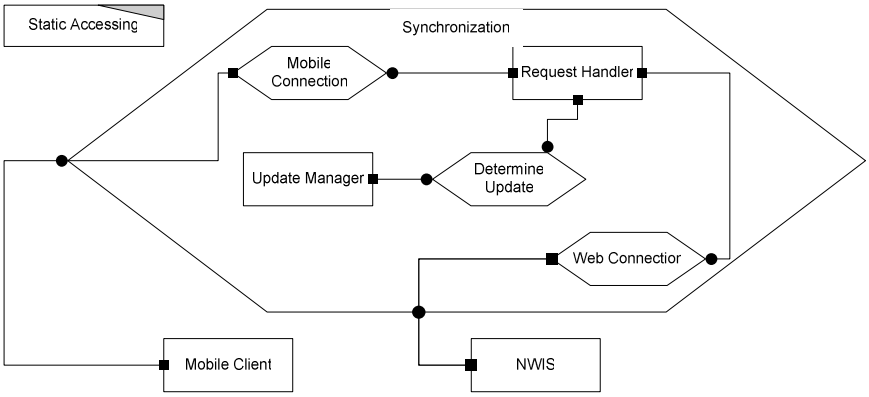
A continuación se resumen los patrones presentados en este trabajo

Web accessing pattern: Static Accessing

Intent. Para proveer al cliente móvil con la posibilidad de acceder a una NWIS, cuando el cliente por si mismo no es capaz de acceder a la Web.

Problem. Tiene que acceder a una NWIS desde un dispositivo con restricciones tales como la falta de capacidad para acceder a la Web por si solo. (Por ejemplo una PDA sin modem o sin capacidad *wireless*). Esto puede pasar en las mas comunes configuración de los dispositivos (por ejemplo la mayoría de los dispositivos Palm y PocketPC no proveen un modem en sus configuraciones básicas). Esto puede ser el caso que la línea de dispositivos objetivos no provea esta funcionalidad.

Solution. Permitir al cliente móvil el acceso al contenido Web a través de un canal de acceso no móvil, ayudado por un dispositivo estático capaz de acceder a la Web.



Rationale. El conector de sincronización es un mecanismo de conexión no móvil que da la ilusión al cliente móvil (posiblemente parcial) de conexión al Web. Dentro del conector sincronización reside el *Request Handler*, el que realiza la mediación entre el cliente móvil y la interfase Web. El motor de decisión de actualización decide si la información en la cliente móvil esta actualizada respecto a la actual NWIS accedida y si no, recupera la misma.

Cada vez que el cliente móvil requiera acceder a la Web, el conector de sincronización recibe el requerimiento, encamina el mismo a través del *Update Manager*, quien decide que información reunir desde la Web, y finalmente físicamente acceder a la Web a través del *Web Connection*.

Consequences. Cualquier dispositivo portátil tendrá la oportunidad de interactuar con una NWIS, mientras que un adecuado conector de sincronización sea construido. La autonomía del cliente es reducida, puesto que el proceso de acceso requiere de la presencia de un mecanismo de sincronización. El hecho de proveer de un adecuado conector de sincronización puede implicar un trabajo adicional para los desarrolladores.

Known uses. El servicio AvantGo utilizado por HotSync de Palm OS Technology para transferir información desde una locación Web particular (AvantGo Channels) al cliente móvil (Palm OS o PocketPC Handhelds) a través de sincronización con un desktop. AcroClip de Casio también utiliza este patrón.

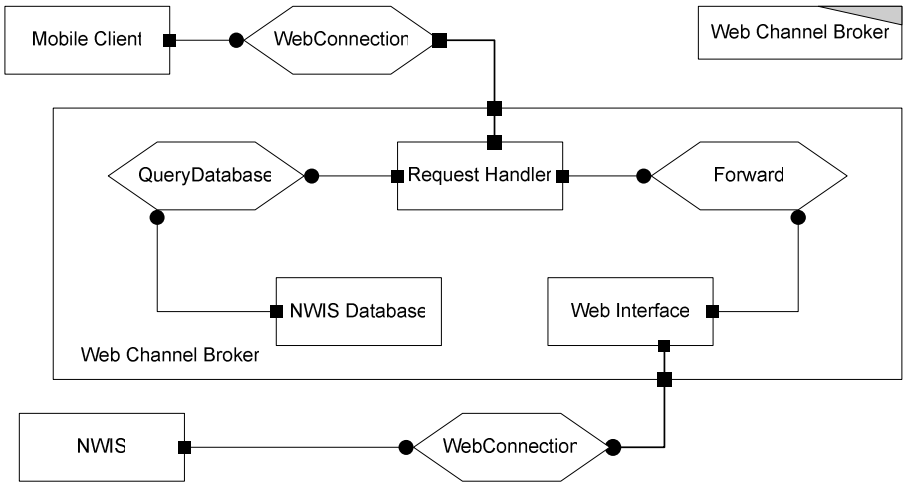
Web navigation pattern: Web Channel Broker

Intent. Simplificar la capacidad de direccionamiento del cliente móvil Web. Crear un espacio de direcciones Web preprocesado por un intermediario.

Problem. Tiene que permitirse al cliente móvil obtener información desde un grupo arbitrario y predefinido de sitios Web. Se quiere mantener el cliente móvil simple (por ejemplo por restricciones del hardware), y por lo tanto se necesita simplificar la forma en que direcciona las páginas de toda la Web. Este puede ser el caso en que se quiere prevenir el acceso del cliente móvil a locaciones arbitrarias en la Web (por ejemplo razones de seguridad corporativa). En este caso se necesita un intermediario con la capacidad de asumir la responsabilidad de direccional toda la Web y presentar un universo transformado al cliente móvil (por ejemplo un conjunto de locaciones corporativas).

Solution. Proveer un broker capaz de interactuar con toda la Web, presentando al cliente móvil una vista transformada (posiblemente reducida) en términos de

Web Channels, donde cada canal es una locación de Web adecuada para el contexto del cliente móvil.



Rationale. El cliente móvil se conecta al *web channel broker* a través del *web connection*. En lugar de acceder a la Web directamente, sus requerimientos son ruteados a través del *web channel broker*. El broker busca en su base de datos interna cual NWIS esta actualmente conectada, y finalmente encamina el requerimiento al sitio actual a través del componente *web interface*.

Concequences. El sistema de direccionamiento del cliente móvil se ve simplificado, pero se debe proveer de un adecuado broker lo cual involucra un trabajo adicional para los desarrolladores. El broker provee el control sobre que subgrupo de la Web puede ser accedido por los clientes móviles.

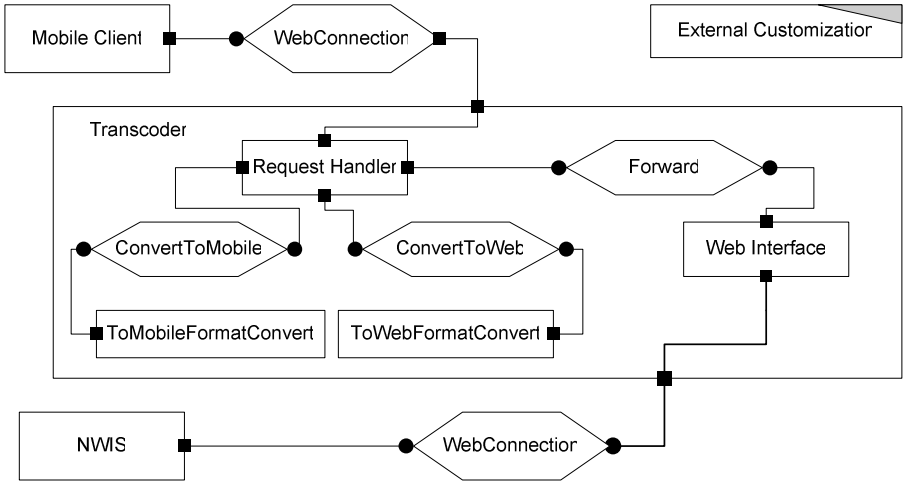
Known uses. AvantGo centraliza el acceso a un creciente grupo de canales, donde el Server AvantGo actúa como broker entre la NWIS y el actual cliente móvil. El servicio AcroClip también ofrece un acceso a través de canales, controlado por un broker propietario.

Customisation pattern: External Customiser

Intent. Proveer un mecanismo para adaptar el contenido de la Web al cliente móvil, con independencia de la NWIS de interés.

Problem. Se tiene que desplegar información desde una NWIS arbitraria en un equipo con restricciones que no soporta el formato en el cual esta codificada dicha información. Las posibilidades de realizar versiones personalizadas de la NWIS es impracticable (porque el numero de NWIS de interés es muy grande, porque el numero de dispositivos a tener en cuenta es muy grande, o porque simplemente no se esta interesado en proveer versiones personalizadas).

Solution. Crear un componente capaz de convertir la información de cualquier arbitraria NWIS a un formato adecuado para el potencial cliente. Este componente es externo a la NWIS de interés.



Rationale. El *transcoder* es un componente capaz de convertir entre la codificación y la forma estándar de un sitio Web, y la requerida por un conjunto de dispositivos móviles. El cliente móvil se conecta al *transcoder* a través de un *web connection*. El *transcoder* pasa el requerimiento a través de un conversor desde el requerimiento del móvil al requerimiento de la Web, y envía le mismo a la actual NWIS a través de el componente *web interface*. Lo mismo ocurre en sentido opuesto y la respuesta desde la NWIS es pasada a través del conversor de formato Web a formato móvil, antes de alcanzar el cliente móvil.

Concequences. El proveedor de la NWIS no tiene trabajo adicional para personalizar la información para los clientes móviles. El *transcoder* automáticamente maneja esta tarea. Construir un *trasncoder* puede ser una tarea dificultosa. Hay tal vez diferentes clientes móviles en mente, lo que significa diferencias que el *trasncoder* debe soportar. El parseo de la fuente de información original puede ser dificultoso en si mismo; ser externo y sin conocimiento del diseño interno de la MWIS; el *trasncoder* puede que tenga que hacer frente a una gran variedad de posibles transformaciones.

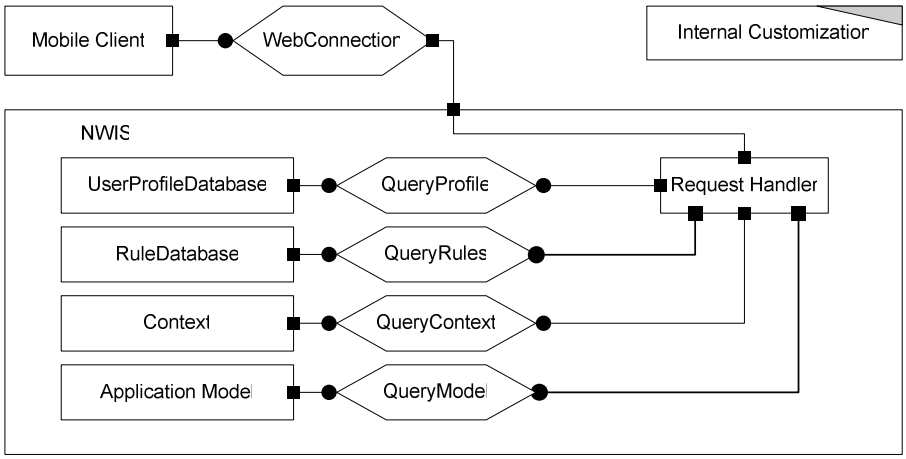
Known uses. PumaTech provee un servicio que permite el acceso a paginas Web estándares vía un dispositivo Palm OS. IBM también provee una solución de transcoder como parte de la familia de productos Websphere. AcroClip posee trasncoder integrados para distintos sitios Web, y también permite a los usuario definir sus propios.

Customisation pattern: Internal Customiser

Intent. Proveer al cliente móvil de una respuesta de la aplicación directamente adecuada para su manipulación, entonces ninguna personalización externa es requerida.

Problem. En muchas situaciones se necesita adaptar funcionalmente la aplicación dependiendo del dispositivo del cliente (un Browser de desktop, un Browser de PDA, etc.) tanto en forma dinámica como en forma fija. Tal vez se necesita no solo adaptar la información a una forma adecuada al dispositivo del cliente, sino también la aplicación debe responder aun comportamiento que puede variar. Como las reglas de configuración están fuertemente atadas a la naturaleza de la NWIS, contar con un personalizador externo que pueda manager todas estas reglas es casi imposible o al menos muy caro.

Solution. Mover los mecanismos de personalización dentro de la NWIS permite proveer diferentes respuestas dependientes del cliente (donde diferentes respuestas puede implicar diferente codificación o diferente comportamiento).



Rationale. El cliente móvil se conecta a la NWIS a través de un *web connection*. La NWIS busca información en diferentes fuentes de personalización para generar la apropiada consulta al actual modelo de aplicación. La respuesta del modelo de aplicación a la consulta personalizada es entonces retornada como la respuesta al requerimiento.

Consequences. La NWIS es especialmente diseñada para un cliente móvil que se tiene en mente, entonces hay menos posibilidades de la que la misma no se adapte al cliente perfectamente. El proveedor de contenido tendrá un trabajo adicional, puesto que ha de diseñar aplicaciones específicas, con mecanismos para manejar potenciales clientes.

Known uses. En el modelo de referencia UWA (*Ubiquituos Web Applications*), las reglas están separadas de la aplicación, y el modelo de contexto en si estas compuesto por los sub-modelos del usuario, el dispositivo y la red. En el *Munich Reference Architecture* para aplicaciones hipermedia adaptativas se utiliza un enfoque similar. WebML también utiliza una variante de este patrón.

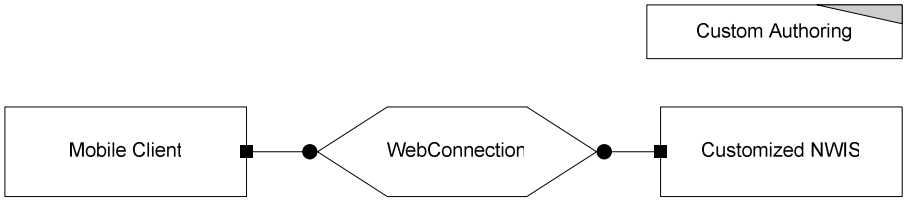
Related patterns. El *external customiser pattern* es similar al *internal customiser pattern* de forma que ambos proveen una forma automática de presentar diferentes vista de la información a los clientes móviles. Sin embargo, mientras la personalización externa pone foco en la apariencia de la información, la personalización interna focaliza en el proceso de generación de la información. Por lo tanto la personalización externa actúa como un Facade sobre el sistema, pasando a ser una solución desentendida de la información. En el *internal customiser pattern,* las reglas y el contexto con la inclusión de los mecanismos de personalización, son concientes del tipo de información que el sistema provee.

Customisation pattern: Custom Authoring

Intent. Para diseñar y construir una versión personalizada de una NWIS para un cliente móvil, es requerido un mecanismo de personalización que no sea costoso.

Problem. Se tiene que desplegar información desde una fuente de información (por ejemplo un sitio Web) a un dispositivo restringido que no soporta la forma en que la fuente de información esta codificada. No se tienen ni los recursos o la experiencia para construir o comprar un mecanismo de personalización automático.

Solution. A cada fuente de información de interés, se le solicita proveer una versión personalizada, de una forma exactamente adecuada para el cliente.



Participants. El cliente móvil extrae información directamente desde la NWIS, a través del *web connection*. Puesto que la NWIS es anteriormente personalizada para el cliente móvil, ningún procesamiento adicional es requerido.

Concequences. La información es especialmente diseñada con el cliente móvil en mente, entonces no es posible que la misma no encaje perfectamente en el mismo. El proveedor de contenido tendrá un trabajo adicional, puesto que deberá proveer una versión personalizada de su fuente de información para cada cliente móvil bajo consideración. Una actualización de la información puede ser una tarea pesada, si existen varias versiones personalizadas para mantener.

Known uses. Las *authoring guidelines* de AvantGo, alientan a los proveedores de contenido a entregar sus NWIS especialmente preparadas para hacer frente a las limitaciones de los dispositivos móviles.

Síntesis

Este trabajo presenta un conjunto de patrones arquitectónicos para Sistemas Web de Información Móvil, organizados dentro de un catalogo. Los patrones arquitectónicos tratan con vistas de alto nivel de un sistema en las cuales sus primitivas son componentes y conectores.

Se encuentran organizados en diferentes categorías y están escritos en un formato similar al propuesto en el libro de GoF.

Resulta interesante el punto de vista sugerido, de no considerar que el diseño se debe basar únicamente en las fuertes restricciones técnicas del dominio, sino que también se deben incluir aquellas relacionas con el objetivo de la aplicación.

Dentro de la taxonomía de áreas de aplicación planteada en el primer capitulo, estos patrones se adhieren al área de las aplicaciones tipo portales.

4.3 RESUMEN CAPITULO

En este capitulo se realizo una pequeña introducción sobre patrones, para luego presentar tres trabajos realizados sobre patrones en el dominio de aplicaciones móviles.

El primer trabajo se orienta casi exclusivamente a las restricciones técnicas propias del dominio, plantea una jerarquía de patrones de los cuales solo detalla dos de ellos.

El segundo trabajo, realiza una recorrido general sobre la problemática entorno de la computación ubicua (las aplicaciones móviles pueden ser consideradas una subclase de la misma). Presenta un lenguaje compuesto por 45 pre-patrones.

Por ultimo, el tercer trabajo trata sobre patrones arquitectónicos extraídos de sistemas Web de información móviles, actualmente utilizados en la industria. Plantea un punto de vista alternativo respecto a las consideraciones a tener en cuenta en el diseño de este tipo de sistemas.

CAPITULO 5

PATRONES Y CLASIFICACIÓN

En este capitulo se presentará un conjunto de patrones arquitectónicos extraídos de aplicaciones conocidas que se encuentran actualmente en uso. El concepto de patrones arquitectónicos surgió algunos años atrás y es similar a la noción de patrones de diseño, solo que los mismos tratan con vistas de alto nivel de un sistema en el cual sus primitivas son componentes y conectores.

5.1 INTRODUCCIÓN

La Arquitectura de Software es una disciplina relativamente nueva, que para sintetizar su esencia se puede expresar que describe el plan estructural de los elementos de un sistema, como los mismos interactúan y como esas interacciones se ajustan a los objetivos de la aplicación. Existen múltiples enfoques de la arquitectura, pero la mayoría coinciden en la separación de las diferentes entidades en diversas vistas. Aquí se utilizará la llamada vista conceptual. En la misma se describe la arquitectura de un sistema con un alto nivel de abstracción, donde los elementos computacionales se los denomina componentes y la interacción entre ellos, conectores. Bajo esta visión, la tarea del arquitecto es la de organizar los diferentes componentes y conectores de forma de cubrir los requerimientos impuestos para el sistema. Para más detalles sobre Arquitectura de Software referirse al Anexo 2.

Los patrones arquitectónicos son similares a los patrones de diseño en el hecho que ambos tratan de capturar, estructurar y transmitir la experiencia del diseñador. Un patrón en si, permite describir un problema recurrente junto con su solución, de manera tal que la misma pueda ser aplicada en diferentes situaciones. Existen diversas definiciones de patrones de software, de diversos autores, pero la mayoría coinciden en que los mismos se componen de tres elementos, un problema, una solución y un determinado contexto donde se aplican. El conjunto de patrones para un dominio especifico, con el tiempo forman un lenguaje que luego es utilizado por los diferentes actores del proceso desarrollo, y permite resolver problemas con un alto nivel de abstracción, por medio de la reutilización de soluciones exitosas.

Ahora bien, los patrones de diseño son utilizados en niveles de micro arquitectura, donde los componentes son las clases y objetos y los mecanismos de interacción, son los mensajes. En cambio los patrones arquitectónicos tratan con niveles de abstracción muy superiores donde los componentes ya no son clases y objetos sino por ejemplo módulos de software y las interacciones entre estos componentes se realizan a través de conectores. Para más detalle sobre Patrones en general referirse al Anexo 3.

A lo largo de este capitulo se presentara un conjunto de patrones arquitectónicos propios del dominio de las aplicaciones móviles. Se tomaron como base algunos de los trabajos realizados anteriormente en el área, donde en función a la evolución que las aplicaciones del dominio han tenido y a la proliferación de las mismas, algunos patrones pueden hoy especializarse o en algunos casos completarse.

5.2 CLASIFICACIÓN DE PATRONES

En torno al desarrollo de aplicaciones móviles se pueden trazar algunos ejes fundamentales que abarcan gran parte de los problemas propios del dominio. Como ya se dijo anteriormente el dominio de este tipo de aplicaciones es altamente restrictivo debido en gran medida a las características técnicas, propias del dispositivo móvil y a las impuestas por la red de interconexión, comúnmente inalámbrica, con todo lo que el hecho de ser inalámbrica significa (ver Anexo 1). Estos ejes serian:

- ***Temas relacionados con el acceso.*** Abarca todo aquello en torno del acceso desde la aplicación que actúa como cliente a la aplicación que actúa como servidora.
- ***Temas relacionados con adaptación.*** Se orienta a la adaptación de la información en función de las limitaciones del dispositivo donde se ejecuta el cliente.
- ***Temas de personalización.*** Involucra lo relacionado a selección por parte del usuario o en función del usuario, ya sea desde el tipo de información que quiere recibir al ingresar a un portal hasta que red utilizar de las varias disponibles.
- ***Temas de seguridad.*** Todo aquello relacionado a preservar la seguridad de la información, orientado en dos grandes grupos: la seguridad sobre el dispositivo y la seguridad sobre la conexión de acceso.
- ***Temas de Interfase.*** En este punto se incluye la temática de desarrollo centrado en el usuario.

La clasificación que se presentará a continuación se basa en estos ejes y la misma surgió de un proceso *top-down*, es decir en primero se armo la clasificación y luego se busco identificar los patrones que deberían cubrir los diferentes grupos. La misma no pretende ser definitiva, sobre todo teniendo en cuenta que este es un dominio relativamente nuevo y en constante evolución. Debería ser vista como una guía organizacional, que muy probablemente evolucionará con posteriores trabajos.

Acceso. Este grupo abarca cuestiones sobre como el acceso físico desde el dispositivo móvil se ve afectado, además de, por los requerimientos propios de las aplicación, por las limitaciones que impone el entorno móvil. Si bien hoy los dispositivos móviles han evolucionado de manera tal que la mayoría cuenta con algún tipo de conexión, o en ciertos casos con mas de uno, el tema acceso sigue siendo de gran importancia para los diseñadores. Una aplicación debe poder manejar varias alternativas de conexión, por ejemplo dial-up o través de una red WI-FI, como también debe permitir al usuario continuar trabajando cuando ninguna de ellas este disponible.
En este grupo se incluyen tópicos como, administración de la conexión, selección de caminos alternativos, ya sea en forma automática o manual; selección de métodos de compresión en función del ancho de banda disponible; seguridad; mensajería, sincronización de datos; entre otros.

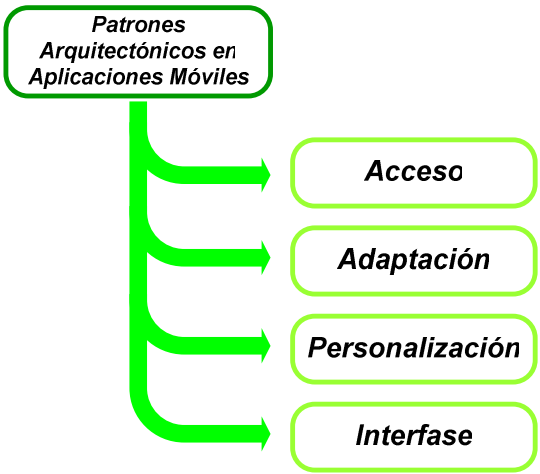
Adaptación. Esta categoría enfoca cuestiones que surgen de la situación en que un cliente móvil tiene que navegar por un sitio Web o acceder a documentos o información, los cuales no son adecuados para el dispositivo que esta utilizando. Por ejemplo el acceso a un documento almacenado en un servidor de archivos, el cual contiene imágenes de alta resolución, si bien es completamente normal para un usuario de una PC de escritorio conectada la red corporativa, deberá ser adaptado para que un usuario pueda visualizarlo en un dispositivo móvil. Las restricciones de pantalla (tamaño y resolución), memoria y poder de computo, jugaran un papel preponderante en la adaptación necesaria. Para el caso de sitios Web, la necesidad de adaptación surge en aquellos dispositivos móviles

que cuentan con Browsers especializados que no son capaces de interpretar la totalidad de la codificación o utilizan un lenguaje diferente como ser el caso de los teléfonos celulares y el protocolo WAP. Esta categoría o grupo se solapa en parte, a la planteada en el trabajo [RR04] de W. Risi y G. Rossi, "*Web navigation*".

Personalización. Esta categoría incluye temas relacionados a la utilización de métodos que permitan la configuración por parte del usuario y/o en función del usuario, de temas como selección de información a sincronizar, selección de acciones a tomar en función del contexto, selección de la información a recibir, entre otras. Esta categoría se puede decir que abarca temas que son transversales al resto de las categorías. Por ejemplo la selección del tipo de acceso a utilizar, dentro de los disponibles, puede hacerse en forma manual y esto si bien, según esta clasificación, estaría dentro de ***Acceso***, en el hecho de seleccionar se esta realizando una personalización. Por otra parte la selección de los sitios Web que el usuario desea acceder, puede considerarse una personalización que afecta por ejemplo la adaptación, ya que no es necesario plantear la adaptación de un sitio para un determinado tipo de dispositivo utilizado por un grupo de clientes si estos no desean accederlo. Un ejemplo de esto podría ser el grupo de ventas de una determinada empresa que acceso al sitio Web de la compañía a través de un dispositivo móvil provisto por la misma. Estos usuarios no accederán, por ejemplo, al sitio de información institucional, si esto ya fue personalizado con anterioridad, entonces ya no será necesario adaptar el mismo.

Interfase. En esta categoría se incluyen temas como diseño de interfases en función de las restricciones impuestas por las características físicas del dispositivo móvil, reingeniería de interfaces de aplicaciones que serán transportadas desde uso en computadoras de escritorio a dispositivos móviles, entre otros. Por ejemplo aplicaciones móviles que han sido exitosas, atribuyen su aceptación a que las mismas han sido concebidas desde un principio para dispositivos móviles, el uso de canales para acceso a sitios Web predeterminados ha facilitado la navegación del usuario móvil, quien no dispone de los mecanismos de ingreso de datos que un usuario de PC, el cual puede escribir cómodamente la URL esta interesado en acceder. Por otra parte una aplicación móvil comúnmente necesita de una interfase sencilla que le permita al usuario acceder a lo que necesita con la menor navegación posible, ejemplo de esto lo constituyen gran parte de las aplicación desarrolladas para Palm.

El siguiente diagrama muestra gráficamente la clasificación propuesta:

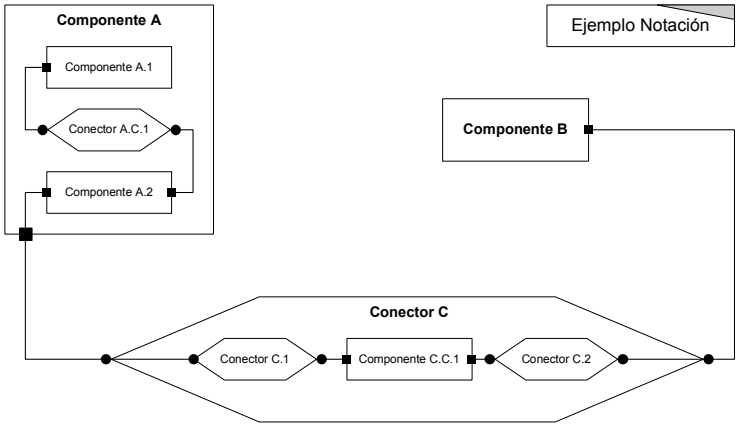


5.3 CATALOGO DE PATRONES

A continuación se presentaran algunos patrones extraídos, los cuales se clasificaran según las categorías mencionadas en la sección previa. El formato a utilizar es similar al utilizado en el libro "*Design Patterns: Elements of Reusable Object-Oriented Software*" [GHJV94], pero adaptado debido a la característica de alto nivel de abstracción que presentan los patrones arquitectónicos. Las secciones de un patrón serán:

> ***Nombre del patrón y Clasificación.*** Se tratara de dar al patrón un nombre que trasmita su esencia y que permite identificarlo dentro del catalogo. La clasificación lo ubicará dentro del esquema presentado en la sección anterior.
> ***Propósito.*** Consiste en una frase breve que describa que es lo que hace el patrón, en que se basa y cual es el problema que resuelve.
> ***Problema.*** Descripción con cierto grado de detalle el problema que el patrón ataca.
> ***Solución.*** Se trata de dar la mayor cantidad de detalles sobre la solución propuesta, de forma que la misma pueda ser fácilmente comprendida.
> ***Motivación.*** Un escenario ilustra el problema y como los componentes del patrón lo resuelven.
> ***Consecuencias.*** Describe como consigue el patrón sus objetivos, cuales son las ventajas e inconvenientes y los resultados de utilizarlo.
> ***Usos Conocidos.*** Ejemplos del patrón en sistemas reales.

De esta manera se mantendrá el formato planteado en "*An architectural pattern catalogue for mobile web information systems*" [RR04], en el cual además se incluye un diagrama para explicar la estructura de la solución. En referencia a los diagramas, con el objeto de mantener un cierto grado de sencillez, debe notarse que los mismos no se ajustan completamente a UML. La siguiente figura muestra un ejemplo de la notación utilizada.



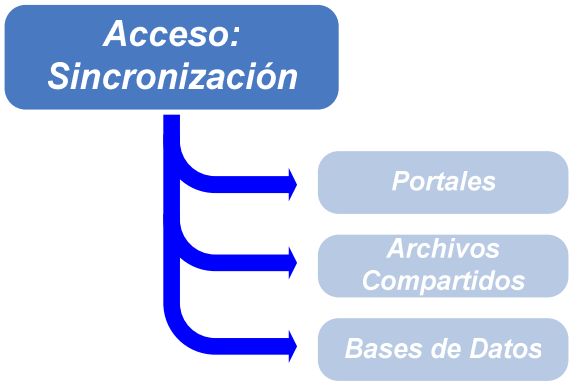
Se puede ver que el Componente A se conecta con el Componente B por medio del conector C. Un componente es una entidad arquitectural que representa una unidad conceptual de computación. El Componente expone su funcionalidad a través de uno o mas puertos (pequeños cuadrados negros), e interactúa con otros Componentes utilizando Conectores que representan los mecanismos de comunicación. Un Conector conecta a dos o más Componentes a través de sus puertos. Por ultimo tanto componentes como conectores pueden ser anidados.

Es importante notar que estos componentes y conectores arquitectónicos pueden no representar exactamente el mapa de componentes y conectores de una determinada implementación. En otras palabras el conjunto de entidades conceptuales que se organizan para alcanzar una funcionalidad puede diferir de los bloques computacionales que se implementan realmente, algunas funcionalidades de una entidad pueden ser ubicadas en diferentes bloques constructivos y viceversa.

5.3.1 Acceso: Sincronización

Este patrón fue presentado por primera vez para este dominio por J. Roth y básicamente describe los mecanismos para mantener actualizada idéntica información almacenada en varios dispositivos, móviles o no, que mantienen una conexión esporádica. La información es consultada y modificada por los usuarios de dichos dispositivos.

Aquí lo que se verá es una especialización de este patrón teniendo en cuenta las áreas de aplicación presentadas en el capitulo 2, esto es: portales, administración de documentos compartidos y el acceso a bases de datos



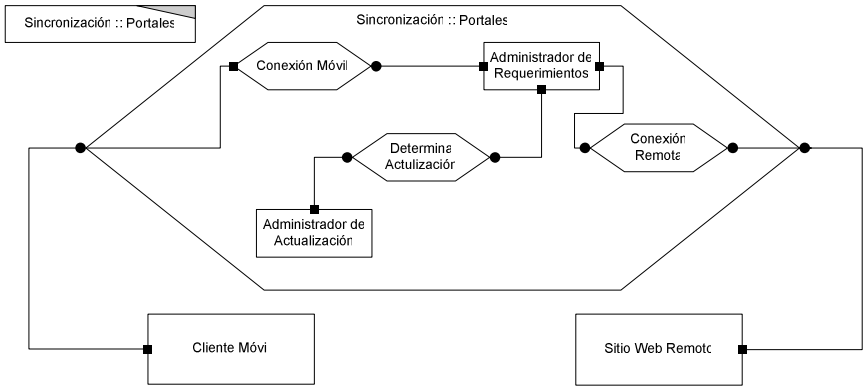
Debe notarse que en el caso del patrón de Sincronización de portales, el mismo fue descripto por W. Risi y G. Rossi, como *Static Accessing*. Aquí se le ha dado una visión ligeramente diferente y complementaria en algunos aspectos.

Sincronización :: Portales

Propósito: Permitir al cliente móvil la navegación a través de sitios Web cuando el dispositivo no sea capaz de acceder al mismo.

Problema: El usuario móvil desea acceder a un sitio Web desde un dispositivo con restricciones que no le permiten hacerlo. Estas restricciones pueden ser propias del dispositivos, por ejemplo no contar con ningún funcionalidad de interconexión, o impuestas por la situación, por ejemplo la no disponibilidad de servicio de red en el lugar en el que se encuentra.

Solución: Disponer de una copia local de los sitios Web que el cliente desea acceder, los cuales previamente deberán ser almacenados en forma local y luego actualizados cada vez que el dispositivo móvil pueda establecer una conexión con el servidor donde reside el sitio Web original. El siguiente grafico esquematiza esta solución.



Motivación: El conector de Sincronización es una entidad que contribuye a que el cliente móvil pueda acceder al sito Web independientemente del estado de la conexión.
Cada vez que la conexión al Sitio Web Remoto este activa, y el cliente solicite el acceso a un Sitio Web el conector de Sincronización deberá cumplir su función, esto es: el Administrador de Requerimientos, recibe el pedido de acceso, luego pasa este al Administrador de Actualización quien decidirá si el sitio local debe o no ser actualizado. Finalmente al Administrador de Requerimientos establecerá la conexión con el Sitio Web Remoto, a través del conector Conexión Remota.

En el escenario de una conexión inestable, al realizar la sincronización en primer medida del sitio local con el sitio remoto, se esta asegurando que el cliente móvil accederá a una copia actualizada. Si se conectará al cliente directamente al sitio remoto y la conexión se perdiera, el cliente debería dirrecionarse al sitio local, pudiendo esto reflejar inconsistencias en la información que el usuario recibe.

Consecuencias: Si se dispone del conector de sincronización adecuado para el dispositivo en cuestión, el mismo permitirá la funcionalidad de acceso a un sitio Web. Se debe tener presente que durante el periodo donde el cliente no disponga de conexión, la información ofrecida por el sitio puede cambiar. Este patrón no es aplicable en los casos que dentro del sitio Web se tengan links que apunten a por ejemplo aplicaciones *back-end*, este seria el caso de un usuario que antes de salir de la oficina sincroniza parte de la Intranet en su dispositivo móvil. Como ya se sabe, un portal agrega varias fuentes de información, realizar la copia todas ellas, por razones obvias, no será posible.
Nótese que la sincronización puede hacerse a nivel de contenido, código o datos.

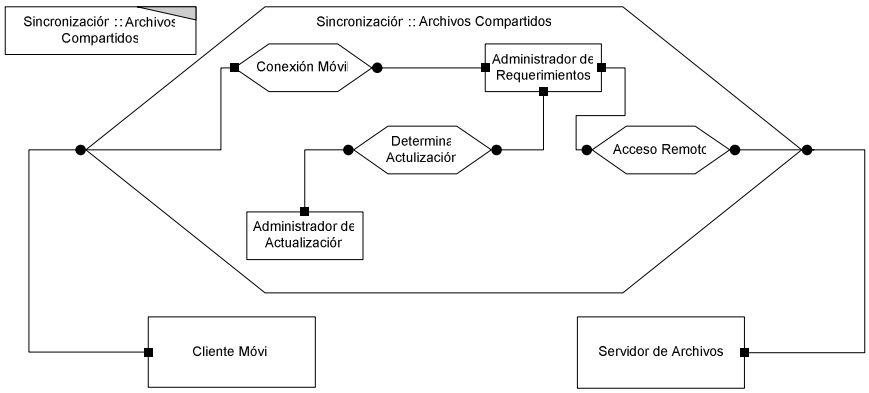
Usos Conocidos: *Intellisync® SyncML Server* es un producto que permite la sincronización de todo tipo de información entre un repositorio central y dispositivos móviles conectados a través de varios tipos de redes. Es un producto que permite ofrecer sobre una red inalámbrica servicios de sincronización multipunto. Para más información consultar: www.intellisync.com
AvantGo es un servicio de software libre que permite acceder desde PDAs o Smartphones a sitios Web especialmente formateados (a través de lo que denominan canales) ya sea en forma online o previamente descargados al dispositivo. Posee un mecanismo de sincronización que actualiza los sitios locales del dispositivo, cada vez que se establezca una conexión a Internet. Para mas información consulta: www.avantgo.com

Sincronización :: Archivos Compartidos

Propósito: Proveer un mecanismo que permita que la información en documentos pueda ser compartida manteniendo la concurrencia de los mismos.

Problema: El usuario necesita acceder a documentos ubicados en un repositorio central, pero en un entorno móvil la conexión entre su dispositivo y el sistema de administración de documentos no puede ser asegurada.

Solución: Se realiza una copia local del árbol de directorios y los documentos de interés y luego se mantienen actualizados mediante sincronizaciones en los periodos que se cuente con conexión al sistema central de administración de documentos.



Motivación: Al igual que el patrón para sincronización de portales se tienen similares entidades con similares funciones. Solamente el Administrador de Actualización sufre cambios en las mismas, ya que en este caso el usuario puede modificar los archivos residentes en el dispositivo móvil y la sincronización deberá llevarse a cabo en los dos sentidos, es decir actualizar los archivos almacenados en el sistema local o actualizar los archivos almacenados en el servidor. Para la sincronización de portales se asumía implícitamente que el portal no era modificado por el cliente móvil.

Consecuencias: Este patrón contribuye a que el sistema móvil disponga de los documentos ubicados en la locación remota, independientemente de la conexión. Comúnmente el conector se encuentra integrado al sistema operativo. La sincronización del árbol de directorios puede realizarse en forma completa o incremental, según se priorice la exactitud entre los datos del Server y el cliente, o la eficiencia de la comunicación. La sincronización de documentos se lleva a cabo en forma completa, es decir, se destruye la copia anterior sobrescribiéndola con la nueva copia del archivo.

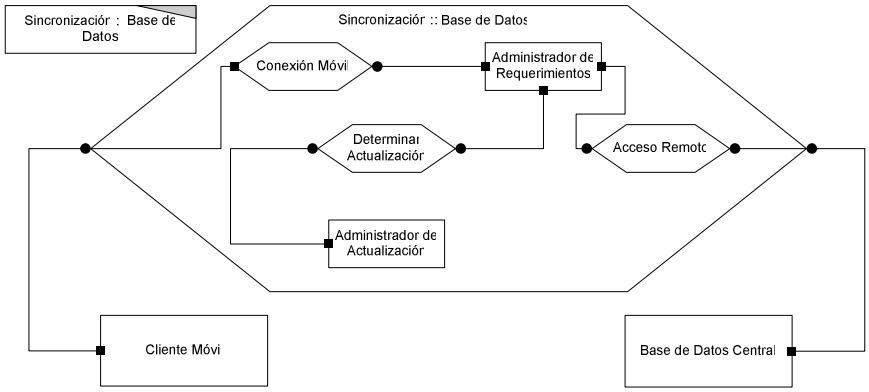
Usos Conocidos: *Windows 2000 Advance Server* permite a usuarios de dispositivos móviles trabajar con archivos de red estando desconectado de la misma. Los archivos "*offline*" proveen la manera de asegurar que se esta trabajando con la mas reciente versión del archivo de red y que el trabajo realizado sobre los mismos será sincronizado cuando se disponga nuevamente de la conectividad. Usando archivos "*offline*" el usuario puede navegar a través de carpetas compartidas y discos de red sin importarle el estado de la conexión. Una vez reconectado a la red, *Synchronization Manager* es el encargado de actualizar los archivos de red con cambios desde que se estuvo desconectado. Para mas detalles consultar: *Windows 2000 Advance Server Documentation*.

Sincronización :: Bases de Datos

Propósito: Permitir a los usuarios móviles acceder a datos remotos desde cualquier lugar o momento independientemente del estado de la conexión.

Problema: En un ambiente móvil, copias de los datos pueden existir en distintos sistemas clientes, los que no se encuentran continuamente conectados a la base de datos central, por lo cual sistema de base de datos no es capaz de prevenir cambios simultáneos por mas de un usuario. Los datos del sistema cliente se desactualizan durante los periodos en que el cliente no esta conectado.

Solución: Proveer de un mecanismo que, según un seguimiento de las modificaciones aplicadas a la información local, y el estado de la conexión, realice la sincronización de los datos.



Motivación: Las entidades que componen el conector de sincronización son similares, pero nuevamente existen diferencias en las funciones del Administrador de Actualización. Si bien la actualización debe llevarse a cabo en los dos sentidos, al igual que en el caso de la sincronización de archivos, la situación es más compleja ya que ahora no se trata de reemplazar un archivo por otro en forma completa. El Administrador de Actualización deberá llevar un detalle de las transacciones realizadas sobre la base de datos local para luego cotejar con las realizadas en la base de datos central y en función de estas intercambiar modificaciones y resolver conflictos. Estos conflictos podrán ser resueltos en forma automático o según su complejidad, requerirán la intervención del usuario.

Consecuencias: Este patrón es útil en aquellas aplicaciones donde la importancia de la actualización de los datos tenga menor importancia que la necesidad de la movilidad. El usuario debe ser advertido de que los datos pueden haber sido modificados en los periodos donde no se tuvo conexión, de manera que si el tomase decisiones sobre estos, las mismas puedan ser revertidas cuando los datos se encuentre sincronizados nuevamente.

Usos Conocidos: *Microsoft® SQL Server™ 2005 Mobile Edition 3.0* permite el acceso concurrente a la misma base de datos. Múltiples usuarios implica múltiples *threads* o procesos puede estar accediendo a la base de datos mientras un *thread* esta sincronizando la base de datos móvil con el SQL Server. Entre otras cosas este esquema *multi-thread* permite sincronizar los datos en background mientras el usuario continua trabajando.
Pylon Application Server es producto que extiende aplicaciones basada en un domino Lotus, como automatización de fuerza de ventas, inventario y

aplicaciones para servicios de campo, a dispositivos móviles. Cuenta con servicio de sincronización que permite mantener actualizada la información de los dispositivos bajo el mismo concepto visto hasta ahora, es decir el de poder acceder a la misma estando *online* o no. Para mas detalles consultar: http://www.ianywhere.com/products/pylon

5.3.2 Acceso: Emulación

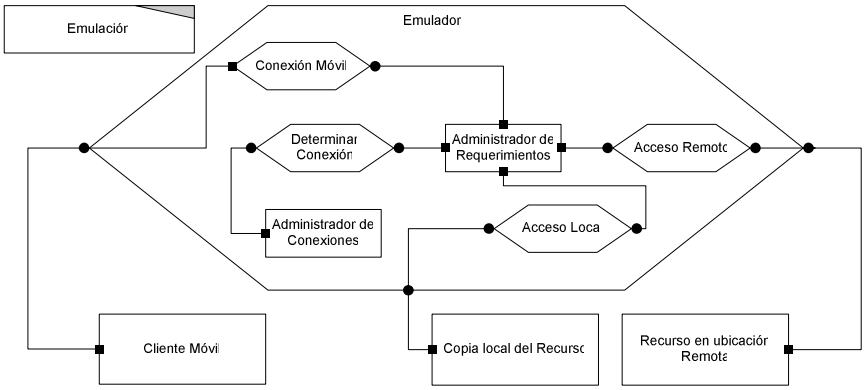
Este patrón trata el tema de generar para el usuario un entorno donde no importa el estado de la conexión que tenga el dispositivo móvil en ese momento, se pueda continuar accediendo a la información de su interés. En este caso el patrón no será especializado según las áreas de aplicación ya que el aspecto de emulación es propio del acceso y las restricciones del dominio móvil en general. Pueden decirse que este patrón en conjunto con el patrón sincronización colaboran para el mismo objetivo, el de permitir al usuario independizarse de los problemas de acceso.

Emulación

Propósito: Enmascarar la no disponibilidad de acceso a la información en línea.

Problema: Las características propias de una conexión inalámbrica, como la perdida de conectividad, no deben afectar el acceso a la información. Independientemente del tipo de conexión, la misma puede perderse por un sin numero de motivos. Falta de cobertura en el sitio en que se encuentra el dispositivo, desperfectos técnicos, entre otras.

Solución: Mantener una copia local de la información para que en caso de problemas de conectividad, a través de un emulador, se encamine la solicitud de acceso del cliente hacia dicha copia.



Motivación: El conector Emulador controla el acceso por parte del cliente móvil, a la copia local del recurso o al recurso original ubicado en el dispositivo remoto. Ante un pedido de acceso el Administrador de Requerimientos consulta al Administrador de Conexiones y en función de la respuesta obtenida encamina el pedido hacia la copia local o el recurso remoto.
El Administrador de Conexiones entre otras funcionalidades mantiene un seguimiento del estado de la conexión.

Consecuencias: El cliente móvil no se ve afectado por las intermitencias que puede sufrir la conexión de red. El Emulador depende de otra entidad para tomar la decisión de encaminar el pedido. Esto encapsula el problema de la red en otro

componente que puede estar diseñado para funcionalidades mas avanzadas que la simple detección de conexión o no, como ser por ejemplo la administración de ancho de banda.

Usos Conocidos: *Windows 2000 Advance Server* permite trabajar con archivos ubicados en la red aun estando desconectado de las misma. Cuando se pierde la conexión el usuario es notificado de este evento y puede continuar trabajando normalmente ya que el sistema operativo de su dispositivo móvil redirecciona sus requerimientos a una copia local. El sistema simula el acceso a la información. Para el caso del servicio de impresión en red, el usuario puede enviar su trabajo normalmente, el que será encolado hasta que se reestablezca la conexión de red. Para mas detalles ver: *Windows 2000 Advance Server Documentation*

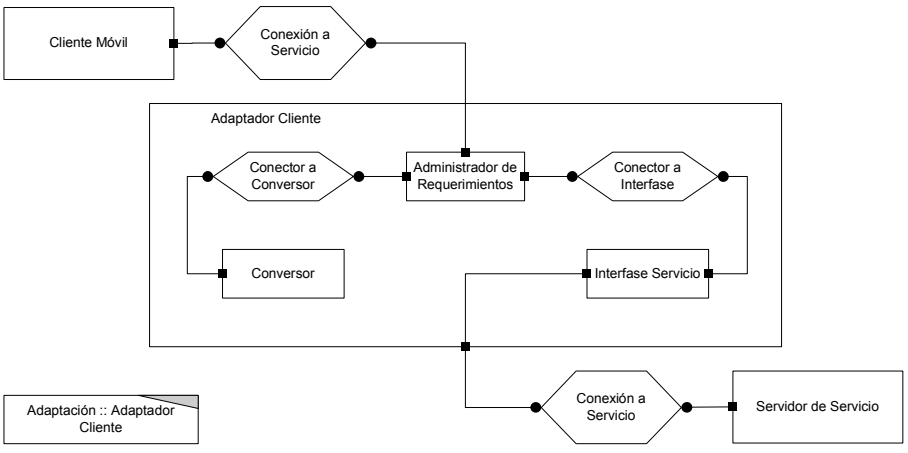
5.3.3 Adaptación: Adaptador Cliente

En este patrón se prioriza que el servicio ofrecido por el proveedor de información, ya se una sitio Web, un Servidor de Archivos, o una base de datos, no sufra modificaciones para poder atender a clientes móviles.

Propósito: Adaptar la información recibida a las restricciones propias del dispositivo móvil

Problema: La información provista por un determinado servicio no tiene las características necesarias como para ser presenta o administrada en el dispositivo móvil. Se pretende que el proveedor del servicio no deba realizar cambios en el mismo para poder atender clientes con restricciones técnicas.

Solución: Proveer al cliente móvil de un mecanismo que le permita transformar la información recibida según sus capacidades.



Motivación: El Adaptador Cliente es una entidad que se encarga de convertir el formato estándar de la información provista por el servicio, a un formato que pueda ser manejado por el cliente móvil.
Cada vez que se recibe información a través de la Interfase del Servicio, el Administrador de Requerimientos encamina la misma al componente Conversor, el cual conoce todas las limitaciones del cliente, para que realice la adaptación antes de entregársela al cliente móvil.

En el sentido inverso, el requerimiento de información podría también ser adaptado al formato que necesite el proveedor del servicio. Esto obliga a que el componente conversor conozca también las características del proveedor del servicio, complicando su desarrollo.

Consecuencias: Para el proveedor del servicio las características del cliente son irrelevantes, no deberá realizar ningún cambio para atender clientes móviles.
El dispositivo móvil deberá tener el suficiente poder de cómputo como para poder realizar las conversiones necesarias.
Para simplificar el conversor, el cliente móvil podría utilizar el mismo protocolo de comunicación que el de un cliente estándar del servicio, entonces todo pedido de información seria enviado por el Administrador de Requerimientos directamente al proveedor sin pasar por el Conversor.
El componente Conversor será propio de un determinado tipo de cliente móvil, se deberán desarrollar tantos conversores como tipos de clientes se tenga.
Dado el aumento en el poder de cómputos, capacidades de almacenamiento, resolución de pantallas, que han tenido con el correr del tiempo los dispositivos móviles, indicaría que el trabajo del componente conversor debería disminuir.
No se prioriza el uso eficiente del medio de comunicación.

Usos Conocidos: Microsoft Windows CE 5.0 - Pocket Internet Explorer ofrece importantes capacidades a los dispositivos basados en Windows Mobile, para obtener una navegación por la Web satisfactoria sin la necesidad de conectarse a través de un Proxy adaptador de contenido.
La versión '8.5 Beta' de Opera, basada en la misma versión del navegador para ordenadores personales, incorpora una tecnología que reduce las páginas Web para adaptarlas a pantallas pequeñas.

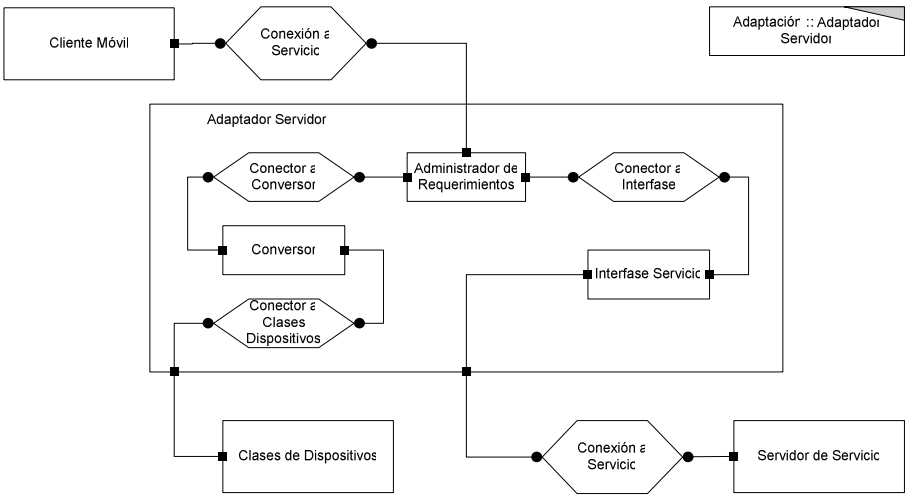
5.3.4 Adaptación: Adaptador Servidor

Este patrón trata de proveer al cliente móvil de la información solicitada en un formato acorde a las restricciones que el mismo posee. A diferencia del patrón Adaptador Cliente, el proveedor del servicio tiene conocimiento de todos los tipos de clientes a los que atiende.

Propósito: Proveer al cliente móvil de la información en un formato adecuado para su manejo.

Problema: El cliente móvil quiere acceder a información brindada por un determinado servidor de servicios. El dispositivo móvil tiene fuertes restricciones en cuanto a su poder de cómputo, almacenamiento, y tamaño y resolución de pantalla.

Solución: Realizar la adaptación de la información del lado servidor, proveyendo de la repuesta adecuada según el tipo de cliente móvil.



Motivación: El cliente móvil se conecta al proveedor del servicio a través del Conector de conexión, solo que entre el proveedor del servicio se antepone el componente Adaptador. En este componente, el administrador de requerimientos es el encargado de encaminar el pedido para hacia el Conversor para luego pasarlo al componente Interfase de Servicio. El Conversor dispone de un conector a un componente externo, donde se encuentran la información de las clases de dispositivos, la cual utilizará para llevar a cabo la conversión.

Consecuencias: Normalmente el poder de cómputos de un servidor es superior al de un dispositivo móvil, por lo que mover el conversor a este, puede ser la mejor opción.
El proveedor del servicio, deberá conocer a todas las clases de clientes que atenderá, es decir sus características y restricciones. Esta información pasará a ser meta información del servicio. La misma requerirá un mantenimiento exhaustivo, dada la gran proliferación de clases de dispositivos.
No conocer la clase de dispositivo del cliente móvil que solicita el servicio, implicará no poder brindarle el servicio, o brindarle uno muy limitado.
El desarrollo del conversor puede llegar a ser altamente dificultoso en un entorno donde la variedad de dispositivos sea muy grande.
Bajo esta solución se hace un uso más eficiente del medio de comunicación.

Usos Conocidos: *IBM WebSphere Everyplace Mobile Portal* es un producto de la familia *WebSphera Everyplace Service Delivery* que permite el desarrollo de contenido y servicios independientemente del dispositivo móvil en el que serán presentados. Se basa en el concepto de mantener el contenido en un formato independiente del dispositivo, y un repositorio con las características de los posibles clientes móviles, lo que le permite ante un requerimiento realizar la adaptación adecuada antes de proceder al envío.

5.3.5 Personalización: Entrega Personalizada

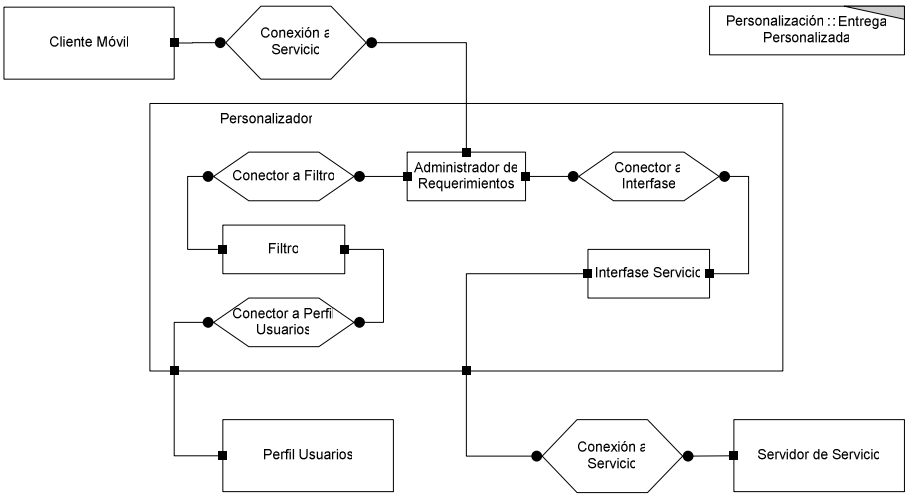
Este patrón aplica a aquellas situaciones donde se desea controlar el acceso a información específica, según el interés, rol que desempeña, hábitos o contexto en el que se encuentra el usuario.

Entrega Personalizada

Propósito: Entregar al cliente móvil información ajustada a sus necesidades.

Problema: Ante el requerimiento del cliente se pretende entregar información acorde a una serie de características del mismo, las cuales pueden ser estáticas o dinámicas. A la vez se pretende incrementar la usabilidad y mejorar la eficiencia de una determinada aplicación, al personalizar su presentación.

Solución: Crear un componente capaz de interactuar con el cliente móvil y el proveedor de servicio para ajustar la respuesta de este ultimo en función de una serie de características propias del usuario.



Motivación: El cliente móvil realiza el pedido de información a través del conector del servicio, el cual lo entrega al Personalizador, donde el Administrador de Requerimiento encamina el mismo al componente Filtro. Este componente a través de un conector accede al componente externo Perfil de Usuarios. Una vez recuperada la información del usuario, el componente Filtro devuelve el requerimiento "ajustado" para que sea enviado al proveedor del Servicio.
La información del perfil del usuario puede ser estática, es decir datos como intereses, roles, hábitos, preferencias; o dinámica como datos de su contexto actual.
El componente Filtro puede tener funcionalidades como agregación de la información, cálculos estadísticos, entre otras.

Consecuencias: Al interactuar con el servicio el usuario percibe eficiencia y control. Se tiene un control sobre que información brindar a cada usuario en cada momento.
El mantenimiento de los datos correspondientes a los perfiles de usuarios, puede ser una tarea complicada. La incorporación de datos "dinámicos" puede ser opcional. La implementación de este patrón en forma completa puede ser compleja y costosa.
La aplicación del proveedor de servicios, puede ser independientemente una nueva aplicación, una aplicación modificada o una aplicación que no sufrió ningún cambio.

Usos Conocidos: *Conference Assistant*, aplicación del grupo *Future Computing Environments* del *Georgia Institute of Technology,* provee información a los

asistentes en función a su perfil y ubicación actual, utiliza una variante simplificada de este patrón.

5.3.6 Interfase: Entrada Rápida

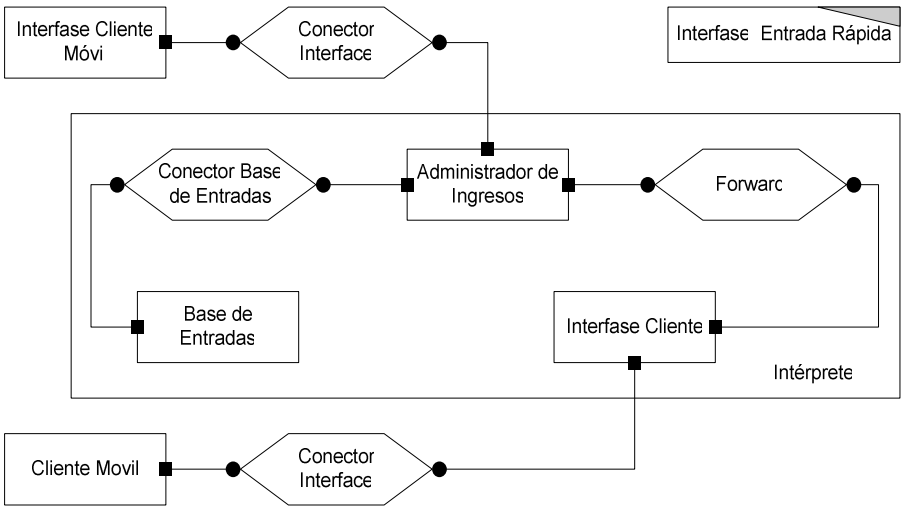
En este patrón se trata el problema que presentan los dispositivos móviles para el ingreso de datos dadas las características de sus interfaces.

Entrada Rápida

Propósito: Proveer de una mecanismo que permita el ingreso de datos de forma mas eficiente.

Problema: El ingreso de datos a través de la interfase altamente restrictiva provoca tareas repetitivas e ineficientes por parte del usuario. Se necesita de un modulo que optimice el ingreso y que a su vez no vuelva complejo al diseño del cliente móvil.

Solución: Crear un componente que haga las veces de interprete entre la lógica de la aplicación y el ingreso de datos por parte del usuario.



Motivación: El ingreso de datos a través de la interfase es tratado previamente por el componente Intérprete antes de ser enviado al cliente móvil. Una posible implementación seria el caso de un teléfono celular y el ingreso de caracteres a través del teclado de 12 teclas. Ante cada ingreso el administrador de entradas realizará una búsqueda en la base de entradas tratando de predecir la palabra que el usuario esta tratando de ingresar, una vez encontrada y confirmada por el usuario, la enviará al cliente móvil. En caso de no encontrarla el usuario deberá ingresar todos sus caracteres pero antes de que el Intérprete la envíe al cliente registrara la misma en la base de entradas para la próxima vez que el usuario deba ingresarla.

Consecuencias: Nótese que tanto se este utilizando un modelo MVC (*Model – View – Controller*) o un modelo PAC (*Presentation – Abstraction – Control*) para el diseño de interfases de la aplicación, este patrón estaría trabajando sobre el *Controller* o *Control*, según sea uno otro respectivamente.

Desde el punto de vista de la aplicación la existencia o no de Intérprete, es transparente.
El usuario percibe una mejora en la eficiencia en el ingreso de datos.
Así como se dio el ejemplo del ingreso de caracteres a través de un teclado telefónico, se podría aplicar también este patrón, por ejemplo, al ingreso de símbolos o gestos en una pantalla "*touch-screen*" de un dispositivo PDA.

Usos Conocidos: T9 utiliza una implementación de este patrón para teléfonos celulares. Fabricantes como *Motorota*, *Samsung*, *Sony Ericsson*, entre otros incorporan T9 para el ingreso de texto en las aplicaciones que corren en sus equipos por ejemplo Agenda y SMS.
TealScript™ de *TealPoint Software* es un sistema de reconocimiento de texto para PDAs que el usuario puede personalizar. Para más información consultar http://www.tealpoint.com/softscrp.htm
Graffiti® de Palm es un sistema de escritura que permite introducir datos rápidamente en un dispositivo PDA, esta es otra implementación de este patrón para dispositivos *handhelp*.

5.4 RESUMEN CAPITULO

A lo largo del presente capitulo se presentó una catalogo de patrones para aplicaciones móviles, haciendo la salvedad que al momento no se encuentra completo, pero podrá ir creciendo con trabajos futuros. Este catalogo muestra una posible clasificación basada en los distintos *concerns* propios del dominio, e incluye al menos un patrón de cada una de clases.

CAPITULO 6

USO DE LOS PATRONES EN APLICACIONES CONCRETAS

En este capitulo se muestra como los patrones presentados anteriormente pueden ayudar a describir la arquitectura de una solución comercial. Se analizaran varias aplicaciones en uso y se planteara un escenario en el cual se necesita proveer de una solución a una necesidad dentro de un entorno móvil.

6.1 INTRODUCCIÓN

Los patrones como concepto que permite capturar conocimiento de soluciones exitosas para un determinado problema, forman parte de las herramientas que facilitan al diseñador, una manera de desarrollar soluciones basadas en la reutilización. Ante un determinado requerimiento el diseñador buscará en el catalogo de patrones el que mas refleje la situación, para luego adaptar la solución que este prescribe al contexto de trabajo actual.

Hoy en día los catálogos de patrones son la fuente más común de patrones, y aun no estando completos como para generar todos los programas de un dominio, colaboran en tareas como el desarrollo de nuevas aplicaciones o la descripción de aplicaciones existentes.

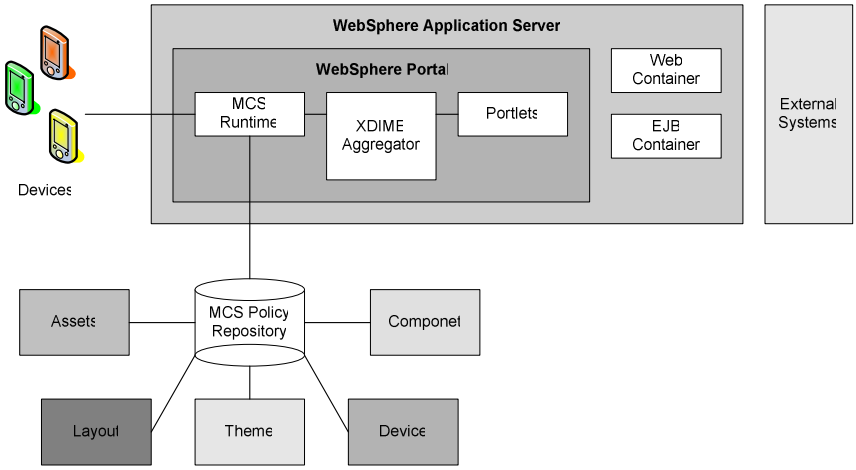
Un catalogo, además del evidente fin de agrupar un conjunto de patrones, colabora en desarrollar un vocabulario que puede ser utilizado por todos lo involucrados en el proceso de desarrollo, lo cual lleva al proceso a un nivel de abstracción superior. Por ejemplo al hablar de sincronización o adaptación automáticamente uno ya puede tener la imagen mental de las soluciones que los patrones para esta área prescriben.

6.2 ANÁLISIS DE WEBSPHERE EVERYPLACE MOBILE PORTAL

WebSphera Everyplace Mobile Portal es parte de la familia *WebSphera Everyplace Service Delivery* [CYS05], productos diseñados para facilitar la agregación de contenido y la rápida entrega de servicios a un gran numero de suscriptores y dispositivos. Pensado para ayudar los Proveedores de Servicios a adaptar, administrar, transformar y escalar aplicaciones existentes ya sean tipo Web o no.

El concepto básico es el de mantener un repositorio con el contenido en un formato genérico (basado en XML) independiente del tipo de dispositivo con el que se pretenda acceder a el, y otro repositorio con la información necesaria para realizar ante un requerimiento de un cliente móvil, la transformación del formato genérico al formato con el lenguaje de marcación soportado por el dispositivo móvil.

El siguiente grafico muestra la arquitectura básica de este producto.



A continuación una breve descripción de los componentes:

Multi-Channel Server (MCS)

Es el componente que transforma el marcado basado en XML (*XDIME – XML-based Device Independent Markup Extentions*) del contenido, en el lenguaje de marcado individual de un dispositivo (WML, cHTML, etc.). El MCS utiliza un repositorio de políticas para administrar un gran número de dispositivos como PDAs, teléfonos celulares, smartphones, Web TVs, entre otros. Este repositorio es un conjunto de archivos de políticas que definen la presentación características de un dispositivo.

Device policy

Las políticas de dispositivos son almacenadas en un archivo XML que contiene los atributos específicos de los dispositivos soportados por MCS. El repositorio de las políticas de dispositivos puede ser actualizado a través de la suscripción a un servicio brindado por el proveedor del producto.

Layout policy

Las políticas de layout especifican las posiciones físicas de los elementos de una página. MCS administra las políticas y mapas de recursos de los dispositivos.

Theme policy

Las políticas de temas especifican el *look and feel* para cada dispositivos

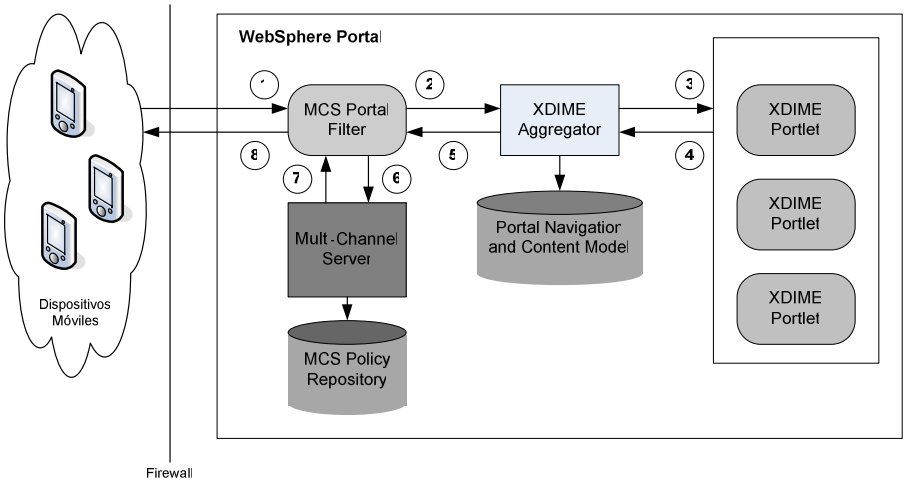
Component policies

MCS utiliza las políticas de componentes para manejar contenido complejo como imágenes, audio, esquemas y elementos visuales dinámicos.

XDIME Aggregator

Este componente extiende la funcionalidad de agregación del portal existente para soportar el lenguaje de marcado propio del producto, XDIME.

La siguiente figura describe un escenario típico de interacción, donde los números corresponden a pasos claves del proceso.



1. *WebSphere Portal* recibe el requerimiento de un dispositivo cliente. Este compara el tipo de cliente con los soportados para determinar el tipo de lenguaje de marcación apropiado
2. *MCS Portal Filter* invoca a *XDIME Aggregator* a procesar el requerimiento. Este determina que links de navegación, iconos, y texto alternativo, mostrar en la pagina.
3. El *XDIME Aggregator* pasa el requerimiento al los *XDIME Portlets* para su procesamiento.
4. Los *XDIME Portlets* devuelven su contenido como XDIME al agregador.
5. El contenido agregado en el lenguaje de marcado XDIME es retornado al *MCS Portal Filter*
6. El *MCS Portal Filter* pasa el contenido al *Multi-Channel Server*, el cual lo transforma en un lenguaje de marcación apropiado para el dispositivo, según las políticas almacenadas en el repositorio.
7. MCS finalmente envía, en un lenguaje de marcación nativo del dispositivo, el resultado del requerimiento original, al *MCS Portal Filter*.
8. El *MCS Portal Filter* entrega el resultado al dispositivo que originó el requerimiento.

El producto descripto es un ejemplo práctico del patrón de adaptación, *Adaptación Servidor*, donde se pretende entregar al cliente móvil la información en el formato adecuado según las características del mismo. El Servidor tiene conocimiento de las restricciones de todos los clientes a los que da servicio. Si un dispositivo no se encuentra en la base de datos del servidor, no se le podrá dar servicio.

Obsérvese que en este caso el patrón contribuye a lograr un alto grado de abstracción en el análisis, llevando a identificar la arquitectura de este producto bajo un único concepto.

6.3 ANÁLISIS DE WINDOWS 2000 SERVER Y SU SERVICIO "OFFLINE FILES"

La solución de *Offline Files* de Microsoft, provee una manera de asegurar que el usuario estará trabajando con la versión mas reciente del los archivos de red y que su trabajo será sincronizado cuando la conexión de red se encuentre nuevamente disponible. Utilizando esta funcionalidad el usuario podrá navegar a través de carpetas compartidas y discos de red sin preocuparse por el estado de la conexión de red.

El usuario deberá indicar a priori cuales archivos, carpetas o discos querrá utilizar aun estando desconectado de la red. Así como también deberá indicar al *Synchronization Manager* de su computadora portátil la manera de sincronizar sus archivos, ya sea cuando se conecte o desconecte de la red.

Mientras el usuario este desconectado, solo será capaz de ver aquellos archivos que indicó como "disponibles *offline*" y cualquier archivo que haya creado después de que la conexión de red se perdió. Los permisos que se tengan sobre los archivos de red y carpetas se mantendrán sin cambios, por ejemplo un archivo marcado como solo lectura se mantendrá así aun estando desconectado de la red.

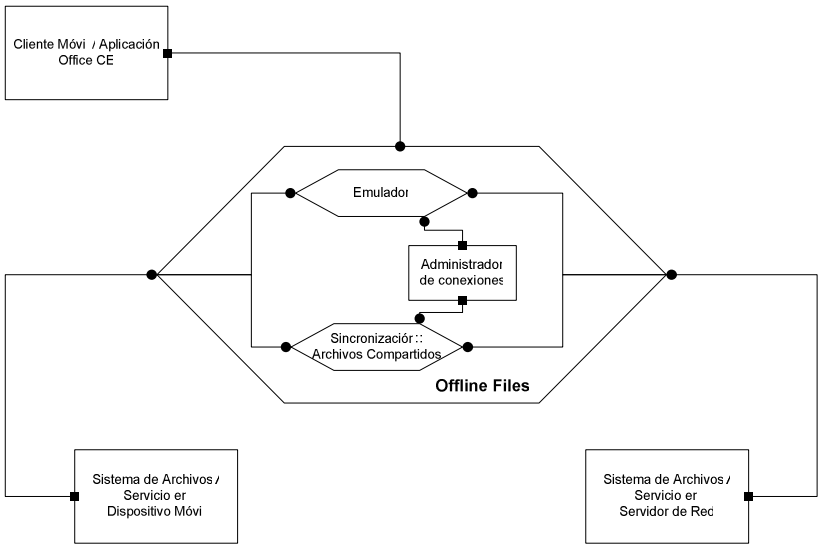
Cuando el usuario se encuentre desconectado de la red, podrá imprimir en la impresora local, pero no podrá imprimir en las impresoras compartidas de la red. En caso necesitar imprimir en una impresora de red, el archivo será encolado para que cuando la conexión de red se reestablezca, el mismo sea enviado a la impresora correspondiente.

Al finalizar su trabajo *offline*, el usuario deberá sincronizar sus archivos nuevamente. Cuando se realiza la sincronización los archivos que fueron abiertos o actualizados mientras se estuvo desconectado de la red, son comparados con las versiones de los archivos que están en la red. En caso de los archivos modificados por el usuario no hayan sido modificados por ninguna otra persona mientras se estuvo desconectado, los mismos serán copiados a la red. Si alguien realizo cambios sobre el mismo archivo de red que el usuario actualizó, tendrá la opción mantener su versión, mantener la de la red o mantener ambas. Para mantener ambas, el usuario deberá renombrar su versión y ambos archivos aparecerán en ambas locaciones. Si el usuario borro un archivo de red estando desconectado y alguien más realizo modificaciones sobre ese archivo en la red, el archivo es borrado del sistema local pero no de la red. Si el usuario realizo cambios sobre un archivo que se borro en la red, tendrá la opción de guardar su versión en la red o borrarla de su sistema local. Y por último, si un archivo es agregado a una carpeta que el usuario dispuso como "disponible *offline*", mientras él estuvo desconectado, este archivo será copiado a su sistema local.

Según las prestaciones de este servicio se podría esquematizar la parte mas representativa de su arquitectura a través de dos de los patrones arquitecturales presentados, estos son el patrón de acceso Sincronización Archivos Compartidos y el patrón de acceso Emulación.

El patrón Sincronización Archivos Compartidos representa al mecanismo que permite la sincronización implementado por el *Synchronization Manager* y el patrón Emulación representaría el mecanismo por el cual ante un requerimiento de acceso a un archivo de red o un servicio de red como el de impresión, estando la red no disponible, encamina el mismo hacia el sistema local, dado la apariencia de disponibilidad de conexión.

Un posible esquema de la arquitectura utilizando los patrones mencionados seria el siguiente.



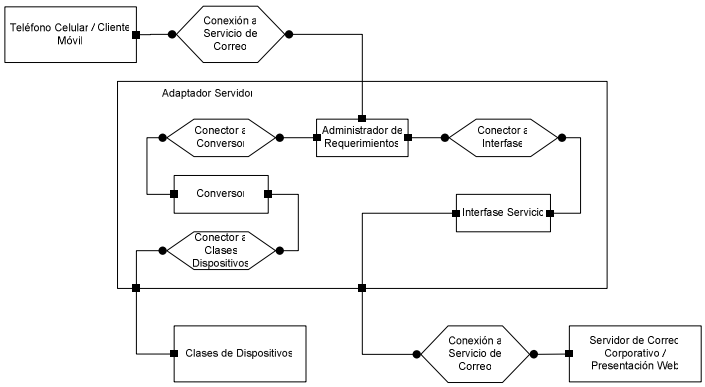
En esta arquitectura el conector *Offline Files* es el que controla, ante un requerimiento del cliente móvil, si el mismo se encamina hacia el sistema local o el sistema remoto en función de la conexión de red. Así mismo es el que implementa la funcionalidad de sincronización entre los dos sistemas.

6.4 DESCRIBIENDO UNA APLICACIÓN CLIENTE DE CORREO PARA TELÉFONOS CELULARES

Supóngase el escenario donde se quiere proveer a los empleados de una determinada empresa, acceso a sus mails corporativos a través de sus teléfonos celulares. El servidor de correo si bien puede manejar los protocoles POP3 y SMTP, también tiene la posibilidad de presentar un acceso tipo Web. Los teléfonos celulares provistos solo manejan el protocolo WAP.

Una vez resueltos los temas de networking y seguridad, el problema principal es el de resolver la adaptación del acceso provisto por el servidor de correo en HTML, a un formato WAP. La solución a esta problemática es descripta a través del patrón “Adaptador Servidor”, el cual propone la utilización de un componente que se encargue de la transformación sin tener que realizar cambios en servidor Web. En este caso se supone que el servidor de correo es un producto cerrado provisto por un tercero que no puede ser modificado en su funcionalidad.

Una posible implementación de la arquitectura propuesta es la siguiente.



Debe observarse que en este caso la adaptación es para un solo tipo de dispositivo, y al utilizar este diseño se esta dejando la posibilidad de extender la funcionalidad a un mayor numero de tipos de dispositivos.

Se puede trabajar también sobre el cliente móvil de forma de proveerlo de un mecanismo que permita al usuario el ingreso de texto eficientemente. Se asume que el usuario no se limitará solo a la consulta de sus mail, sino que pretenderá responderlos o generar nuevos. Ante esta situación se podría utilizar una arquitectura como la propuesta por el patrón de interfase “Entrada Rápida”. En este caso el componente interprete podría ser implementado como lo esta en la solución conocida como T9 basada en el teclado telefónico, o como un modulo de reconocimiento de voz. En esta última alternativa se debería evaluar el poder de cómputos del dispositivo móvil, en este caso el teléfono celular.

Luego de identificar los diferentes componentes de una posible solución, el arquitecto podrá decidir entre la compra de productos comerciales que realicen estas funcionalidades o el desarrollo de una aplicación a medida que se ajuste a la situación descripta. Esta decisión ya será función del presupuesto disponible o del tiempo y recursos con los que cuenta.

6.5 RESUMEN CAPITULO

En este capitulo se analizaron varias aplicaciones comerciales con el objeto de mostrar como los patrones ayudan a identificar las posibles arquitecturas de las mismas. En el primer caso, de una descripción de arquitectura lo suficientemente detallada, se realizo una abstracción de la misma para finalmente mostrar que era simplemente la implementación de un patrón específico. En el siguiente caso, de la descripción de la funcionalidad de un servicio, se llego a plantear una posible arquitectura del mismo basada en los patrones presentados.

Por ultimo desde un escenario ficticio donde se tienen una serie de requerimientos, se llegó a proponer una arquitectura que cubra los objetivos planteados.

CAPITULO 7

CONCLUSIONES Y TRABAJO FUTURO

7.1 CONCLUSIONES

En los últimos años los dispositivos móviles han evolucionado de tal manera que hay quienes afirman, se esta en presencia de una nueva era en el área de la computación. Los desarrollos de software para escenarios que incluyen participación de dispositivos móviles, así como para aquellos que fueron concebidos exclusivamente para permitir la movilidad, proponen una serie de nuevos desafíos. Estos desafíos pueden ser afrontados desde cero, o tratar de aplicar la experiencia que se ha adquirido en otros dominios del desarrollo de software. Este enfoque si bien discutible, puede evitar situaciones donde sin saberlo se este reinventando la rueda.

En esta tesis se buscó extender el concepto de patrones, el cual ha sido considerado uno de los aportes mas importantes de estos últimos tiempos en materia de desarrollo de software, a un dominio relativamente nuevo como es el caso de las aplicaciones móviles. Si bien no es una idea completamente innovadora, el concepto de patrones ya se ha aplicado en varias áreas del desarrollo de software, es indudable el beneficio de los mismos.

A partir del estudio de la aplicaciones móviles en general y el análisis de implementación comerciales, se identificaron una serie de patrones arquitectónicos los cuales fueron clasificados según diferentes aspectos propios del dominio, para luego finalmente conformar con estos, un catalogo de patrones.

El hecho de trabajar a niveles de arquitectura de la aplicación permite tener una visión global donde se pueden dejar de lado detalles por ejemplo de implementación, que en un primer momento complicarían la compresión del funcionamiento global. Por otra parte, a partir de una aplicación que se conoce su funcionalidad, resulta más sencillo dilucidar su arquitectura a nivel de componentes y conectores, que aproximar su diseño a nivel de clases y objetos. Sobre todo si no se dispone de la documentación de la misma. Una aplicación cualquiera puede cubrir los requerimientos planteados para esta, con varios diseños alternativos, por lo que resultaría menos probable que el diseño imaginado a nivel de clases y objetos sea el que realmente se utilizo al momento de crearla. Por otro lado si bien puede ocurrir algo similar con la arquitectura, al tratar con niveles de abstracción superiores la probabilidad de éxito es mayor.

Contar con un catalogo de patrones, aun estando incompleto, puede significar mejoras en aspectos como:

- *Aprendizaje del dominio.* Los desarrolladores que recién comienzan cuentan con un conjunto de conceptos que les facilita la compresión de las características de este tipo de aplicaciones.
- *Transferencia de conocimiento.* Las soluciones propuestas por los patrones pueden ser transferidas de proyectos en proyectos.
- *Tiempos de desarrollos.* Se tiene una mejora, ya sea al reutilizar las soluciones descriptas, o al evitar soluciones erróneas
- *Diseños.* Al contar con la base proporcionada por lo patrones resulta mas sencillo concentrase en problemas de ordenes superiores.
- *Descripción de aplicaciones.* Las abstracciones propuestas en los patrones asisten al diseñador tanto en el desarrollo de nuevas aplicaciones como en la compresión de aplicaciones existentes.

- *Lenguaje común.* Este catalogo podría como cualquier otro catalogo de patrones contribuir a la formación de un lenguaje común de diseño de alto nivel.

El catalogo presentado debería completarse con trabajos posteriores, donde se identifiquen los patrones que seguramente surgirán al aumentar la cantidad de desarrollos en el dominio de las aplicaciones móviles.

7.2 TRABAJO FUTURO

En este trabajo se trato el dominio de las aplicaciones móviles en forma general, tratando no hacer diferencias entre los distintos tipos de aplicaciones que pueden plantearse. El espíritu fue el de tratar los temas desde un alto nivel de abstracción, con el objeto de obtener una herramienta de carácter general. En trabajos futuros, además de resultar interesante completar el catálogo con nuevos patrones, seria interesante tratar de focalizar en los distintos subdominios de las aplicaciones móviles, y profundizar en cada uno de ellos. A modo de ejemplo, el desarrollo de aplicaciones *context-aware* para teléfonos celulares, es un área relativamente nueva donde el tratar de introducir herramientas de la Ingeniería de Software, posiblemente se traduzca en mejoras del proceso de desarrollo y por ende en mejorar la calidad de los productos. En muy pocos años el avance tecnológico que ha sufrido la telefonía celular ha abierto la puerta al desarrollo de aplicaciones cada vez más complejas, y a la vez, la dinámica del negocio produce que el ciclo de vida de una aplicación sea relativamente corto. Este escenario propone un entorno atractivo para implementar herramientas que han sido exitosas en otros dominios, o tal vez generar nuevas.

BIBLIOGRAFÍA

[ACKL96] P.S.C. Alencar, D.D. Cowan, Thomas Kunz, and C.J.P. Lucena - ***A Formal Architectural Design Patterns-Based Approach to Software Understanding***. Proceedings of the 4th International Workshop on Program Comprehension (WPC '96). March 1996

[App00] B. Appleton - ***Patterns and Software: Essential Concepts and Terminology.*** Synthesis of material from many knowledgeable and well respected members of the patterns community. February 2000.

http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html

[Arc01] ArcStream Solutions, Inc. - ***A Primer on Mobile Application Security***. White Paper ArcStream Solutions. June 2001

[Bri04] D. Britton Johnston, PeerDirect Corporation - ***Mobilized Software - Component and Schema Management***. Mobilized Software Web Book Chapter 1. 2004 freeMARKETpress.

http://www.mobilizedsoftware.com/webbooks/mss2/chpt1.jhtml

[Bur03] P. Burris, Appergy – ***Mobilized software - Mobile Application Architecture Patterns: An Introduction***. Mobilized Software Web Book Introduction. 2003 freeMARKETpress.

http://www.mobilizedsoftware.com/webbooks/mss2/intro.jhtml

[CD03] K C. Cousins, D. Robey - ***Patterns of Use within Nomadic Computing Environments: An Agency Perspective on Access*** -- *Anytime, Anywhere*. Workshop on Ubiqutous Computing Environment. Weatherhead School of Management Case Western Reserve University, Cleveland, OH USA. October 24-26, 2003

[Che02] S. Cheng, D. Garlan, B. Schmerl, J. P. Sousa, B. Spitznagel, P. Steenkiste, N. Hu - ***Software Architecture-based Adaptation for Pervasive Systems.*** International Conference on Architecture of Computing Systems (ARCS'02): Trends in Network and Pervasive Computing, April 8-11, 2002.

[Chu04] E. S. Chung, J. I. Hong, J. Lin, M. K. Prabaker, J. A. Landay, A. L. Liu - ***Development and Evaluation of Emerging Design Patterns for Ubiquitous Computing.*** Symposium on Designing Interactive Systems. Proceedings of the 2004 conference on Designing interactive systems: processes, practices, methods, and techniques

[CK00] G. Chen, D. Kotz – ***A Survey of Context-Aware Mobile Computing Research.*** Technical Report: TR2000-381. Dartmouth College Hanover, NH, USA. Year of Publication: 2000.

http://elans.cse.msu.edu/ni/restrict/ChenKotz2000.pdf

[Cop96] J. O. Coplien, Bell Laboratories, The Hillside Group - ***Software Patterns***. Published in 1996.

http://users.rcn.com/jcoplien/Patterns/WhitePaper/

[CYS05] J. Chamberlain, S. Yan, W. Sent - ***WebSphere Everyplace Mobile Portal Version 5 Development and Design*** - IBM RedBook. Publish Date: 29 March 2005.

http://www.redbooks.ibm.com/abstracts/redp3942.html

[DA01] A. K. Dey, Gregory D. Abowd – ***A Conceptual Framework and Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications.*** Anchor article of a special issue on context-aware computing in the Human-Computer Interaction (HCI) Journal, Volume 16 (2-4), 2001, pp. 97-166.

[DA99] A. K. Dey, G. D. Abowd - ***Towards a Better Understanding of Context and Context-Awareness.*** Panel at the 1st International Symposium on Handheld and Ubiquitous Computing (HUC '99). Panelists are: Peter Brown (University of Kent at Canterbury), Nigel Davies (University of Lancaster), Pete Steggles (AT&T Laboratories, Cambridge) and Mark Smith (HP Laboratories, Palo Alto). September 27-29, 1999. pp. 304-307

[Dey00] A. K. Dey - ***Providing Architectural Support for Building Context-Aware Applications.*** PhD thesis, College of Computing, Georgia Institute of Technology, December 2000.

http://www.cc.gatech.edu/fce/contexttoolkit/

[Dia04] M. D. Díaz - ***How to develop a software architecture: pattern languages.*** Technical Article. April 2004.

http://www.moisesdaniel.com/wri/htdsalpEn.pdf

[DR01] L. Davis, R. Gamble – *Conflict Patterns:* ***Toward identifying Suitable Middleware.*** Conference on Information Reuse and Integration, Las Vegas, NV, 2001.

[DSAF99] A. K. Dey, D. Salber, G. D. Abowd, M. Futakawa - ***An Architecture To Support Context-Aware Applications.*** GVU Technical Report GIT-GVU-99-23. Submitted to the 12th Annual ACM Symposium on User Interface Software and Technology (UIST '99), June 1999.

[FC04] P. Fahy, S. Clarke - ***CASS - Middleware for Mobile Context-Aware Applications.*** MobiSys 2004 Workshop on Context Awareness, Hyatt Harborside in Boston, Massachusetts, USA. June 6, 2004

[Fow03] M. Fowler - ***Who Needs an Architect?.*** IEEE Software. Published by the IEEE Computer Society. August 2003

[Gal01] M. Galic, J. Edling, C. Hisler, P. Holm, W. King - ***Access Integration Pattern Using IBM WebSphere Portal Server*** - IBM RedBook. Publish Date: 15 November 2001.

http://publib-b.boulder.ibm.com/Redbooks.nsf/RedpieceAbstracts/sg246267.html

[Gar01] D. Garlan - ***Next generation software architectures: Recent research and future directions***. Presentación, Columbia University, Enero de 2001.

[GHJV94] E. Gamma, R. Helm, R. Johnson, J. Vlissides, - ***Design Patterns. Elements of Reusable Object-Oriented Software.*** Addison-Wesley. Published October 1994

[Gir05] F. Girardin – ***Building a Mobile Locative, and Collaborative Application.*** Swiss Federal Institute of Technology in Lausanne. February 2005.

http://www.girardin.org/fabien/catchbob/catchbob_postmortem.pdf

[GS95] D. Garlan, M. Shaw - ***An Introduction to Software Architecture.*** Carnegie Mellon University Technical Report CMU-CS-94-166, January 1994. Reprinted in "CMIS 460: Software Design and Development Faculty Course Guide", University of Maryland, Office of Instructional Development, Summer 1995

[HIR02] K. Henricksen, J. Indulska, A. Rakotonirainy - ***Modeling Context Information in Pervasive Computing Systems.*** First International Conference on Pervasive Computing, Pervasive'2002, August 2002 LNCS(2414), pages 167-180, Zurich.

[Hof03] T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann - ***Context-Awareness on Mobile Devices - The Hydrogen Approach***. hicss, p. 292a, 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 9, 2003.

[Hur03] S. V. Hurtado Gil - ***Representación de la arquitectura de software usando UML.*** S & T: Sistemas y Telemática. Publicación semestral de la Facultad de Ingeniería de la Universidad Icesi. Numero 1 – Enero/Junio 2003

[Int03] Intel Corporation - ***Intel®Mobile Application Architecture Guide.*** Intel®Software Network. Technical Article. 2003.

http://cache-www.intel.com/cd/00/00/06/18/61807_61807.pdf

[JSED03] H. Javahery, A. Seffah, D. Engelberg, D.Sinnig - ***HCI Patterns in the Reengineering of Multiple User Interfaces.*** Human-Centered Software Engineering Group. Department of Computer Science, Concordia University. Quebec, Canada. Marzo 2003

[Kak03] M. Kakihara - ***Emerging Work Practices of ICT - Enable Mobile Professionals*** – Doctoral Thesis. Department of Information Systems. London School of Economics and Political Science. University of London. October 2003

[KB99] T. Kunz, J. P. Black - ***An Architecture for Adaptive Mobile Applications.*** Proceedings of Wireless 99, the 11th International Conference on Wireless Communications, Calgary, Alberta, Canada, July 1999.

[KCL03] P. Kovari, A. Barbosa Coqueiro, L. Liguori - ***Patterns: Pervasive Portals, Patterns for e-business Series*** - IBM RedBook. Publish Date: 09 September 2003.

http://www.redbooks.ibm.com/abstracts/SG246876.html

[KDEF05] P. Kovari, P. Dermody, D. Ehrle, S. Fassmann - ***Patterns: Pervasive and Rich Device Access Solutions*** - IBM RedBook. Publish Date: 24 February 2005.

http://www.redbooks.ibm.com/redpieces/abstracts/sg246315.html

[KR03] A. Kameas, D. Ringas - ***GAS: an Architectural style for ubiquitous computing that treats everyday objects as Communicating Tangible Components.*** First IEEE International Conference on Pervasive Computing and Communications (PerCom'03). March 23-26, 2003, Fort Worth, Texas, USA.

[Kur04] A. Kureshy, MSDN Microsoft Corporation – ***Architecting Disconnected Mobile Applications Using a Service Oriented Architecture.*** MSDN Library. Technical Articles. September 2004

[Kur95] P. Kruchten - ***Architectural Blueprints—The "4+1" View Model of Software Architecture.*** Paper published in IEEE Software 12 (6) November 1995, pp. 42-50

[Lar03] G. Larman - ***UML y Patrones. Introducción al análisis y diseño orientado a objetos.*** Prentice Hall. Segunda edición en español, 2003.

[Las03] A. Lasso, MSDN Microsoft Corporation - ***Arquitectura de Software.*** MTJ.NET Revista en línea de la Comunidad de desarrolladores de Microsoft en español. Abril 2003.

[LB03] J.A. Landay, G. Borriello - ***Design Patterns for Ubiquitous Computing.*** University of California, Berkeley. University of Washington. August 2003

[LSD02] H. Lei, D. M. Sow, J. S. Davis II - ***The Design and Applications of a Context Service.*** ACM SIGMOBILE Mobile Computing and Communications Review, v.6 n.4, p.45-55, October 2002

[Mar01] C. A. Marcos – ***Patrones de Diseño como Entidades de Primera Clase.*** Tesis Doctoral. Universidad Nacional del Cenro de la Provincia de Buenos Aires. Facultad de Ciencias Exactas. Doctorado en Ciencias de la computación. Tandil, Marzo 2001

[Mar02] M. Pérez Mariñán - ***Los peligros de los patrones de diseño.*** Articulo de difusión de conocimiento. Diciembre 2002.

http://www.javahispano.org/articles.article.action?id=74

[MV04] M. McNett, G. M. Voelker - ***Access and Mobility of Wireless PDA Users.*** Technical Report CS2004-0780, Department of Computer Science and Engineering, University of California, San Diego, Feb. 2004.

[OMA02] Open Mobile Alliance. SyncML White Paper - ***Building an Industry-Wide Mobile Data Synchronization Protocol***. 2002.

http://www.openmobilealliance.org/syncml/technology.html

[Pat04] Y. Patel, Sand Hill Systems - ***Mobilized Software - Electronic Forms Application Architecture for Mobile Solutions.*** Mobilized Software Web Book Chapter 3. 2004 freeMARKETpress.

http://www.mobilizedsoftware.com/webbooks/mss2/chpt3.jhtml

[PW92] D. E. Perry, A. L. Wolf - ***Foundations for the Study of Software Architecture.*** ACM SIGSOFT Software Engineering Notes, 17:40--52, October 1992.

http://citeseer.ist.psu.edu/perry92foundation.html

[Reg04] O. Rege, C. Kleisath, iAnywhere Solutions, Inc., A Sybase Company – ***Mobilized Software - Mobile Database Management and Synchronization.*** Mobilized Software Web Book Chapter 2. 2004 freeMARKETpress.

http://www.mobilizedsoftware.com/webbooks/mss2/chpt2.jhtml

[Rey04] C. Reynoso, MSDN Microsoft Corporation - ***Introducción a la Arquitectura de Software.*** Versión 1.0 Marzo 2004.

http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.asp

[RK04] C. Reynoso, N. Kicillof, MSDN Microsoft Corporation – ***Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.*** Versión 1.0 Marzo 2004.

http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/style.asp

[Rot01] J. Roth - ***Patterns of Mobile Interaction.*** Proceedings of Mobile HCI 2001: Third International Workshop on Human Computer Interaction with Mobile Devices, M. D. Dunlop and S. A. Brewster (Eds), IHM-HCI 2001 Lille, France, 10. September 2001, 53-58

[Rot02] J. Roth - ***Seven Challenges for Developers of Mobile Groupware.*** Workshop " Mobile Ad Hoc Collaboration", CHI 2002, Minneapolis, April 22, 2002

[RR04] W. A. Risi, G. Rossi – ***An Architectural pattern catalogue for mobile web information systems.*** International Journal of Mobile Communications (IJMC) Volume 2 - Issue 3 – 2004

[SC96] M. Shaw y P. Clements - ***A field guide to Boxology: Preliminary classification of architectural styles for software systems***. Documento de Computer Science Department and Software Engineering Institute, Carnegie Mellon University. Abril de 1996. Publicado en Proceedings of the 21st International Computer Software and Applications Conference, 1997.

[SD05] G. Lapid Shafridi, D. Doce, Microsoft Corporation – ***Multi-User Considerations in Data Synchronization for SQL Server 2005 Mobile Edition 3.0.*** MSDN Library. Technical Articles. March 2005.

[SD96] M. Shaw, D. Garlan - ***Formulations and Formalisms in Software Architecture.*** Invited for special volume of Lecture Notes in Computer Science, Computer Science Today: Recent Trends and Developments, Jan van Leeuwen (Ed), Springer-Verlag 1996, pp. 307-323.

[Sha01] M. Shaw - ***The Coming-of-Age of Software Architecture Research.*** Proceedings of the 23rd International Conference on Software Engineering, Toronto, Canada, IEEE Computer Society, 2001, pp. 656-664a

[Wei03] A. Weilenmann – ***Doing Mobility,*** Doctoral dissertation. School of Economics and Commercial Law. Göteborg University. July 2003

[Wel03] M. van Welie - ***Pattern Languages in Interaction Design: Structure and Organization.*** Proceedings of Interact '03, 1-5 September, Zürich, Switserland, Eds: Rauterberg, Menozzi, Wesson, p527-534, ISBN 1-58603-363-8, IOS Press, Amsterdam, The Netherlands

[Win01] T. Winograd - ***Architectures for Context***. Human-Computer-Interaction, 16(2), December 2001.

[ZoP03] Palm Source – ***Zen of Palm: Designing Products for Palm OS.*** Document Number 3100-002-HW. June 13, 2003.

http://www.palmos.com/dev/support/docs/

GLOSARIO

Bluetooth

Tecnología inalámbrica que opera en banda de 2.4 GHz (donde no se necesita licencia). Se trata de una tecnología pensada para la creación de redes de ámbito personal (de cobertura reducida, normalmente de unos diez metros). Las redes se suelen construir en modo "ad-hoc" utilizando dispositivos heterogéneos como teléfonos móviles, dispositivos manuales ("handhelds") y computadoras portátiles. A diferencia de otras tecnologías inalámbricas como Wi-Fi, Bluetooth ofrece perfiles de servicios más detallados; por ejemplo un perfil para actuar como un servidor de ficheros basado en FTP, para la difusión de ficheros ("file pushing"), para el transporte de voz, para la emulación de línea serie y muchos más.

cHTML *(Compact HTML)*

Es un subconjunto de HTML el cual fue diseñado para pequeños dispositivos. Las mayores características de HTML que fueron excluidas del cHTML son: Imágenes JPEG, Tablas, mapeo de imágenes, Múltiples tipos de letras y estilos, color e imágenes background, entre otras.

CODA

CODA es un FileSystem distribuido que tuvo sus orígenes en AFS2. Posee varias características que son deseables para un Network FileSystems. Actualmente, CODA provee los siguientes:

- Permite la operación en forma "desconectada" para computación móvil
- Se encuentra disponible libremente bajo una licencia "liberal"
- Alta performance a través de una cache persistente del lado del cliente
- Replicación de Servers
- Modelo seguro para autenticación, encriptación y control de acceso
- Operación continua aún durante fallas parciales de la red
- Adaptación de ancho de banda
- Buena escalabilidad
- Semántica bien definida para "*sharing*", aun en presencia de fallas de red

DECT (*Digital Enhanced Cordless Telecommunications*)

DECT define una tecnología de acceso radio para comunicaciones inalámbricas. Conceptualmente, da lugar a sistemas de comunicaciones sin hilos full-dúplex similares a los "celulares", estando la principal diferencia en que DECT está optimizado para coberturas locales o restringidas con alta densidad de tráfico.

Principales Características

1. Calidad: la utilización de técnicas de modulación ADPCM (Modulación por pulsos codificados diferencial adaptativa) a 32 Kbit/s para digitalizar la voz aseguran un alto nivel de calidad en las comunicaciones, al menos tan bueno como a través de un teléfono cableado.
2. Capacidad: La combinación de las técnicas TDMA/TDD/CDCS posibilitan un tratamiento de las interferencias y las colisiones muy eficaz que da lugar a una capacidad nominal de 10000 erlangs/Km2, muy superior a cualquier otra tecnología celular.
3. Seguridad: En el proceso de codificación se incluyen algoritmos de protección de la información, lo que junto con la técnica de CDCS hace que la escucha indeseada sea virtualmente imposible.
4. CDCS: la tecnología CDCS (selección dinámica continua de canal) implica que el equipo terminal es quien elige el canal radio y la ventana de tiempo sobre la que

realizar la comunicación en base a una monitorización periódica de las portadoras y ventanas que recibe. Esto significa que una misma comunicación cambia constantemente de "lugar" en el protocolo, y a criterio del equipo terminal. Las consecuencias fundamentales son que el handover es muy rápido, imperceptible para el usuario, y que todas las estaciones de cobertura del sistema transmiten a las mismas frecuencias, es decir, no se requiere planificación previa de frecuencias.

5. Perfiles: el estándar incluye una serie de perfiles de interconexión que posibilitan el desarrollo de aplicaciones y la integración con redes. Dichos perfiles son adicionales a la especificación radio y se encuentran en distinto grado de maduración en el proceso ETSI. Cabe destacar el perfil denominado GAP (Generic Access Profile) que garantizará la total compatibilidad entre terminales de distintos fabricantes, de manera similar a lo que sucede en las redes celulares.

ESN *(Electronic Serial Number)*

Se trata de una numero identificador univoco embebido en un teléfono inalámbrico, por el fabricante para evitar fraudes.

GPS (*Global Positioning System*)

Sistema de Posicionamiento Global originalmente llamado NAVSTAR, es un Sistema Global de Navegación por Satélite (GNSS) el cual que permite determinar en todo el mundo la posición de una persona, un vehículo o una nave, con una desviación de cuatro metros. El sistema fue desarrollado e instalado, y actualmente es operado, por el Departamento de Defensa de los Estados Unidos.

El GPS funciona mediante una red de satélites que se encuentran orbitando alrededor de la tierra. Cuando se desea determinar la posición, el aparato que se utiliza para ello localiza automáticamente como mínimo cuatro satélites de la red, de los que recibe unas señales indicando la posición y el reloj de cada uno de ellos. En base a estas señales, el aparato sincroniza el reloj del GPS y calcula el retraso de las señales, es decir, la distancia al satélite. Por "triangulación" calcula la posición en que éste se encuentra. La triangulación consiste en averiguar el ángulo de cada una de las tres señales respecto al punto de medición. Conocidos los tres ángulos se determina fácilmente la propia posición relativa respecto a los tres satélites. Conociendo además las coordenadas o posición de cada uno de ellos por la señal que emiten, se obtiene la posición absoluta o las coordenadas reales del punto de medición. También se consigue una exactitud extrema en el reloj del GPS, similar a la de los relojes atómicos que desde tierra sincronizan a los satélites.

GSM (*Global System for Mobile communications*)

Es un sistema digital de telefonía móvil que es ampliamente utilizado en Europa y en otros países del mundo. GSM utiliza una variación del acceso múltiple por división de tiempo (TDMA) y es la más utilizada de las tres tecnologías actuales de telefonía inalámbrica (TDMA, GSM y CDMA). GSM digitaliza y comprime voz y datos, y después los envía en un canal junto con otras dos series de datos del usuario en particular. Opera en las bandas de frecuencia de 900MHz, 1800MHz y 1900MHz. GSM es el estándar de telefonía inalámbrica por excelencia en Europa. Tiene en la actualidad más de 500 millones de usuarios en todo el mundo y está disponible en más de 120 países. Ya que varios operadores de GSM tienen acuerdos de Roaming con otros operadores, los usuarios frecuentemente continúan utilizando sus teléfonos cuando han viajado a otros países.

HotSync

Método de intercambio o respaldo de información entre una PDA y una computadora de escritorio a través de una conexión serial o USB

HTML *(Hypertext Markup Language)*

Lenguaje utilizado para la creación de documentos de hipertexto e hipermedia. Es el estándar usado en el World Wide Web. El hipertexto hace referencia a la capacidad de navegación de una página a otra. Las etiquetas hacen referencia a las instrucciones para realizar una determinada acción.

HTTP *(Hypertext Transport Protocol)*

Protocolo para transferir archivos o documentos hipertexto a través de la red. Se basa en una arquitectura cliente/servidor.

JDBC *(Java Database Connectivity)*

Serie de funciones que permite que programas en Java ejecuten comandos SQL, permitiendo esto que los programas puedan usar bases de datos en ese esquema. Dado que casi todos los sistemas de administración de bases de datos relacionales soportan SQL, y que Java corre en la mayoría de las plataformas, JDBC hace posible escribir una sola aplicación de base de datos que pueda correr en diferentes plataformas e interactuar con distintos DBMS.

LBS *(Location Based Services)*

Servicios que integran información sobre la ubicación o posición de un dispositivo móvil con otras fuentes de información para proveer un valor agregado al usuario.

MAN *(Metropolitan Area Network)*

Redes inalámbricas que cubren desde decenas hasta miles de kilómetros. Entre las mismas se pueden enumerar:

- Telefonía celular analógica y celular
- Radiolocalización de dos vías (pagers)
- Radio enlaces terrestres de microondas
- Laser/infrarrojo
- WLL (Wireless Local Loop)
- LMDS/MMDS
- Comunicaciones por satélite

MVC *(Model-View-Controller)*

El patrón Modelo - Vista - Controlador fue introducido inicialmente en la comunidad de desarrolladores de Smalltalk-80. MVC divide una aplicación interactiva en 3 áreas: procesamiento, salida y entrada. Para esto, utiliza las siguientes abstracciones:

Modelo: Encapsula los datos y las funcionalidades. Es independiente de cualquier representación de salida y/o comportamiento de entrada.

Vista: Muestra la información al usuario. Obtiene los datos del modelo. Pueden existir múltiples vistas del modelo. Cada vista tiene asociado un componente controlador.

Controlador: Reciben las entradas, usualmente como eventos que codifican los movimientos o pulsación de botones del ratón, pulsaciones de teclas, etc. Los eventos son traducidos a solicitudes de servicio para el modelo o la vista. El usuario interactúa con el sistema a través de los controladores.

OBEX (*Object Exchange*)

OBEX es un protocolo de sesión que puede ser descripto como un protocolo binario HTTP. OBEX esta optimizado para ad-hoc wireless links y puede ser utilizado para el intercambio de todo tipo de objetos como archivos, fotos, *calendar entries* (vCal) y *business cards* (vCard).

OBEX fue especificado por la IrDATM (*Infrared Data Association*), y si bien el protocolo es muy bueno para conexiones infrarrojas, no se encuentra limitado solamente a ellas. De hecho, OBEX es tan flexible como para correr sobre transportes como TCP/IP o Bluetooth. Así mismo OBEX puede ser llamado IrOBEX cuando es usado sobre un medio infrarrojo.

ODBC *(Open Database Connectivity)*

Estándar de acceso a Bases de Datos cuyo objetivo es hacer posible el acceso a cualquier dato de cualquier aplicación, sin importar qué Sistema Gestor de Bases de Datos (DBMS por sus siglas en Ingles) almacene los datos. ODBC logra esto al insertar una capa intermedia llamada manejador de Bases de Datos, entre la aplicación y el DBMS.

PAC *(Presentation-Abstraction-Control)*

Corresponde a un patrón arquitectónico que define una estructura para software interactivo como una jerarquía de agentes cooperantes. Cada agente es responsable de un aspecto específico de funcionalidad de la aplicación y consiste de tres componentes: Presentación, Abstracción, y Control. Esta subdivisión separa la interacción hombre-máquina de la funcionalidad y de la comunicación con otros agentes.

PalmVNC

PalmVNC es un cliente para la familia de handhelps de Palm, que provee acceso remoto al desktop PC, Mac, o UNIX workstation sobre una conexión normal TCP/IP.

PAN *(Personal Area Network)*

Nueva categoría en redes inalámbricas que cubre distancias cortas y cerradas. Algunas de estas tecnologías son Bluetooth, 802.15 y HomeRF.

PDA *(Personal Digital Assistant)*

Siglas con las que se denominan a las computadoras de mano o *handhelds*. También conocidas como agendas electrónicas, pese a tener muchas más funciones que la de una simple agenda.

PEBBLES

El proyecto Pebbles explora como dispositivos *handheld*, tales como PDAs (*Personal Digital Assistants*) incluyendo dispositivos que corren PalmOS o Pocket PCs, y teléfonos móviles, pueden ser usados cuando se comunican con una computadora personal (PC), con otros PDAs, y con otros dispositivos computarizados como teléfonos, radios, hornos a microondas, automóviles, entre otros.

RDBMS *(Relational Database Management System)*

Sistema Administrador de Bases de Datos Relacionales

Roaming

Concepto utilizado en comunicaciones inalámbricas que está relacionado con la capacidad de un dispositivo para moverse de una zona de cobertura a otra, sin la interrupción del servicio ni pérdidas de conectividad.

SDK *(Software Development Kit)*

Conjunto de aplicaciones para desarrollar programas en un determinado lenguaje o para un determinado entorno

SMS (*Short Message Service*)

Servicio de mensajería para teléfonos celulares. Permite enviar a un celular un mensaje de hasta 160 caracteres. Este servicio fue habilitado inicialmente en Europa, actualmente se haya globalmente difundido.

SQL *(Structured Query Language)*

Lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas.

SSL *(Secure Sockets Layer)*

Protocolo que permite un intercambio de información entre el servidor web y el navegador del usuario (cliente) de forma segura. El objetivo de esta tecnología es permitir que sólo el usuario autorizado pueda efectuar las operaciones. Mientras el protocolo HTTP utiliza un formato de dirección que empieza por http://, las direcciones en el protocolo SSL empiezan por https://.

SyncML

Constituye un *Framework* para sincronización de datos en escenarios móviles. SyncML fue recientemente consolidado al proyecto OMA (*Open Mobile Alliance*), el cual es impulsado por Ericsson, IBM, Lotus, Motorola, Nokia, entre otros. Más información puede ser encontrada en http://www.syncml.org

UMTS (*Universal Mobile Telecommunications System*)

La tecnología UMTS es el sistema de telecomunicaciones móviles de tercera generación, que evoluciona desde GSM pasando por GPRS.
El principal avance es la tecnología WCDMA (*Wide Code Division Multiple Access*) heredada de la tecnología militar, a diferencia de GSM y GPRS que utilizan una mezcla de FDMA (*Frecuency Division Multiple Access*) y TDMA (*Time Division Multiple Access*). La principal ventaja de WCDMA consiste en que la señal se expande en frecuencia gracias a un *wide code* que sólo conocen el emisor y el receptor. Esta original forma de modulación tiene numerosas ventajas como:

- Altas velocidades de transmisión de hasta 2 Mbps,
- Alta seguridad y confidencialidad
- Alta resistencia a las interferencias.
- Posibilidad de trabajar con dos antenas simultáneamente debido a que siempre se usa todo el espectro y lo importante es la secuencia de salto, lo que facilita el *handover* (proceso de traspaso de la señal de una antena a otra)

URL *(Universal Resource Locator)*

Sistema unificado de identificación de recursos en la red. Es el modo estándar de proporcionar la dirección de cualquier recurso en Internet.

VPN *(Virtual Private Network)*

Tecnología que permite la transmisión de información privada sobre redes de uso público de manera segura, utilizando conexiones virtuales.

WAP (*Wireless Application Protocol*)

WAP es un protocolo basado en los estándares de Internet que ha sido desarrollado para permitir a teléfonos celulares navegar a través de Internet. Con la tecnología WAP se pretende que desde cualquier teléfono celular WAP se pueda acceder a la información que hay en Internet así como realizar operaciones de comercio electrónico.

WAP es una serie de tecnologías que consisten en: WML, que es el lenguaje de etiquetas, WMLScript es un lenguaje de *script*, lo que vendría a ser JavaScript y el Wireless Telephony Application Interface (WTAI)

Las características principales de WML son:

- Soporte para imágenes y texto, con posibilidad de texto con formato.
- Tarjetas agrupadas en barajas. Una pagina WML es como una página HTML en la que hay una serie de cartas, al conjunto de estas cartas se les suele llamar baraja.
- Posibilidad de navegar entre cartas y barajas de la misma forma que se navega entre páginas Web.
- Manejo de variables y formularios para el intercambio de información entre el teléfono celular y el servidor.
- WML es un lenguaje de marcas similar al HTML. WML es compatible con XML 1.0. Las páginas WML son llamadas barajas ya que están compuestas por cartas, un navegador WAP, solo puede mostrar un carta al mismo tiempo.

WI-FI *(Wireless Fidelity)*

Término acuñado por la *Wi-Fi Alliance* que se utiliza normalmente cuando se hace referencia a algún tipo de red 802.11, ya sea 802.11b, 802.11a, banda dual, etc. El estándar 802.11 se refiere a redes inalámbricas que operan en 2.4GHz.

WLAN *(Wireless Local Area Network)*

Red de área local a la que un usuario puede tener acceso a través de una conexión inalámbrica.

WML *(Wireless Markup Language)*

Versión reducida del código HTML que utiliza la tecnología WAP para visualizar contenido Web a través de un teléfono móvil.

WMLscript

WMLScript es un lenguaje que puede ser considerado como un dialecto de JavaScript. Lo que Javascript es para HTML en el entorno Web, lo es WMLScript para WML en el entorno WAP.

La intención de los creadores de WMLScript es dotar de un poco de inteligencia propia al teléfono en el que se reciben las páginas WML, de modo que pueda descargar al servidor WAP de realizar más tareas que las estrictamente imprescindibles; a la vez,

evitar acabar con el poco ancho de banda que actualmente posee este sistema, dado que se suprimen algunas conexiones entre el teléfono y el servidor.
La potencia de WMLScript es, hasta este momento, bastante limitada, pero permite al teléfono realizar cálculos, personalizar páginas WML o validar los datos introducidos por el usuario antes de enviarlos al servidor WAP, por citar algunas utilidades.
WMLScript es un lenguaje débilmente tipificado y que no acepta objetos. Se compila en el servidor a un código intermedio denominado bytecode que es lo que se envía al teléfono cuando este lo requiere.
A diferencia de JavaScript, el código en WMLScript no se encuentra en las páginas WML, sino que se encuentra en ficheros externos que, como se ha dicho, deben ser compilados.
Un fichero con código fuente WMLScript tendrá extensión .wmls, mientras que una vez compilado llevará extensión .wmlsc

XML *(Extensible Markup Language)*

Lenguaje universal de marcado para documentos estructurados y datos en la Web, más amplio, más rico y más dinámico que HTML. No solo es un lenguaje de marcado, sino también un metalenguaje que permite describir otros lenguajes de marcado. Permite el uso ilimitado de los tipos de datos que pueden utilizarse en Internet, lo cual resuelve los problemas que surgen entre las organizaciones que deben intercambiar datos procedentes de estándares distintos. Más información puede encontrarse en www.xml.com

ANEXO 1

CONTEXTO Y APLICACIONES MÓVILES

Dentro del la computación móvil*, Context-aware computing* es considerado un paradigma, en el cual las aplicaciones pueden descubrir y tomar ventaja de la información contextual.

A.1.1 INTRODUCCIÓN

Antes de comenzar a desarrollar esta sección, resulta interesante presentar una serie de conceptos académicos relacionados con aplicaciones móviles que habitualmente suelen dar lugar a confusiones.

Mobile Compunting. Termino genérico que describe la aplicación de dispositivos de computación y de comunicaciones, pequeños, portátiles y de conexión inalámbrica. Esto incluye por ejemplo dispositivos como *Laptops* con tecnología WLAN, teléfonos celulares, y PDAs con alguna tecnología de comunicación inalámbrica.

Location Based Services. Se refiere a aplicaciones que proveen un servicio basado en la actual posición geográfica del usuario y su dispositivo. Por ejemplo un LBS en una red de teléfonos celulares, la posición se conoce gracias a una radiolocalización lograda por la triangulación de posiciones conocidas de antemano como la de las celdas que el Terminal del usuario esta utilizando para establecer una comunicación.

Ubiquitous computing. Mark Weiser fue quien introdujo la visión de computación ubicua. En sus palabras "*la computación ubicua fundamentalmente se caracteriza por la conexión de objetos en el mundo real con la computación*". Weiser veía la tecnología solamente como un medio para un fin y como algo que debería quedar en segundo plano para permitir al usuario concentrarse completamente en la tarea que está realizando. Utilizaba el término "computación ubicua" en un sentido más académico e idealista como una visión de tecnología discreta, centrada en la persona, mientras que la industria ha acuñado para eso el término "computación pervasiva", o ampliamente difundida (*pervasive computing*)

Pervasive computing. Es la tendencia a incrementar la ubicuidad, conectando dispositivos computacionales en el entorno, una tendencia que ha traído la convergencia de electrónica de avanzada, en particular inalámbrica, e Internet. Los dispositivos de la computación pervasiva no son computadoras personales como las habituales, son dispositivos mucho más pequeños e incluso invisibles que se encuentran embebidos en todo tipo de objetos imaginables, incluyendo automóviles, herramientas, ropas, etc., todas los cuales se encuentran interconectados a través de una red inalámbrica.

Todos estos conceptos de una forma u otra se encuentran relacionados con las aplicaciones móviles y aparecen en la literatura muchas veces como sinónimos o extremamente vinculados. Las aplicaciones móviles son sistemas desarrollados para dispositivos como los indicados en *Mobile Computing*. La mayoría de las aplicaciones móviles, por no indicar todas, se encuentran dentro de las aplicaciones que utilizan la ubicación del usuario y su dispositivo, es decir, ligadas al concepto de LBS. Y para el caso de *Ubiquitous computing* y *Pervasive computing*, la visión de Weiser se orientaba a

computación en cualquier momento y en cualquier, conceptos que derivan nuevamente a los mismos que los de aplicaciones móviles.

A continuación se presentaran los fundamentos entorno a las Aplicaciones *Context-Aware*.

A.1.2 DEFINICIÓN DE CONTEXTO

El uso del contexto es de suma importancia en las aplicaciones interactivas, particularmente en aquellas en el que el contexto del usuario varía muy rápidamente. Tal es el caso de aplicaciones móviles, donde la libertad de movimiento del usuario y su dispositivo, causan este efecto.

La primera vez que se introdujo el termino *context-aware*, fue en el trabajo de Schilit y Theimer (*Disseminating Active Map Information to Mobile Hosts*), donde se refiere a contexto como ubicación, entidades en las proximidades de personas y objetos, y los cambios de estos objetos. Brown en su trabajo (*Context-Aware Applications*), define contexto como ubicación, identidad de las persona alrededor del usuario, hora del día, temperatura, estación, etc. Ryan por su parte define el contexto como ubicación del usuario, ambiente, identidad y tiempo. Dey en su trabajo, *Context-Aware Computing: The CyberDesk Project*, enumeras contexto como el estado emocional del usuario, su foco de atención, su ubicación y orientación, día y fecha, objetos, y personas en el ambiente del usuario.

Estas definiciones del contexto son difíciles de aplicar cuando se trata de delimitar que tipo de información puede o no ser considerada cómo contextual. En todas ellas los aspectos importantes del contexto son: donde esta el usuario, con quien esta y cuales son los recursos disponibles en su proximidad.

Por ultimo la definición que actualmente ha sido ampliamente aceptada [DA99] es la siguiente:

> *Contexto es cualquier información que puede se usada para caracterizar la situación de una entidad. Donde una entidad es una persona, lugar u objeto que es relevante para la interacción entre un usuario y una aplicación, incluyendo al usuario y la aplicación.*

Esta definición hace mas sencillo, al desarrollador de una aplicación, enumerar el contexto de una aplicación, para un escenario dado. Si una pieza de información puede ser utilizada para caracterizar la situación de un participante de una interacción, entonces esta información es contexto.

A.1.3 CATEGORÍAS DE CONTEXTO

Existen ciertos tipos de contextos que son en práctica más importantes que otros, estos son: Ubicación, Identidad, Actividad y Tiempo. Comparadas con las categorías propuestas por Schilit (donde estas, con quien estas y que objetos hay a tu alrededor), estas caracterizan la situación de manera mas completa, ya que incluyen la información de actividad y tiempo.

Con estos tipos de contexto no solo se responden las cuestiones de quien, que, cuando y donde, sino que también se indican otras fuentes de información contextual. Por ejemplo si se tiene la identidad del usuario, seguramente se tendrá información secundaria como e-mail, número telefónico, edad, lista de contactos, etc. En otras palabras se tiene un sistema de dos niveles en el cual las categorías Ubicación, Identidad, Actividad y Tiempo determinan el primer nivel y el segundo nivel esta

compuesto por cualquier otro tipo de información contextual. Las piezas de información secundaria tienen como característica que puede ubicar en un espacio de información cuyo índice es uno o varios de los tipos de contexto del primer nivel. Por ejemplo los números telefónicos de los usuarios pueden encontrarse indexados por la identidad del usuario.

Esta caracterización ayuda a los diseñadores a seleccionar el contexto a utilizar en sus aplicaciones, estructurar la forma de uso del contexto, y explorar otros contextos relevantes. Las cuatro primeras piezas de contextos indican los tipos de información necesarias para caracterizar una situación y el uso de las mismas provee una manera de usar y organizar el contexto.

A.1.4 DEFINICIÓN DE CONTEXT-AWARE COMPUTING

El concepto de *Context-Aware Computing*, fue discutido por primera vez en 1994 por Schilit y Theimer, donde se lo consideraba como software que se adaptaba de acuerdo a donde se lo usaba, a que objetos y personas se encontraban en las proximidades y los cambios que tenían esos objetos en el tiempo. Sin embargo es comúnmente aceptado el hecho de que la primera investigación llevada a cabo en esta área se corresponde al trabajo Olivetti Active Badge de 1992.

Al igual que el caso de la definición de contexto, la definición de aplicaciones *context-aware* ha tenido muchas versiones y diferentes autores. La primera definición dada por Schilit y Theimer se restringía a aplicaciones que iban desde aquellas que eran simplemente informadas sobre el contexto hasta aplicaciones que se adaptaban por si mismas al contexto.

Las definiciones de *context-aware computing* caen dentro de dos categorías: las que usan el contexto y las que se adaptan al contexto.

Dentro del primer grupo se pueden ubicar las definiciones de Hull o Pascoe quienes definen *context-aware computing* como las habilidades de los dispositivos computaciones de detectar y sensar, interpretar y responder a los aspectos del entorno del usuario y el dispositivo en si mismo. Salber define *context-aware* como la habilidad de proveer el máximo de flexibilidad a un servicio (desde el punto de vista computacional) basándose en la adquisición en tiempo real del contexto.

En el grupo de las que se “adaptan al contexto”, varios investigadores han definido las aplicaciones *context-aware* como aquellas que dinámicamente cambian o adaptan su comportamiento en función del contexto de la aplicación y del usuario. Por ejemplo Ryan las define como aquellas aplicaciones que monitorean la entrada desde los sensores del entorno y permiten a los usuarios seleccionar desde un rango de contextos físicos y lógicos de acuerdo con su actual interés o actividad. Brown define las aplicaciones *context-aware* como aquellas aplicaciones que proveen información y/o realizan acciones de acuerdo con el presente contexto del usuario el cual es detectado por sensores.

Finalmente la definición presentada por Dey y Abowd [DA99], y ampliamente aceptada es la siguiente:

> *Un sistema es context-aware si el mismo usa el contexto para proveer información relevante y/o servicio al usuario, donde la relevancia depende la tarea del usuario.*

Es de notar que esta definición es mucho más general que las anteriores, y no hace referencia al hecho de cómo detectar e interpretar el contexto. Se deja librada esta funcionalidad a que por ejemplo sea llevada a cabo por otras entidades o aplicaciones. Bajo la misma, una aplicación es considerada *context-aware* cuando de alguna manera

responde en influencia del contexto. Tampoco se hace referencia a la adquisición del contexto en tiempo real.

A.1.5 FUNCIONES CONTEXT-AWARE

Dey y Abowd realizaron una clasificación de las funciones *context-aware* que pueden ser implementadas por una aplicación de este tipo. La misma introduce tres categorías de funciones, relacionadas con la presentación de información, la ejecución de un servicio, y el almacenamiento de información contextual para que otro mas tarde pueda recuperarla.

Presentación de información y servicios. Se refiere a aplicaciones que presentan información de contexto al usuario, o utilizando el contexto presenta una apropiada selección de acciones al usuario. Como ejemplo se puede citar la aplicación para teléfonos celulares que en función de la ubicación del usuario dentro de una ciudad le presenta los locales de comida rápida más próximos.

Ejecución Automática de servicios. Describe aplicaciones que disparan un servicio, o reconfiguran su comportamiento de acuerdo al cambio de contexto del usuario. Como ejemplo, la aplicación que cambia a modo silencioso el teléfono celular, cuando el usuario ingresa a un teatro o cine, en un determinado horario.

Anexar información contextual para su posterior recuperación. Incluye aquellas aplicaciones que pueden poner en una etiqueta (virtual), datos con información relevante del contexto, dejándolas para un posterior uso. Como ejemplo se puede citar el proyecto "*Virtual Information Towers*" llevando a cabo en la Universidad de Stuttgart, donde una VIT es una estructura de información asignada a una ubicación geográfica, que tiene una cierta área de visibilidad, y que es accesible por un usuario con un dispositivo móvil que se aproxime a la misma. Es un concepto un poco más elaborado que las *stick-e note*.

La siguiente tabla muestra un análisis interesante realizado por Dey y Abowd donde se analizan aplicaciones existentes en ese momento, bajo la luz de la taxonomía de tipos de contexto y funcionalidades *context-aware*.

System Name	System Description	Context Type				Context-aware		
		A	I	L	T	P	E	T
Classroom 2000	Capture of a classroom lecture							
Cyberguide	Tour guide							
Teleport	Teleporting							
Stick-e Documents	Tour guide							
	Paging and reminders							
Reactive Room	Intelligent control of audiovisuals							
GUIDE	Tour guide							
CyberDesk	Automatic integration of user services							
Conference Assistant	Conference capture and tour guide							
Responsive Office	Office environment control							
NETMAN	Network maintenance							
Fieldwork	Fieldwork data collection							
Augment-able Reality	Virtual post-it notes							
Context Toolkit	In/Out Borrad							
	Capture of serendipitous meetings							
Active Badge	Call forwarding							

A:	Activity	**L:**	Location	**P:**	Presentation	**T:**	Tagging
I:	Identity	**T:**	Time	**E:**	automatic Execution		

Se evidencia claramente la importancia de la ubicación (*location*) en lo que se refiere a aplicaciones *context-aware*. Es habitual encontrar en la literatura una subclasificación denominada *location-aware*, que incluye a las aplicaciones que son principalmente influenciadas (usan o se adaptan) según la posición del usuario y su dispositivo. Se desprende que el hecho de hablar de ubicación conlleva el concepto de movilidad, y por ende el de aplicaciones móviles.

A.1.6 DESARROLLO DE APLICACIONES CONTEXT-AWARE.

El campo de las aplicaciones *context-aware*, como ya se ha dicho, es relativamente nuevo y se encuentra en un periodo de gran crecimiento. Desde la ingeniería de software se han realizado varios aportes, algunos de ellos tratando de utilizar las metodologías y herramientas ya existentes. A continuación se resumen dos trabajos que enfocan el desarrollo de aplicaciones *context-aware* y han tenido una indiscutible incidencia en el área.

A.1.6.1 Context Toolkit

Este enfoque constituye la tesis doctoral de Anind K. Dey [Dey00]. El mismo presenta una *Framework* para el desarrollo de aplicaciones *context-aware*. Sostiene que los desarrollos realizados hasta ese momento, son hechos a media donde se mezcla en el diseño los requerimientos propios de la aplicación junto con el manejo de sensores en forma "artesanal", donde el cambio de un sensor implica cambios en la aplicación. Su trabajo introduce el concepto de separar la aplicación de todo lo referente a la recolección, adaptación e interpretación de la información del contexto. Idealmente, propone manejar la información de contexto como si fuera la información de entrada de un usuario. Para esto realiza una importante abstracción que permitirían a los diseñadores utilizar la información de contexto sin importar como fue recolectada, estos son los denominados *Context Widgets*.

Los *Context Widgets* facilitan la separación de la semántica de la aplicación de los detalles de manejo a bajo nivel de la entrada de información de contexto. Proveen los mismos beneficios que los *widgets* GUI, por ejemplo separación de "*concerns*", reutilización, fácil acceso a la información a través de mecanismos de "*poling & notification*", etc. Encapsulan una parte del contexto y abstraen los detalles de cómo el mismo fue obtenido.

Además del importante concepto de *Widget,* introdujo dos abstracciones mas, *Interpreters*, y *Aggregators*, ya que si bien los *context widgets* son similares a los *GUI widgets*, existen características propias del contexto que necesitaban ser tenidas en cuenta.

Los *Context Interpreters* son responsables de realizar la interpretación de la información de bajo nivel del contexto entregándola en un nivel de abstracción superior. Por ejemplo un sistema de posicionamiento dentro de un edificio podría brindar coordenadas X-Y-Z, y el *Context Interpreter* devolver el piso y el numero de habitación. Nuevamente el hecho de la separación de los intérpretes de la aplicación facilita la reutilización de los mismos.

Los *Context Aggregators* realizan la agregación del contexto de las distintas entidades presentes en el entorno. En su definición de contexto, Dey expresa la necesidad de recolectar la información relevante de las entidades

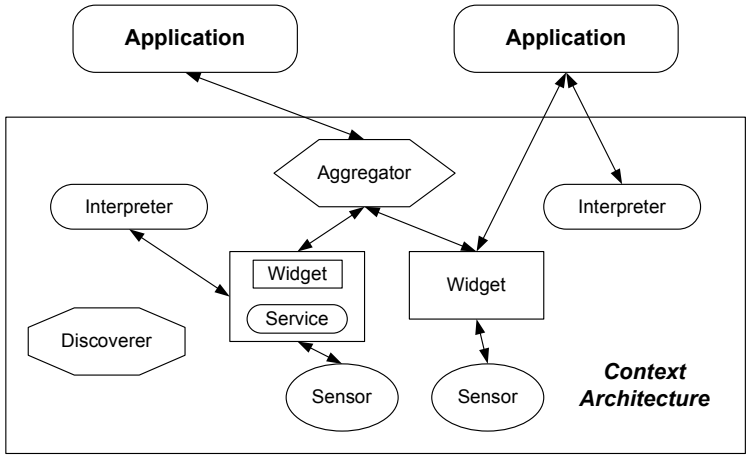
(personas, lugares y objetos) presentes en el entorno. Con solo la abstracción de los *Widgets*, una aplicación debería comunicarse con cada uno de ellos presente para caracterizar el contexto de una entidad. Esto tiene un impacto negativo en cuanto a eficiencia y mantenimiento. El *Aggregator* es una abstracción que permite a la aplicación interactuar con un único elemento, para cada entidad en la que este interesada.

Dentro del *Framework* presentado existe otra abstracción, los *Context Services*. Un *context service* es análogo a un *context widgets*, pero mientras que este último es responsable de extraer información del entorno a través de un sensor, el *context service* es responsable de actuar o cambiar la información de contexto.

Finalmente presenta un conjunto de requerimientos para una arquitectura que permita la construcción, ejecución y evolución de aplicaciones *context-aware*:

- Permitir a las aplicaciones el acceso a la información de contexto desde maquinas distribuidas de la misma manera que las mismas acceden a la información ingresada por el usuario localmente
- Soportar la ejecución sobre diferentes plataformas y el uso de diferentes lenguajes de programación.
- Soportar la interpretación de la información de contexto.
- Soportar la agregación de la información de contexto.
- Soportar la independencia y persistencia de los *Context Widgets* (vale aclarar que los *context widgets* a diferencia de los *GUI widgets*, no son instanciados y controlados por la misma aplicación)
- Soportar el almacenamiento de la información contextual histórica.
- Permitir el descubrimiento de recursos.

La siguiente figura muestra la arquitectura en bloques del *Framework* propuesto en el *Context Toolkit*.



En resumen esta arquitectura presenta varios beneficios como: la separación de *concerns* entre la aplicación y todo lo relativo a contexto, la reutilización, el hecho de ocultar a la aplicación los detalles a bajo nivel relativo a los sensores, entre otros. Como limitaciones, la arquitectura no hace referencia a la falta de confiabilidad que puede tener la información recolectada y como tratarla, como manejar la falla de componentes, y como hacer transparente la adquisición del contexto desde los componentes distribuidos (para que una

aplicación pueda hacer uso de *widgets*, *interpreters* o *aggregators* debe conocer el *hostname* y el *port* donde el componente esta siendo ejecutado).

A.1.6.2 Hydrogen Context-Framework

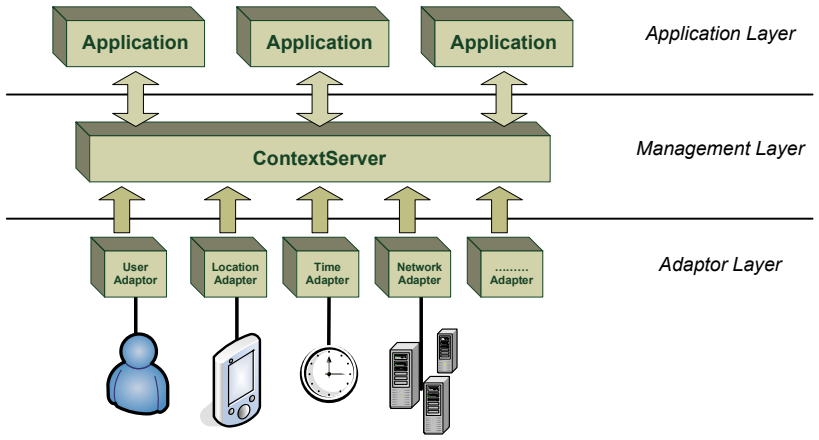
Corresponde al trabajo realizado por un grupo de investigadores austriacos sobre el desarrollo de aplicaciones *context-aware*, el cual resulta de particular interés para esta tesis ya que esta enfocado a aplicaciones para dispositivos móviles [Hof03]. Proclaman que las pocas arquitecturas que proveen soporte para aplicaciones *context-aware* no hacen referencia a los especiales requerimientos de un dispositivo móvil, en particular las limitaciones en la conexión a la red, el bajo nivel de cómputos y las características de los usuarios móviles.

Definen una serie de requerimientos para una arquitectura de un *Framework* que soporte *context-awareness* sobre dispositivos móviles:

- ***Poco Peso.*** La arquitectura debe tener en cuenta las restricciones en cuanto al limitado poder de computo de los dispositivo móviles
- ***Extensibilidad.*** Dada la limitada disponibilidad de sensores y *slots* de expansión de los dispositivos móviles, no es posible que un solo dispositivo adquiera toda la información del contexto. Por lo tanto la arquitectura debe soportar la conexión a sensores remotos.
- ***Robustez.*** La arquitectura debe ser lo suficientemente robusta como para soportar la desconexión de los sensores remotos.
- ***Meta-Información.*** Teniendo en cuenta el hecho que los sensores remotos pueden proveer información controversial, el modelo de contexto debe contener meta-información como por ejemplo: distancia del dispositivo al sensor en la cual es valida su precisión.
- ***Compartir el contexto.*** Más allá de todo lo mencionado se hace notar que cuando un dispositivo móvil adquiere el contexto nunca lo hace en forma completa. Es deseable disponer de un mecanismo para compartir el contexto adquirido con otros dispositivos.

Cabe aclara que estos requerimientos se incluyen puntos que no fueron tratados en el modelo *Context Toolkit,* por ejemplo el hecho de tener en cuenta las posibles desconexiones de los sensores y la posible inconsistencia de la información presentada por los mismos. También es de notar que tienen una fuerte componente asociada a dispositivos móviles. Por otra parte, si se mantiene el concepto de separación de *concerns*, afirmando que la aplicación no se debe entremezclar con el manejo y el almacenamiento del contexto, lo cual le permite concentrase en su propio propósito.

El modelo presentado consiste en una arquitectura de tres capas llamada *Hydrogen Context-Framework*.

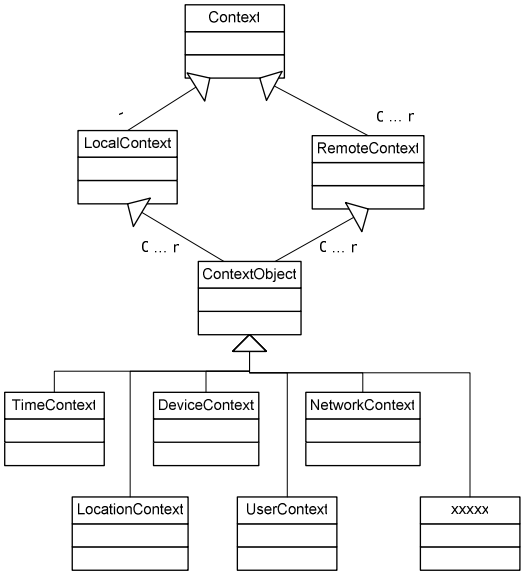


Las tres capas separan los asuntos de interactuar con los sensores físicos y, almacenamiento y mantenimiento del contexto, de la aplicación en si. El componente central es el llamado *ContextServer*, el cual almacena el contexto para que las aplicaciones puedan realizar búsquedas en el mismo.

La capa "*Adaptor*" es responsable de obtener la información desde los sensores, respecto del contexto físico, posiblemente enriquecer dicha información con información lógica del contexto y entregarla a la capa "*Management*". Por otra parte esta capa al separar la adquisición del contexto permite que el mismo pueda ser utilizado por más de una aplicación simultáneamente. Comúnmente cuando se accede a un sensor directamente desde la aplicación se lo hace en forma exclusiva a este recurso, por ejemplo una interfase serial solo puede ser accedida por una aplicación al mismo tiempo.

La capa "*Management*", a través del *ContextServe*r embebido en ella, provee métodos simples para que la aplicación pueda extraer o suscribirse al contexto. El *ContextServer* tiene la posibilidad de compartir información con otros dispositivos, de forma que aquellos que no tienen la capacidad de adquirir determinados tipos de datos, utilicen los adquiridos por otros como si fueran adquiridos por ellos mismos. A su vez tiene dos modalidades de trabajo, una "*pull*" y otra "*push*" para ofrecer la información de contexto.

Además de la arquitectura descripta, los autores proponen una arquitectura para el contexto. Básicamente distinguen en entre contexto local y contexto remoto. Sostienen que los dispositivos dentro de un rango se pueden comunicar entre si y pueden almacenar información adicional del contexto del otro que puede ser útil para su propio contexto. Las aplicaciones no solo pueden considerar el contexto del dispositivo en el que están corriendo sino que también pueden considerar el contexto de los otros dispositivos que se encuentre en comunicación con el propio. En el caso de encontrarse en otro dispositivo, la información de contexto puede ser mutuamente intercambiada, permitiendo una representación local del contexto del dispositivo remoto. A continuación se presenta un diagrama UML propuesto por los autores para la representación del contexto.



El contexto local contiene varios *ContextObjects*, los cuales tienen información contextual, provista por cada sensor que tiene anexado. Adicionalmente cada dispositivo puede almacenar contextos remotos de dispositivos que se encuentren accesibles por la red.

Como se puede ver en el diagrama el Context-Framework propuesto tiene cinco tipos de contextos (Tiempo, Dispositivo, Usuario, Ubicación y Red) pero puede ser extendido por especialización del *ContextObject*.

Hydrogen Context-Framework presenta algunas diferencias/ventajas respecto de otras arquitecturas propuestas. Las tres capas de la arquitectura se ubican dentro del mismo dispositivo, esto hace que el mismo ya no sea tan dependiente de la red, para obtener el contexto. Por otra parte ya no hay necesidad de Servidores remotos, cada dispositivo tiene un Server local lo que hace que esta arquitectura sea robusta frente a frecuentes desconexiones de red (uno de los requerimientos identificados para aplicaciones en escenarios móviles). Abandona el concepto de otros modelos de almacenar el "contexto histórico" (con el objetivo de cumplir con el requerimiento de "poco peso"). Por ultimo el "*Context Sharing*" es soportado a través de una red peer-to-peer, lo que lleva a otro motivo para que no sea necesario disponer de un Server centralizado.

A.1.7 TECNOLOGÍAS Y DISPOSITIVOS

Dos ramas tecnológicas han permitido a los usuarios obtener movilidad, la evolución de los dispositivos portátiles y el surgimiento de la comunicación inalámbrica [CK00]. La disminución de tamaño de las computadoras, sin perder o en algunos casos aumentar sus capacidades y el incremento de los vínculos inalámbricos han permitido a los usuarios acceder a su información personal, a los datos corporativos o un recurso público en cualquier momento y desde cualquier lugar.

Existen actualmente varias *wireless handheld computers* disponibles, corriendo sistemas operativos como Palm OS, Microsoft Pocket PC (Windows CE), y Symbian EPOC. Hay varias maneras en que estos dispositivos pueden conectarse sin el uso de cables:

- ***Redes inalámbricas celulares.*** Por ejemplo, Palm VII se conecta automáticamente al portal www.palm.net cada vez que su antena es desplegada. Existen varios fabricantes que producen modems celulares para puertos seriales o puertos de expansión de dispositivos portátiles. Omnisky provee un modem que puede ser utilizado en una Palm V. Qualcomm integra un dispositivo Palm y un teléfono celular en un nuevo producto pdQ SmartPhone
- ***Redes locales inalámbricas.*** Por ejemplo el Symbol PPT 2700 tiene incorporada una antena Spectrum24 que soporta IEEE 802.11, estándar para comunicaciones inalámbricas, y el ITU H.323 estándar para comunicación multimedia. Existen actualmente varios módulos de expansión que soportan IEEE 802.11 en forma de CompactFlash, PCMCIA, entre otros.
- ***Redes de área personal inalámbricas.*** La comunicación inalámbrica tipo PAN (*Personal Area Network*), o BAN (*Body Area Network*) permiten la comunicación inalámbrica entre dispositivos en un rango reducido con bajo costo y con redes de a lo sumo 8 a 16 nodos. *Bluetooth* es uno de los estándares basado en RF más prometedor en reemplazar los cables entre dispositivos. Existen actualmente disponibles dispositivos con capacidad *Bluetooth* como ser el teléfono móvil Sony Ericsson W800i.

A.1.8 DETECCIÓN DEL CONTEXTO

Para que sea posible el uso del contexto en aplicaciones, por supuesto, deben existir mecanismos que detecten el actual contexto y lo entreguen a la aplicación. A continuación se describen algunos mecanismos actualmente usados con tal propósito.

A.1.8.1 Detección de la ubicación

Dado que la ubicación es el contexto que más importantes cambios tiene cuando el usuario se mueve, un sistema confiable de seguimiento de ubicación es crítico para aplicaciones *context-aware* [CK00]. Esto es sencillo de obtener si es el usuario quien provee la información de ubicación al sistema. Técnicas comúnmente usadas para estos son las tarjetas que se deslizan frente a un sensor o un lector que huellas digitales que el usuario debe utilizar al entrar o salir de un recinto. Estos métodos requieren de la explicita cooperación del usuario y solo proveen una tosca granularidad y baja precisión (el usuario puede olvidarse informar al sistema cuando abandona un recinto). A continuación se presentan varias técnicas automáticas para obtener la ubicación.

A.1.8.1.1 Outdoors

La opción obvia para un sistema de posicionamiento *Outdoor* es el uso de GPS (*Global Positioning System*). Recientemente el gobierno de US a realizado modificaciones que permiten el uso civil de de la señal GPS con una precisión de 10 a 20 metros. Por ejemplo, los sistemas de navegación para automóviles se vieron beneficiados con esta nueva política.

Sin embargo existen otros dispositivos de bajo costo con capacidades similares la GPS, por ejemplo Bulusu, Heidemann y Estrin proponen en su trabajo "*GPS-less low-cost outdoor localization for very small devices*", una técnica de ubicación basada en conectividad. El algoritmo de localización produce una precisión de 3 metros en base a puntos de referencia conocidos y un idealizado modelo de radio.

A.1.8.1.2 Indoors

GPS no funciona *indoors* ya que la señal es muy débil y no penetra la mayoría de los edificios. Por otra parte las reflexiones producen fluctuaciones que hacen que la precisión del sistema sea inaceptable. Es un problema desafiante el construir un sistema ideal de localización *indoor* que provea información espacial de gran fino, con una alta tasa de actualización, no intrusito, barato, escalable y robusto.

La mayoría de los proyectos de investigación tienen su propio sistema ubicación. El *Olivetti Active Bagde*, *Xerox ParcTab* y el proyecto *Cyberguide* utilizan sistemas basados en infrarrojo. El Personal *Shopping Assistant* propuesto por AT&T y el *Pinger near-filed* de Hewlett-Packard, utilizan un sistema de radiofrecuencia.

Los sistemas mencionados están basados en el uso de celdas, en los que el dispositivo detecta en que celda se encuentra o el sistema determina que dispositivos se encuentran en una determinada celda. Existen otros enfoques como el sistema RADAR de *Microsoft Research* el cual utiliza una señal en la red de comunicaciones que mide para determinar la distancia entre el emisor y el receptor.

Un enfoque no celular lo constituyen los proyectos *Active Floor* y *Smart Floor*, los cuales identifican a las personas por su manera de caminar. La precisión en identificar a un usuario en movimiento es de aproximadamente el 90%. Constituye la forma menos intrusita de proveer de la información de ubicación de usuarios al sistema.

A.1.8.1.3 Híbridos

Existen algunos sistemas de ubicación basados en redes de dominio, los cuales no están específicamente dirigidos al uso *outdoor* o *indoor*. El proyecto GUIDE toma este enfoque, basándose en un sistema celular "*wireless Ethernet*", el cual se encuentra desplegado un área metropolitana.

Mas allá del uso *outdoor* o *indoor*, existen dos enfoques generales para hacer que el dispositivo móvil sea consiente de su actual ubicación. En uno el sistema sigue la ubicación de los dispositivos móviles a través del monitoreo de *beacoms* ubicados en los mismos, y el dispositivo móvil realiza una búsqueda en la base de datos central para obtener su actual ubicación. En el otro el dispositivo móvil monitorea pasivamente *beacoms* ubicados en las estaciones base de cada celda y realiza búsquedas en una base de datos local para obtener su actual ubicación. En este ultimo caso, si el dispositivo móvil solo realiza búsquedas en una base de datos local para obtener su ubicación, el mismo tiene una privacidad completa y puede elegir en dar a conocer su actual ubicación al mundo, o a un grupo por el determinado.

A.1.8.2 Detección de contexto de bajo nivel

Existen otros tipos de contexto mas allá de la ubicación, sin embargo solo algunos han sido ampliamente estudiados. A continuación se presenta una lista de los mismos:

- ***Tiempo.*** Esta información contextual no es difícil de obtener del reloj propio del dispositivo. Algunas aplicaciones permiten la incorporación de esta información a la de ubicación. A si mismo,

existen otras forma de contexto de tiempo como ser, día de la semana, día del mes, mes del año, estación, etc.

- ***Objetos Cercanos.*** Si el sistema registra la ubicación de personas y otros objetos, es sencillo de determinar cuales se encuentran en las proximidades, por un medio de una búsqueda en la base de datos de ubicaciones.
- ***Ancho de banda de la Red.*** Es un contexto computacional muy importante. Actualmente existen sistemas implementados como módulos que notifican a la aplicación ante un cambio en el ancho de banda. Trabajos mas reciente incluyen Administradores de Congestión.
- ***Orientación.*** La orientación es algo posible de medir. Un sensor simple de orientación, basado en dos *switches* de mercurio, se encuentra incorporado a la *Newton MessagePad,* lo que le permite a las aplicaciones ajustar la pantalla cuando la orientación cambia.
- ***Otros contextos de bajo nivel.*** Se trata de sensores especialmente diseñados para algunos tipos de contextos, como ser de temperatura, humedad, presión, ruido, etc.

A.1.8.3 Detección de contexto de alto nivel

En adición a la información contextual como ubicación, nivel de ruido, temperatura, también se tiene interés en información de contexto de alto nivel como ser la actividad que esta llevando a cabo el usuario. Es si embargo una gran desafió determinar el complejo contexto social. Un enfoque esta basado en una cámara y la tecnología de procesamiento de imágenes. Otro posible enfoque es el de consultar directamente la agenda del usuario para averiguar que es lo que supuestamente va a realizar, esto tiene como gran desventaja que el usuario puede olvidar completar la agenda, o directamente realizar actividades fuera de cualquier programación. Y un tercer método consiste en la utilización de de técnicas de inteligencia artificial para reconocer contextos complejos en función de sensores de bajo nivel.

A.1.9 RESUMEN ANEXO

Se presentaron los fundamentos de las aplicaciones *context-aware*, como contexto, categorías de contexto y funciones *context-aware*. Se analizaron dos trabajos representativos del área. Y por ultimo se dieron algunos detalles sobre tecnología entorno a este tipo de aplicaciones y mecanismos para la detección del contexto.

ANEXO 2

ARQUITECTURA DE SOFTWARE

Actualmente si se pretende encontrar una definición concreta de Arquitectura de Software, sus limites e incumbencias, se confronta con un sin numero de visiones diferentes, cada una de ellas con sus correspondientes fundamentos. En el presente capitulo se pretende lograr una visión general, con el objeto de introducir al lector de esta tesis en los conceptos teóricos que le sean útiles en el marco de mejorar su compresión.

A.2.1 INTRODUCCIÓN

Los comienzos de la Arquitectura de Software (AS) se remontan a la década del '60 con el trabajo de Edsger Dijkastra, pero la disciplina comienza a tomar forma mucho mas tarde con los trabajos de Dewayne Perry de AT&T Bell Laboratorios, y Alexander Wolf de Universidad de Colorado. Se puede afirmar que ellos fueron realmente los fundadores. Ya a fines del siglo XX, los mayores aportes provienen de lo que ha dado por llamarse la escuela estructuralista de Carnegie Mellon: David Garlan, Mary Shaw, Paul Clements y Robert Allen [GS95], [SD96], [Sha01].

Se trata entonces de una práctica relativamente nueva que actualmente se encuentra en un auge creativo con el desarrollo de sus técnicas en los trabajos de Rick Kazmna, Mark Klein y otros investigadores en el contexto del *Software Engineering Institute* (SEI).

Retomando el trabajo de Perry y Wolf, vale la pena transcribir algunas partes del mismo ya que puede considerarse como el primer estudio en que aparece la frase Arquitectura de Software en el sentido en que hoy se conoce.

> "*El propósito de este paper es construir el fundamento para la arquitectura de software. Primero desarrollaremos una intuición para la arquitectura de software recurriendo a diversas disciplinas arquitectónicas bien definidas. Sobre la base de esa intuición, presentamos un modelo para la arquitectura de software que consiste en tres componentes: elementos, forma y razón (rationale). Los elementos son entidades, ya sea de procesamiento, datos o conexión. La forma se define en términos de las propiedades de, y las relaciones entre, los elementos, es decir, restricciones operadas sobre ellos. La razón proporciona una base subyacente para la arquitectura en términos de las restricciones del sistema, que lo más frecuente es que se deriven de los requerimientos del sistema. Discutimos los componentes del modelo en el contexto tanto de la arquitectura como de los estilos arquitectónicos....*"

El resto del trabajo presenta una reseña de las técnicas y herramientas de diseño anteriores al mismo y una visión de cómo evolucionarían en un futuro. Afirman que se dejaría de utilizar el término diseño para pasar a usar el de arquitectura, en referencia a nociones de codificación, abstracción, estándares, y de estilos.

El dominio de la AS desde sus comienzos y hasta la fecha, fue y sigue siendo difuso dentro del marco global de la ingeniería de software y la practica puntual del diseño. Garlam y Perry, en su trabajo realizado para la IEEE en abril 1995 [GS95] enumeran lo que consideraban las futuras áreas de investigación de la AS:

- Lenguajes de descripción de arquitecturas
- Fundamentos formales (bases matemáticas, teorías de interconexión, etc.)
- Técnicas de análisis arquitectónicas
- Métodos de desarrollo basados en arquitecturas

- Recuperación y reutilización de arquitecturas
- Codificación y guía arquitectónica
- Herramientas y ambientes de diseño arquitectónico
- Estudios de casos

Clements y Northrop, en su reporte técnico "*Software architecture: An executive overview*", consideran que la reutilización es el criterio fundamental para justificar la existencia de la disciplina AS. Afirman que las decisiones de alto nivel tomadas para un miembro de una familia de programas, no necesitan ser re-validadas, re-inventadas y re-descriptas. Se plantea que el concepto de estilo plasma estas ideas, es decir la de reutilización.

En otro trabajo realizado por Clements, "*Coming attractions in Software Architecture*", define los cinco temas que considera fundamentales para la AS, a saber:

- Diseño o selección de la arquitectura: como crear o seleccionar una AS en función de los requerimientos funcionales y no funcionales.
- Representación de la arquitectura: como representar una AS para luego poder comunicarla. Incluye el hecho de seleccionar que información debe ser comunicada.
- Evaluación y análisis de la arquitectura: como analizar una AS para predecir las cualidades del sistema que describe.
- Desarrollo y evolución basados en arquitectura: como construir y mantener un sistema partiendo de su arquitectura
- Recuperación de arquitectura: como hacer para que dado un sistema ya en producción, el mismo evolucione cuando los cambios afectan su estructura. Es decir si no se dispone de la AS del sistema original, se deberá realizar primero la extracción de la misma.

En forma contemporánea surge otro gran tema, el de patrones, que se ve plasmado en dos grandes obras: la que realizó, los que luego se dieron por llamar la Banda de los Cuatro (*Design Patterns: Abstraction and Reuse of Object Oriented Design*) y la serie POSA (*Pattern-Oriented Software Architecture*). Los patrones influenciaron en la AS como disciplina y algunos de los arquitectos de esa época formaron un acercamiento a la visión de patrones, sobre todo basándose en el concepto que tanto estos como la arquitectura, presentan la idea dominante de la reutilización.

Ya en el siglo XXI la AS aparece orientada a establecer modalidades de análisis, diseño, verificación, diseño basado en escenarios, redefiniendo de esta forma las metodologías ligadas al ciclo de vida en términos arquitectónicos. Pareciera que todo lo que se ha hecho en la Ingeniería de Software debe ahora replantearse para incluir a la AS. De nuevo el origen de todas estas nuevas metodologías se centra en los trabajos generados por el SEI de *Carnegie Mellon University*.

A.2.2 DEFINICIONES

De las variadas definiciones que actualmente existen sobre Arquitectura Software, las cuales pueden encontrarse en gran parte en la compilación realizada por el SEI, se puede observar que la mayoría se circunscriben en los siguientes conceptos o visiones generales: uno enfocado en el trabajo dinámico de la AS dentro del proceso de ingeniera o el diseño (su lugar en el ciclo de vida de un producto software), otro estático dirigida a la configuración o topología de un sistema contemplado desde un alto nivel de abstracción, y por ultimo el que intenta describir la disciplina que trata los dos primeros. En otras palabras, de una forma u otra la mayoría de las definiciones actuales versan en su esencia en los mismos conceptos.

La definición que se ha acordado como oficial es la provista por el documento de IEEE STD 1471-2000, que expresa:

> "*La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.*"

Teniendo en cuenta que la AS se inscribe en el campo de la Ingeniería de Software, vale la pena contrastar sus definiciones, entonces conforme al estándar IEEE 610.12.1990, se entiende como Ingeniería de Software a:

> *"La aplicación de una estrategia sistemática, disciplinada y cuantificable al desarrollo, aplicación y mantenimiento del software; esto es, la aplicación de la ingeniería al software."*

Se puede notar que la noción clave de la arquitectura es la organización (un concepto cualitativo o estructural), mientras que la ingeniería tiende fundamentalmente a ser una noción sistemática susceptible de cuantificarse.

De todas maneras mas allá de la gran cantidad de definiciones, existe un acuerdo general sobre que la AS, se refiere a la estructura a grandes rasgos del sistema, estructura consistente en componentes y relaciones entre ellos. Estas cuestiones estructurales se vinculan con el diseño de software que se manifiesta en las etapas tempranas del proceso de creación de un sistema; pero este diseño ocurre a un nivel más abstracto que el de los algoritmos y las estructuras de datos.

Existen dentro del área un amplio numero de investigadores que han realizado aportes significativos, pero podría afirmarse que las contribuciones mas importantes provinieron del SEI.

En la década del 90 surge la programación basada en componentes, que en su momento de mayor auge impulso a arquitectos como Paul Clements, quien presentó una reconocida definición:

> "La AS es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones."

También en esa misma época se desataca el trabajo realizado por Mary Shaw y David Garlan [GS95], quienes abordaron la tarea de explicar las diferencias entre definiciones en función de distintos modelos (estructural, *framework*, dinámico, proceso y funcional) que aparecían con cierta regularidad en los trabajos de AS. La clasificación que lograron es la siguiente:

> 1) **Modelos estructurales:** Es el modelo de vistas más común de arquitecturas. Sostienen que la AS está compuesta por componentes, conexiones entre ellos y otros aspectos tales como configuración, estilo, restricciones, semántica, análisis, propiedades, racionalizaciones, requerimientos, necesidades de los participantes. Los modelos estructurales están comúnmente soportados por el desarrollo de lenguajes de descripción arquitectónica (ADLs). Los ejemplos incluyen Aesop, C2, Darwin, UniCon, and Wright., etc.
>
> 2) **Modelos de *framework*:** Son similares a la vista estructural, pero su énfasis primario radica en la estructura coherente del sistema completo, en vez de

concentrarse en su composición. Los modelos de framework a menudo se refieren a dominios o clases de problemas específicos. El trabajo que ejemplifica esta variante incluye arquitecturas de software específicas de dominios, como CORBA, o modelos basados en CORBA, o repositorios de componentes específicos, como PRISM.

3) **Modelos dinámicos:** Los modelos dinámicos son complementarios a los modelos estructurales y *frameworks*. Comúnmente suelen referirse a los cambios en la configuración del sistema, o a la dinámica involucrada en el progreso de la computación, tales como valores cambiantes de datos. Ejemplos serian: Chemical Abstract Machine, Archetype, Rex, Conic, Darwin, y Rapide

4) **Modelos de proceso:** Se concentran en la construcción de la arquitectura, y en los pasos o procesos involucrados en esa construcción. En esta perspectiva, la arquitectura es el resultado de seguir un argumento de proceso. Esta vista se ejemplifica con el actual trabajo sobre programación de procesos para derivar arquitecturas.

5) **Modelos funcionales:** Una minoría considera la arquitectura como un conjunto de componentes funcionales, organizados en capas que proporcionan servicios hacia arriba. Es tal vez útil pensar en esta visión como un *framework* particular.

Mas allá de las discrepancias entre las diversas definiciones, es común entre todos los autores el concepto de la AS como un punto de vista que concierne a un alto nivel de abstracción. Donde el concepto de abstracción es tomado en el sentido de extraer las propiedades esenciales, o identificar los aspectos importantes, o examinar selectivamente ciertos aspectos de un problema, posponiendo o ignorando los detalles memos sustanciales o irrelevantes.

A modo de resumen y para cerrar la idea sobre como será tratado el concepto de Arquitectura de Software en la presente tesis, se asumirá que el mismo hace referencia a la estructura de elementos que componen un sistema, sus interacciones y como estas se ajustan para alcanzar los objetivos del mismo.

A.2.3 ESTILOS ARQUITECTÓNICOS

Un estilo es un concepto descriptivo que define a grandes rasgos una forma de articulación u organización arquitectónica. Los estilos solo se manifiestan en la arquitectura teórica descriptiva de alto nivel de abstracción. El conjunto de estilos constituyen un catalogo con las formas básicas posibles de estructuras de software, a partir del cual las formas complejas se logran mediante las composición de los estilos fundamentales [RK04]. Como se verá, no existe un catalogo único, sino varias versiones que en su mayoría se superponen en su contenido.

Los estilos conjugan componentes, conectores, configuraciones y restricciones, donde al estipular los conectores como elemento de juicio de primera clase, el concepto de estilo se ubica en una posición que ni el método de modelado orientado a objetos o el UML cubren satisfactoriamente. Si bien una forma describir un estilo seria a través del uso de lenguaje natural o de diagramas, la mejor manera de hacerlo es a través de un lenguaje de descripción arquitectónica o a través de lenguajes formales de especificación.

A diferencia de los patrones de diseño, los estilos se ordenan en seis o siete clases fundamentales y unos veinte ejemplares como máximo.

Vale la pena aclara la relación entre estilos, patrones de diseño y patrones de arquitectura. Los patrones de diseño de software buscan codificar y hacer reutilizables un conjunto de principios a fin de diseñar aplicaciones de alta calidad. Se aplican en principio solo en la fase de diseño, aunque la comunidad ha comenzado a definir y aplicar patrones a otras etapas del proceso de desarrollo. Los estilos se han aplicado en la fase de análisis arquitectónico en términos de patrones de arquitectura. Sin bien los patrones de arquitectura son similares a los de diseño, los primeros se concentran en la estructura de alto nivel del sistema. Algunos autores sostienen que los patrones arquitectónicos son virtualmente lo mismo que los estilos, pero si bien existen convergencias entre ambos conceptos, los patrones se refieren más a prácticas de reutilización y los estilos conciernen a teorías sobre la estructura de los sistemas. El concepto de patrones y sus diferentes usos es tratado mas profundamente en el Anexo 1 de la presente tesis.

De los diferentes catálogos de estilos que se pueden encontrar en la literatura, se observa que los mismos se subdividen o articulan en 5 ó 6 clases. A modo de ejemplo se presenta el presentado por David Garlan en su trabajo "*Next generation software architectures: Recent research and future directions*" [Gar01]:

1. Flujo de datos
   - Secuencial en lotes
   - Red de flujo de datos (tuberías y filtros)
   - Bucle de control cerrado

2. Llamada y Retorno
   - Programa principal / subrutinas
   - Ocultamiento de información (ADT, objeto, cliente/servidor elemental)

3. Procesos interactivos
   - Procesos comunicantes
   - Sistemas de eventos (invocación implícita, eventos puros)

4. Repositorio Orientado a Datos
   - Bases de datos transaccionales (cliente/servidor genuino)
   - Pizarra
   - Compilador moderno

5. Datos Compartidos
   - Documentos compuestos
   - Hipertexto
   - Fortran COMMON
   - Procesos LW

6. Jerárquicos
   - En capas (intérpretes)

En este trabajo el autor señala los inconvenientes de la vida real que comúnmente va en contra de las taxonomías:

- Los estilos casi siempre se usan combinados
- Cada capa o componente puede ser internamente de un estilo diferente al de la totalidad
- Varios estilos se encuentran ligados a dominios específicos, o a líneas de producto particulares.

Es de destacar, que la caracterización de los estilos no constituye un reflejo pormenorizado de su estructura, mas bien son una visión rápida y genérica que acentúa

sus valores esenciales y sus características distintivas. Comúnmente su representación grafica es sumamente informal y minimalista. Shaw y Clements en su trabajo "*A field guide to Boxology: Preliminary classification of architectural styles for software systems*" [SC96], han llamado a la misma "*boxology*".

A continuación se detallan algunos de los estilos citados:

Arquitectura de Pizarra o Repositorio

En este estilo arquitectónico existen dos componentes principales: una estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre él. A su vez se han definido dos subcategorías:

1. Los tipos de transacciones en el flujo de entrada definen los procesos a ejecutar, entonces el repositorio puede ser una base de datos tradicional.
2. El estado actual de la estructura de datos dispara los procesos a ejecutar, entonces el repositorio es lo que se llama una pizarra pura o tablero de control.

Habitualmente un sistema de pizarra se implementa para resolver problemas en los cuales las entidades en forma individual son incapaces de alcanzar una solución, o para los que no existe una solución analítica, o para los que si existe pero no es viable dada la dimensión del espacio de búsqueda. Todo modelo de este tipo consiste en tres partes, a saber:

- Fuentes de conocimiento, necesarias para resolver el problema
- Una pizarra que representa el estado actual de la resolución del problema
- Una estrategia que regula el orden en el que operan las fuentes.

El comportamiento en general seria el siguiente: en una primera instancia se establece el problema en la pizarra. Las fuentes tratan de resolverlo cambiando el estado. Entre las fuentes la única forma de comunicación es a través de la pizarra. Finalmente, si de la cooperación resulta una solución adecuada, la misma es presentada en la pizarra.

Este estilo de arquitectura se ha utilizado en aplicaciones que requieren complejas interpretaciones de proceso de señales, o en sistemas que involucren acceso compartido a datos, con agentes sin acoplamiento.

En el paper "*Architectures for Context*" de Terry Winograd [Win01], se presenta este estilo de arquitectura como modelo para administrar el contexto, realizando un comparación del misma con otros dos modelos posibles para este dominio. Plantea que el estilo de pizarra fue desarrollado en el contexto de la inteligencia artificial, donde cada componente o agente tenía información parcial y una nueva fuente podía ser agregada fácilmente. El hecho de un bajo acoplamiento era pagado con la escasa eficiencia de comunicación. Cada comunicación requiere dos saltos y usa en general una estructura de mensajes no optimizada. Los beneficios son la facilidad de configuración y su robustez. En su investigación sobre "*Interactive Workspaces*" donde exploraron la integración de múltiples dispositivos para múltiples usuarios compartiendo un mismo espacio físico (*iRoom*), utiliza una arquitectura de comunicación tipo pizarra para soportar "*context-aware computing*"

Arquitecturas Orientadas a Objetos

En la caracterización clásica de Garlam y Shaw, los objetos representan una clase de componentes los cuales son responsables de preservar la

integridad de su propia representación. Se desataca el hecho que la representación interna de un objeto no es accesible desde otros objetos y no se establece como una cuestión definitoria el principio de herencia.

Las características principales de este estilo arquitectónico son las siguientes:

- Los componentes se basan en principios de OO: encapsulamiento, herencia y polimorfismo. Son asimismo las entidades de modelado, diseño e implementación.
- En general la distribución de objetos es transparente y no tiene importancia si los objetos son locales o remotos. Un ejemplo de este estilo seria CORBA (Common Object Request Broker Architecture), en el cual un Object Request Broker media las interacciones entre objetos clientes y objetos servidores en ambientes distribuidos.
- Los componentes (objetos), interactúan entre si a través de invocaciones de funciones y procedimientos.

Se pude considerar este estilo como perteneciente a una familia arquitectónica más amplia que algunos autores llaman Arquitecturas de Llamada-y-Retorno

En cuanto a ventajas se puede mencionar entre otras el hecho que se pueda modificar la implementación de un objeto sin afectar a sus clientes, y que un objeto pueda ser reutilizado en el entorno de desarrollo.

Como desventaja, el principal problema del estilo es el hecho que para poder interactuar con otro objeto se debe conocer su identidad, es decir se debe conocer los métodos o procedimientos que implementa.

En los estudios arquitectónicos de estilos, el modelo de objetos aparece como relativamente subordinado en relación con la importancia que ha tenido en el ámbito de la ingeniería de software. Se podría decir que no es un estilo relevante en cuanto a estilos de implementación.

Arquitectura Basada en Componentes

El objetivo de la tecnología de componentes software es construir aplicaciones complejas mediante ensamblado de módulos (componentes) que han sido previamente diseñados a fin de ser reutilizados en múltiples aplicaciones. La ingeniería de programación que sigue esta estrategia de diseño se le conoce por CBSE (*Component-based Software Engineering*).

La arquitectura basada en componentes sostiene que una aplicación consiste en muchos componentes prefabricados que se ensamblan entre sí para proporcionar los servicios que se necesitan en la misma.

En la tecnología de componentes la interfaz constituye el elemento básico de interconexión. Cada componente debe describir de forma completa las interfaces que ofrece, así como las interfaces que requiere para su operación. Y debe operar correctamente con independencia de los mecanismos internos que utilice para soportar la funcionalidad de la interfaz.

Las características más relevantes son la modularidad, la reutilización y componibilidad y en todos ellos coincide con la tecnología orientada a objetos de la que se puede considerar una evolución. Sin embargo, en la tecnología basada en componentes también se requiere robustez ya que los componentes han de operar en entornos mucho más heterogéneos y diversos.

Existen entornos normalizados para el desarrollo de componentes, un ejemplo de estos seria el modelo COM (Componet Object Model). El mismo define un estándar para objetos y la intercomunicación entre ellos. Toda comunicación se realiza a través de operaciones que son proporcionadas dentro de interfaces.

A.2.4 LENGUAJES DE DESCRIPCIÓN ARQUITECTÓNICA

Los lenguajes de descripción de Arquitecturas (ADL), ocupan una parte importante del trabajo arquitectónico desde la fundación de la disciplina. Se trata de un conjunto de variado nivel de rigurosidad, casi todas ellas de extracción académica, que surgiendo en los comienzos de la década del ´90.

Si bien los ADLs se encuentran difundidos en el ámbito universitario, no ocurre lo mismo en la industria, donde los mismos son muy pocos conocidos y menos aun llegan a aplicarse en proyectos. En la práctica las configuraciones arquitectónicas se suelen describir por medio de cajas, líneas y comentarios o anotaciones realizados en el mismo esquema.

Los ADLs permiten modelar una arquitectura mucho antes que se realice la programación de las aplicaciones que la conforman, analizar su adecuación, determinar sus puntos críticos, sus riesgos y hasta poder simular su comportamiento, todo esto a un alto nivel de abstracción. Se utilizan para satisfacer requerimientos descriptivos que las herramientas basadas en objetos en general y UML en particular, no satisfacen completamente. Para citar algunas: el soporte de los modelos OO solo proporcionan una única forma primitiva de interconexión, la invocación de un método, esto dificulta modelar formas de interacción mas complejas o diferentes, tampoco soportan la definición de familias de sistemas o estilos, no existen recursos sintácticos para caracterizar clases de sistemas en términos de las restricciones de diseño que debería observar cada miembro de la familia, no brindan soporte formal para caracterizar y analizar propiedades no funcionales, entre otras.

Si bien no existe hoy en día una definición única de que es un ADL, comúnmente se acepta que un ADL debe proporcionar un modelo explicito de componentes, conectores y sus respectivas configuraciones. Se considera deseable que un ADL suministre soporte de herramientas para el desarrollo de soluciones basadas en las arquitecturas y su posterior evolución. A continuación se presentan los elementos constitutivos primarios que la mayoría de los ADLs provee:

- **Componentes:** Representan los elementos computacionales primarios de un sistema. Se corresponden con las cajas de las descripciones del tipo caja-y-línea. Los componentes pueden presentar varias interfaces, las que definen puntos de interacción con otros componentes o con su entorno. Ejemplos: servidores, bases de datos, filtros, etc.
- **Conectores:** Representan las interacciones entre los componentes. Se corresponden a las líneas de las descripciones del tipo caja-y-línea. Sobre los conectores se pueden definir los roles de los componentes que participan en la interacción. Ejemplo: la conexión de un cliente con un Proxy.
- **Configuraciones o Sistemas:** Se permiten estructuras jerárquicas es decir la representación de una determinada configuración o sistema puede ser tomado como un componente a ser utilizado en un diagrama de mayor nivel de abstracción.
- **Propiedades:** Representan información semántica sobre un sistema más allá de su estructura. Distintos ADLs ponen énfasis en diferentes clases de propiedades, pero todos tienen alguna forma de definir propiedades no funcionales, o pueden admitir herramientas complementarias para analizarlas y determinar, por ejemplo, cuestiones de seguridad, escalabilidad, etc.
- **Restricciones:** Se corresponden también con información semántica pero relacionada con restricciones de diseño que se deberán tener en cuenta. Por ejemplo la máxima cantidad de conexiones simultaneas a un servicio.

- **Estilos:** Capacidad para soportar los estilos mas difundidos

En la actualidad algunos de los ADLs conocidos son los siguientes: ADML, Aesop, Artek, C2, CHAM, Darwin, Jacal, LILEANNA, MetaH/AADL, Rapide, UML, UniCon, Weaves y Wright

A.2.5 CONCEPTO DE VISTAS

Una vista es un subconjunto resultante de practicar una selección o abstracción sobre una realidad, desde una perspectiva determinada.

La idea de observar un sistema complejo desde múltiples punto de vista no es original de la AS. La motivación para introducir múltiples vistas se basa en la separación de incumbencias. Las vistas se introdujeron como una herramienta conceptual para poder manejar la complejidad de artefactos tales como especificaciones de requerimientos o modelos de diseño. A modo de ejemplo para el caso de especificación de requerimientos, vale citar la separación propuesta por Leite del Baseline de Requisitos en cinco vistas: vista del modelo léxico, vista del modelo básico, vista del modelo de escenarios, vista de hipertexto y vista de configuración.

Retomando el tema Vistas desde la AS, no existe un límite para el número de vistas posibles, ni un procedimiento para establecer lo que una vista debe o no abstraer. El estándar IEEE 1471 si bien no delimita el número posible de vistas, señala lineamientos para su construcción.

No existe un documento único que especifique cuales son las vistas que se deben presentar en la descripción de la arquitectura de un sistema, existen al respecto diferentes enfoques. El cuadro siguiente muestra un enfoque presentado en el libro "*El lenguaje unificado de modelado*" de Rumbaugh, Jacobson, y Booch, que gira en torno a UML, y en el que se definen ocho vistas.

Área	Vista	Diagramas	Conceptos Principales
Estructural	Vista estática	Diagrama de clases	Clase, asociación, generalización, dependencia, realización, interfaz
	Vista de casos de uso	Diagramas de casos de uso	Caso de uso, actor, asociación, extensión, inclusión, generalización de casos de uso
	Vista de implementación	Diagrama de componentes	Componente, interfaz, dependencia, realización
	Vista de despliegue	Diagrama de despliegue	Nodo, componente, dependencia, localización
Dinámica	Vista de máquinas de estados	Diagrama de estados	Estado, evento, transición, acción
	Vista de actividad	Diagrama de actividad	Estado, actividad, transición de terminación, división, unión
	Vista de interacción	Diagrama de secuencia	Interacción, objeto, mensaje, activación
		Diagrama de colaboración	Colaboración, interacción, rol de colaboración, mensaje
Gestión del modelo	Vista de gestión del modelo	Diagrama de clases	Paquete, subsistema, modelo

Otro enfoque ampliamente difundido es el que expresa que las posibles vistas de la arquitectura de una aplicación serían:

- *Vista Conceptual*: Describe la arquitectura en términos de los elementos computacionales de alto nivel, los cuales abarcan la solución. En esta vista el arquitecto organiza los elementos computacionales propios del dominio específico, y la interacción entre ellos, construyendo de forma que se cubran los requerimientos de la aplicación.
- *Vista Lógica*: Muestra los componentes principales de diseño y sus relaciones de forma independiente de los detalles técnicos y de cómo la funcionalidad será implementada. Los arquitectos crean modelos de diseño de la aplicación, los cuales son vistas lógicas del modelo funcional y que describen la solución. El esquema de capas comúnmente utilizado en la actualidad en la versión de 3 capas, se corresponde con esta vista.
- *Vista Física*: Esta vista plasma la distribución del procesamiento entre los distintos equipos que conforman la solución, incluyendo los servicios y procesos de base. Los elementos definidos en la vista lógica se corresponden con componentes de software o de hardware que definen más precisamente como se ejecutará la solución.
- *Vista de Implementación*: Esta vista describe como se implementaran los componentes lógicos y físicos agrupándolos en subsistemas organizados en capas y jerarquías.

Mas allá de las diferentes alternativas o enfoques existentes sobre las vistas que la arquitectura de un sistema debe presentar, es útil comprender que estas son enfoques de la misma cuestión y la utilización de una u otra debe ser inclinada en gran parte por quien será el destinatario de la misma. En otras palabras, seguramente una vista que puede ser útil para los desarrolladores, puede no serlo para el usuario final del sistema.

A.2.6 ARQUITECTURA VS. DISEÑO

En el ambiente puramente académico, se sostiene que la AS difiere ampliamente del mero diseño. De todas maneras como en muchos otros ámbitos de la informática en general, los limites son difusos y se prestan a varias interpretaciones. A modo de ejemplo, Taylor y Medvidovic, presentan lo que ellos consideran las diferentes posturas de ese momento:

- Una postura afirma que la Arquitectura y el Diseño son la misma cosa
- Otra que, la AS se encuentra en un nivel de abstracción superior al del diseño, y no es más que otro artefacto del proceso de desarrollo de software.
- Y una tercera que la AS difiere del diseño, aludiendo que corresponde a una nueva disciplina.

Estos autores consideran que la segunda postura es la que mas se ajusta a la verdad, afirman que arquitectura y el diseño bogan por el mismo propósito, pero que AS tiene el foco en la estructura del sistema y sus interconexiones, mientras que el diseño se concentran en abstracciones de más bajo nivel, mas cercanas al código.

Shaw y Garlam sostienen que la AS es el primer paso en la producción de un diseño de software, dentro de una secuencia formada por:

1. Etapa de Arquitectura: desde los requerimientos especificados para el sistema, se asocian los componentes que deberá tener el mismo.

2. Etapa de diseño de código: los componentes de esta etapa son las primitivas del lenguaje de programación. Se definen los algoritmos y las estructuras de datos
3. Etapa de diseño ejecutable: similar a la etapa anterior pero con un nivel de detalle mucho más profundo. Son tratadas cuestiones como asignación de memoria, formato de datos, etc.

Clements por su parte sostiene que el diseño basado en AS representa un nuevo paradigma que difieren en forma sustancial a sus predecesores y que deberán desarrollarse nuevas metodologías sustentadas en el mismo o ajustar las actuales para que puedan contemplarlo

Como se pueden ver existen diferentes posturas sobre la relación entre la arquitectura y el diseño, pero la mayoría coincide que la AS se desarrolla en un nivel de abstracción mucho elevado que el diseño. Afirman que la arquitectura es de "grano grueso", mientras que el diseño ahonda en detalles mucho más finos.

A.2.7 APORTE DE LA ARQUITECTURA DE SOFTWARE

A pesar del poco tiempo trascurrido desde la aparición de la disciplina, la AS ha demostrado ser de gran utilidad en un gran numero de escenarios. Nadie se atrevería a esta altura de los acontecimientos en dudar sobre la necesidad de una visión arquitectónica.

A continuación se presenta un resumen de los aportes que diversos entendidos consideran que ha traído la AS:

- *Comunicación Mutua*: dado el alto nivel de abstracción de la AS, la mayoría de los participantes pueden utilizar el mismo como base para crear un entendimiento mutuo, y mejorar de esta forma la comunicación.
- *Decisiones tempranas de diseño*: las decisiones de diseño que se plantean en al AS son tomadas en el comienzo del proceso de desarrollo, y las mismas afectaran al sistema en todo su ciclo de vida, hasta en la etapa misma de mantenimiento.
- *Restricciones constructivas*: Una descripción arquitectónica proporciona guías para el desarrollo, al indicar los componentes y las dependencias entre ellos, que fijaran limites, identificaran las principales interfaces, describirán las distintas interacciones, entre otros puntos.
- *Reutilización*: varios autores justifican la existencia de la disciplina basándose en el concepto de reutilización de modelos arquitectónicos a través de distintas aplicaciones.
- *Evolución*: La AS suele exponer de una forma mas clara los ejes sobre los que puede evolucionar un sistema.
- *Análisis*: Las descripciones arquitectónicas aportan nuevas oportunidades para el análisis. Los modelos permiten ahondar en temas como verificación de consistencias, conformidad con atributos de calidad, análisis de dependencias, entre otros.
- *Administración*: Los proyectos exitosos consideran una arquitectura viable como un logro clave del proceso de desarrollo. La evaluación crítica de una arquitectura conduce a una mejor compresión de los requerimientos, las estrategias de implementación y los riesgos potenciales.

A.2.8 RESUMEN ANEXO

En esta sección se realizó una presentación de la Arquitectura de Software como disciplina, comenzado con una breve recopilación histórica para luego describir los conceptos fundamentales en torno a la misma como Vistas, Estilos y Lenguajes de descripción Arquitectónica. Así mismo se la comparó a grandes rasgos con el diseño de software. Por ultimo se listaron los que la comunidad considera, son los aportes mas significativos de la disciplina.

ANEXO 3

PATRONES

Para cualquier ciencia o disciplina de ingeniería, es fundamental el hecho de contar con un vocabulario para expresar sus conceptos, y de un lenguaje que los relacione. Los patrones dentro de la comunidad de software, han cubierto esta necesidad al crear un cuerpo de literatura que ayuda a los desarrolladores a resolver los problemas recurrentes encontrados en el proceso de desarrollo. Los patrones forman un lenguaje común que permite comunicar la experiencia adquirida sobre problemas y las soluciones que fueron aplicadas con éxito.

A.3.1 INTRODUCCIÓN

Los patrones como elementos de reutilización tienen sus principios en la arquitectura (en el sentido clásico) donde fueron introducidos con el objetivo de capturar y posteriormente utilizar diseños que se habían aplicado en otras construcciones y que se catalogaron como complejos.

El arquitecto Christopher Alexander fue el primero en tratar de crear un formato específico para patrones en la arquitectura. Argumentaba que los métodos comunes aplicados en la disciplina daban lugar a productos que no satisfacían las demandas y requerimientos de los usuarios, y que eran ineficientes a la hora de conseguir el propósito de todo diseño y esfuerzo de la ingeniería: mejorar la condición humana. Describió algunos diseños eternos para tratar de conseguir sus metas. Propuso así, un paradigma para la arquitectura basado en tres conceptos: la calidad, la puerta y el camino.

- *La Calidad (La Calidad sin Nombre):* expresa la esencia de todas las cosas vivas y útiles que hacen sentir vivos a los seres humanos, les da satisfacción y mejoran la calidad de vida.
- *La Puerta*: expresa el mecanismo que permite alcanzar la calidad. La puerta es el conducto hacia la calidad. Se manifiesta como un lenguaje común de patrones.
- El Camino: expresa que siguiendo el camino se puede atravesar la puerta para llegar a la calidad.

Plantea que un patrón trata de extraer la esencia de un diseño (lo que puede llamarse la calidad sin nombre) para que pueda ser utilizada por otros arquitectos cuando se enfrente a problemas parecidos que el diseño resolvió.

Formuló la siguiente definición de Patrón:

"Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo ni siquiera en dos oportunidades de la misma forma".

Este concepto es transportado al ámbito del desarrollo de software orientado a objetos y se aplica al diseño. De allí es adoptado para el desarrollo en general y actualmente se aplica en la mayoría de las etapas del proceso.

Entre 1964 y 1979 Christopher Alexander escribió varios libros en referencia a este tema, en el que se destaca "*A pattern Language: Towns/Building/Construction*", donde describe 253 patrones en el formato que él mismo especificó.

En 1987, ya en ámbito del desarrollo de software, Ward Cunningham y Kent Beck, presentaron el libro "*Using Pattern Languajes for Object-Oriented Programs*". Ellos basándose en las ideas de Alexander, desarrollaron un pequeño lenguaje de patrones para los desarrolladores que trabajaban en Smalltalk

Poco después, en 1991, Jim Coplien confeccionó un catalogo de Idioms en C++ el que fue publicado en el libro "*Advanced C++ Programming Styles and Idioms*"

Desde 1990 a 1994, los que luego dieron por llamarse la banda de los cuatro (GoF) (Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides) recopilaron el primer catalogo de patrones de diseño, el que fue publicado en libro "*Design Patterns: Elements of Reusable Object-Oriented Software*".

Luego del trabajo de GoF, el cual se puede considerar introdujo finalmente el concepto de patrones en el desarrollo de software, las publicaciones y estudios referidos a patrones se han incrementado en gran numero, llevando el concepto de patrones al punto de considerarlo como el mas importante de los últimos tiempos en materia de desarrollo de software.

A.3.2 PATRONES DE SOFTWARE

Los patrones para el desarrollo de software se constituyen con uno de los más recientes avances en la Tecnología Orientada a Objetos. Los mismos participan en la intención de capturar, recopilar, y transmitir la experiencia del diseñador.

Considérese por un momento como el hombre a lo largo de la historia ha dominado una técnica en particular. Se podría decir que esto no es otra cosa que un proceso evolutivo que entre otras, cuenta con las siguientes etapas:

1. En una primera etapa los expertos aprenden a través del ensayo y error o por enseñanza de otros expertos. Las cosas son resueltas de manera artesanal
2. En una etapa subsiguiente, esta actividad toma más cuerpo y se crea una ciencia alrededor de ella.
3. Posteriormente se desarrollan técnicas soportadas dentro de la ciencia creada en la etapa anterior.
4. Sobre las técnicas se consigue un conocimiento común de cómo aplicarlas. Se logra establecer un metalenguaje entre los expertos que permite, entre otras cosas, la evolución de la ciencia en un sentido mas abstracto

Enfocando este proceso al dominio del desarrollo de software, se tiene una primera etapa donde crear una aplicación era un proceso artesanal, con el correr del tiempo surge la ingeniería de software, evolucionan diversas técnicas y herramientas, donde finalmente el concepto de patrones como técnica para el desarrollo, viene a encapsular de alguna manera el conocimiento o experiencia adquirido con el correr de los años, de forma que pueda ser transmitido, y se logre con ellos un lenguaje común dentro de la disciplina. Entonces lo que antes analistas o desarrolladores aplicaban en forma intuitiva para obtener buenas soluciones, ahora es modelado en patrones que pueden ser utilizados por otros analistas o desarrolladores que se encuentran con problemas similares.

Las metodologías Orientadas a Objetos tienen dentro de sus principios el "*no reinventar la rueda*" para la resolución de diferentes problemas, o en otras palabras, no

resolver cada problema desde cero. De esta forma los patrones se convierten en una parte importante de las mismas con el fin de lograr la reutilización.

A.3.3 DEFINICIONES

Al recorrer la literatura sobre patrones existen variadas definiciones, a continuación se presentan algunas de ellas:

> "Un patrón es una solución a un problema, aceptada como correcta, a la que se ha dado un nombre y que puede ser aplicada en otros contextos."
>
> "Un patrón es la abstracción de una forma concreta que puede repetirse en contextos específicos."
>
> "Un patrón es una información que captura la estructura esencial y la perspicacia de una familia de soluciones probadas con éxito para un problema repetitivo que surge en un cierto contexto y sistema."
>
> "Un patrón es una unidad de información nombrada, instructiva e intuitiva que captura la esencia de una familia exitosa de soluciones probadas a un problema recurrente dentro de un cierto contexto."
>
> "Cada patrón es una regla de tres partes, la cual expresa una relación entre un cierto contexto, un conjunto de fuerzas que ocurren repetidamente en ese contexto y una cierta configuración software que permite a estas fuerzas resolverse por si mismas."
>
> "Un patrón es una pieza de literatura que describe un problema de diseño y una solución general para es problema en un contexto particular"

A pesar de la gran variedad de definiciones, todas ellas rondan sobre los mismos conceptos: un problema y su solución exitosa, en un determinado contexto.

A.3.4 CARACTERÍSTICAS Y CUALIDADES

Se debe tener en cuenta que no todas las soluciones que tengan las características de un patrón, serán un patrón. Para serlo debe probarse que dicha solución se corresponde con un problema que se repite. Para que una solución sea considerada un patrón la misma debe pasar por una serie de pruebas que han sido llamadas test de patrones, mientras tanto solo es una buena solución o proto-patrón.

Las características deseables de un buen patrón son:

- *Debe solucionar un problema*: los patrones capturan soluciones, no solo principios o estrategias abstractas
- *Debe ser un concepto probado*: ser soluciones demostradas, no teorías o especulaciones.
- *La solución no es obvia*: muchas técnicas de solución de problemas tratan de hallar soluciones por medio de principios básicos. Los mejores patrones generan una solución a un problema de forma indirecta.
- *Describe participantes y relaciones entre ellos*: los patrones no sólo describen módulos sino estructuras del sistema y mecanismos más complejos.

Los patrones indican repetición, si algo no se repite probablemente no sea un patrón. Vale aclarar que la repetición no es la única característica importante. También se necesita probar que un patrón es adaptable y que es útil. La repetición es un concepto que se puede medir, es cuantificable, pero la adaptabilidad y la utilidad son conceptos cualitativos. Para mostrar la repetición, por ejemplo se puede tomar el criterio que se presente en por lo menos tres sistemas existentes; para mostrar la adaptabilidad habrá que explicar como el patrón es exitoso y por último para mostrar la utilidad se deberá explicar porque es exitoso y beneficioso.

Varios autores han listado las cualidades que son deseables en un patrón, a continuación se presenta un resumen de las listadas en el libro "*Christopher Alexander: an Introduction for Object-Oriented Designers*" de Doug Lea:

- Encapsulamiento y abstracción: cada patrón encapsula un problema bien definido y su solución en un dominio particular. Los patrones deberían de proporcionar límites claros que ayuden a cristalizar el entorno del problema y el entorno de la solución.
- Extensión y variabilidad: cada patrón debería ser abierto por extensión de tal forma que pueden aplicarse junto con otros patrones para solucionar problemas de mayor complejidad. Un patrón debería ser también capaz de realizar una variedad infinita de implementaciones (de forma individual, y también en conjunción con otros patrones).
- Generatividad y composición: cada patrón, una vez aplicado, genera un contexto resultante, el cual concuerda con el contexto inicial de uno o más de uno de los patrones del lenguaje. Está secuencia de patrones podría luego ser aplicada progresivamente para conseguir la generación de una solución completa. Los patrones se encuentran jerárquicamente relacionados, la mayoría admiten tanto una composición ascendente como una descendente dentro de la jerarquía.
- Equilibrio: cada patrón debe realizar algún tipo de balance entre sus efectos y restricciones.

A.3.5 BENEFICIOS DEL USO DE PATRONES

Costos, satisfacción del cliente, productividad y reducción de los tiempos de desarrollo, son talvez los puntos mas críticos en todo desarrollo de software. Los patrones contribuyen indirectamente en cada uno de ellos.

Productividad: La productividad de un diseñador novato se ve mejorada por el uso de patrones, al proveer experiencia en el dominio, los patrones acortan el circuito de descubrimiento interno para muchas importantes estructuras de diseño. Pero más importante aun es hecho que el uso de patrones evita el re-trabajo ocasionado por inexpertas decisiones de diseño.
Reducción de los tiempos de desarrollo: Los patrones de software permiten la reutilización de soluciones de diseño, lo que ocasiona la reducción del tiempo requerido para la etapa de diseño, ya que los desarrolladores al utilizar un patrón solo deben adaptarlo para el caso y no pensar una solución desde cero.
Costos: la reducción de costos es proporcional a los dos puntos anteriores
Satisfacción del cliente: es mayormente el resultado de los factores anteriores

Puede estar claro que el uso de patrones trae sus beneficios, pero existen puntos a tener en cuenta en referencia a que cosas los patrones no pueden hacer:

- Puesto que los patrones capturan experiencia, algunos importantes patrones para un nuevo dominio, no estarán allí para soportar el desarrollo del primer sistema. Se podría argumentar que patrones importantes pueden trascender de dominio en dominio y entonces

colaborar para el desarrollo del primer sistema. Esto es cierto en parte, pero cada nuevo dominio tendrá sus nuevos patrones que solo saldrán a la luz luego de adquirir experiencia en el dominio.

- Los patrones guían a los humanos no a las maquinas. Los mismos no generan código ni se encuentran viviendo dentro de una herramienta CASE. Los patrones, son literatura que asiste al proceso humano de tomar decisiones. Nunca podrán reemplazar a un programador humano.
- Por más que los patrones conlleven el conocimiento de diseñadores expertos, los mismos no volverán de un día para otro a novatos en expertos. Los patrones por si solos no pueden inculcar el sentido intuitivo de estética de los grandes diseñadores. Talvez su uso si pueda acortar el camino a recorrer para convertirse en un experto.

Es importante de destacar que los patrones son una herramienta más de las que disponen los diseñadores a la hora de atacar un problema. No son "la herramienta", no son la tan esperada bala de plata.

A.3.6 SECCIONES DE UN PATRÓN

Existe actualmente una gran variedad de ideas en cuanto a las secciones que deben conformar un patrón. Obviamente la Solución, es el corazón del patrón, pero un diagrama de su estructura también es de suma importancia. Las secciones que pueden decirse comunes entre los diferentes autores son:

- Name (Nombre)
- Intent (Propósito)
- Problem (Problema)
- Context (Contexto)
- Forces (Fuerzas)
- Solution (Solución)
- Sketch (Esquema)
- Resulting Context (Contexto Resultante)

Name
El nombre de los patrones es importante por los menos por dos razones:

1) Son la primera cosa que encuentran los diseñadores cuando están buscado una solución, si el nombre del patrón codifica adecuadamente lo que el mismo representa, el diseñador puede encontrar el patrón que mas se ajuste a sus necesidades aun en un lenguaje de patrones que no le es familiar
2) El nombre de los patrones conforman rápidamente el vocabulario del equipo de desarrollo. Un nombre corto bien elegido facilita la comunicación entre los desarrolladores.

Intent
El propósito es una frase o sentencia que engloba lo que el patrón hace, describiendo los puntos importantes del diseño y los problemas que el ataca. La sección *Intent* fue usada por primera vez en los patrones de GoF.

Problem
Esta sección describe el problema a ser resuelto. Sentencias cortas y concisas ayudan al diseñador a la lectura rápida.

Context
Incluye una historia de patrones que deben haber sido aplicados antes de considerar el actual. Especifica tamaño, alcance, lenguaje de programación, y

todo aquello que si cambia puede invalidar al patrón. El contexto de un patrón es crucial para el éxito de un lenguaje de patrones. Es difícil de escribir un buen contexto. La sección contexto madura con la experiencia: un diseñador encuentra una situación especial que invalida el patrón, entonces el contexto evoluciona volviéndose más restrictivo.

Forces
Los patrones no son reglas que se deben seguir ciegamente. Deben ser entendidos para luego adaptarse a las actuales necesidades. Si se entienden las fuerzas (forces) de una patrón, entonces se habrá entendido el problema y se habrá entendido la solución. Desde un punto de vista mas practico, las fuerzas ayudan al diseñador a entender como aplicar efectivamente un patrón.
El termino fuerzas (forces) es heredado de los patrones de arquitectura de Alexander. En una construcción arquitectónica existen estructuras, paredes, y otros elementos que balancean la fuerza de gravedad. De ahí la idea de balance de fuerzas implantada en los patrones. En software la idea de fuerzas es utilizada figurativamente.

Solution
Una buena solución tendrá suficientes detalles como para que los diseñadores conozcan que hace el patrón, pero en general no será suficiente para abarcar todos los contextos. Algunos patrones proveen solamente una solución parcial, pero dejan el camino para que otros patrones puedan ser usados para alcanzar una solución general.

Sketch
Alexander sostenía que el esquema era la esencia del patrón. En software los esquemas ayudan a los diseñadores a entender la relación entre las partes. Comúnmente comunican estructuras. Para citar un ejemplo, en el libro de GoF, se utilizan diagramas OMT para presentar esquemas de solución para cada uno de los patrones allí descriptos.

Resulting Context
Cada patrón transforma un sistema de un contexto a un nuevo contexto. El contexto resultante de un patrón es la entrada del patrón que le sigue.
El contexto resultante expresa: las fuerzas que han sido resueltas, que problemas pueden aparecer por el uso de este patrón, y finalmente que patrones relacionados pueden venir después.
Los contextos relativos a patrones mantienen a los mismos unidos en un lenguaje de patrones.

A.3.7 LENGUAJES DE PATRONES

Un lenguaje de patrones es una colección de patrones que construyendo uno con otros se puede generar un sistema. Un patrón aislado resuelve un problema aislado, un lenguaje de patrones construye un sistema [App00]. Es a través de un lenguaje de patrones que los patrones alcanzan su máximo potencial. Como gran parte de los otros conceptos, este también es heredado de la arquitectura, donde Alexander fue quien lo popularizó.

Un lenguaje de patrones no debe ser confundido con un lenguaje de programación. Un lenguaje de patrones es una pieza de literatura que describe una arquitectura, un diseño, un *framework*. Si bien tiene una estructura, no es en el mismo sentido formal que puede ser una estructura de un lenguaje de programación.

El termino “lenguaje de patrones” ha sido fuente de confusión, algunos autores lo han usado indistintamente con el termino “sistema de patrones”. Esta cercanamente relacionado con la noción de Parna sobre “*Software Family*”. Una *Software Family* comprende varios miembros relacionados unos con otros por sus partes en común, distinguibles entre ellos por su variabilidad. Se puede pensar una lenguaje de patrones como una colección de reglas para construir todos y solamente los miembros de una familia.

Un lenguaje de patrones no es simplemente un árbol de decisión de patrones. Esto es cierto porque los patrones de un lenguaje de patrones forman un grafo acíclico, y no grafo jerárquico. El número de caminos distintos a través del lenguaje es muy grande.

Un catalogo de patrones no constituye un lenguaje de patrones. Un catalogo puede no estar completo como para generar todos los programas en un dominio. Esto no significa que un catalogo no tenga una estructura, ni que se le reste importancia dentro del mundo de los patrones, de hecho hoy en día los catálogos son la fuente mas común de patrones. Solamente que el concepto de lenguaje de patrones tiende a ser mas amplio. De todas formas aun un lenguaje de patrones manejable no puede soportar todos los aspectos del diseño de un sistema en general.

A.3.8 CLASES DE PATRONES

En el marco de la ingeniería de software, existen diferentes ámbitos donde se pueden aplicar patrones. La siguiente es una de las posibles clasificaciones: Patrones de Arquitectura, Patrones de Diseño, Patrones de Análisis, Patrones de Proceso o de Organización, y los denominados Idiomas.

La diferencia entre estas clases de patrones está en los diferentes niveles de abstracción y detalle, y del contexto particular en el cual se aplican o de la etapa en el proceso de desarrollo. Así, los patrones de arquitectura son estrategias de alto nivel que involucran a los componentes, las propiedades y mecanismos globales de un sistema. Los patrones de diseño son tácticas de media escala relacionados con la estructura y el comportamiento de entidades y sus relaciones. No influyen sobre toda la estructura del sistema, pero define micro arquitecturas de subsistemas y componentes. Los patrones de análisis se refieren a la etapa de análisis del ciclo de vida de construcción de software. Los patrones organizacionales describen la estructuración del personal en el desarrollo de software. Los patrones de programación o idiomas, son específicos de las técnicas de un lenguaje de programación que afectan a partes pequeñas del comportamiento de los componentes de un sistema.

La siguiente tabla corresponde a un resumen de la clasificación planteada:

Clase de Patrón	Comentario	Problemas	Soluciones	Fase de Desarrollo
Patrones de Arquitectura	Relacionados a la interacción de objetos dentro o entre niveles arquitectónicos	Problemas arquitectónicos, adaptabilidad a requerimientos cambiantes, performance, modularidad, acoplamiento	Patrones de llamadas entre objetos (similar a los patrones de diseño), decisiones y criterios arquitectónicos, empaquetado de funcionalidad	Diseño inicial

Patrones de Diseño	Conceptos de ciencia de computación en general, independiente de aplicación	Claridad de diseño, multiplicación de clases, adaptabilidad a requerimientos cambiantes, etc	Comportamiento de factoría, Clase-Responsabilidad-Contrato (CRC)	Diseño detallado
Patrones de Análisis	Usualmente específicos de aplicación o industria	Modelado del dominio, completitud, integración y equilibrio de objetivos múltiples, planeamiento para capacidades adicionales comunes	Modelos de dominio, conocimiento sobre lo que habrá de incluirse (p. ej. logging & reinicio)	Análisis
Patrones de Proceso o de Organización	Desarrollo o procesos de administración de proyectos, o técnicas, o estructuras de organización	Productividad, comunicación efectiva y eficiente	Armado de equipo, ciclo de vida del software, asignación de roles, prescripciones de comunicación	Planeamiento
Idiomas	Estándares de codificación y proyecto	Operaciones comunes bien conocidas en un nuevo ambiente, o a través de un grupo. Legibilidad, predictibilidad	Sumamente específicos de un lenguaje, plataforma o ambiente	Implementación, Mantenimiento, Despliegue

También se puede hablar de otros tipos de patrones software, como ser:

- Patrones de programación concurrente.
- Patrones de interfaz gráfica.
- Patrones de organización del código.
- Patrones de optimización de código.
- Patrones de robustez de código.
- Patrones de la fase de prueba.
- Patrones en la ingeniería de requerimientos

Para el propósito del presente trabajo resulta interesante analizar un poco más en particular las clases: Patrones Arquitectónicos y Patrones de Diseño.

A.3.8.1 Patrones Arquitectónicos

Este tipo de patrones son aquellos que tratan con vistas de alto nivel de un sistema en el cual sus primitivos son componentes y conectores.

Los patrones arquitectónicos son aproximadamente lo mismo que lo que se acostumbra a definir como estilos arquitectónicos y ambos términos se utilizan indistintamente en al literatura. Algunos patrones coinciden con los estilos hasta en el nombre con que se los designa. Ver Anexo 2

Una buena referencia en este tema lo constituye el libro "*Pattern-Oriented Software Architecture: A System of Patterns*" de F. Buschmann, donde los patrones propuestos son:

Layers
Permite estructurar aplicaciones que se pueden descomponer en grupos de subtareas, donde cada grupo está en un determinado nivel de abstracción.

Pipes & Filtres
Provee una estructura para sistemas que procesan un flujo de datos. Cada etapa del proceso es encapsulada como un filtro. Los datos se pasan entre filtros adyacentes mediante Pipes. Recombinando filtros se obtienen familias de sistemas relacionados.

Blackboard
Útil para sistemas en que no se conoce una solución o estrategia determinista. Varios subsistemas especializados ensamblan su conocimiento para construir una posible solución parcial.

Broker
Permite estructurar sistemas distribuidos desacoplados que interaccionan mediante invocación de servicios remotos. Un componente Broker es responsable de coordinar la comunicación, así como de transmitir resultados y excepciones.

Model-View-Controller
Divide una aplicación interactiva en 3 componentes. *Model* contiene la información y funcionalidad principal. *Views* muestran información al usuario. *Controllers* gestionan la entrada de usuario. Un mecanismo de propagación de cambios asegura la consistencia entre el modelo y la interfaz de usuario.

Presentation-Abstraction-Control
Estructura una aplicación interactiva como una jerarquía de agentes que cooperan. Cada agente es responsable de un determinado aspecto de la funcionalidad y consta de tres componentes: Presentación, Abstracción y Control, que separan la interacción con el usuario de la funcionalidad central y la comunicación con otros agentes.

Microkernel
Separar un mínimo núcleo funcional de funcionalidad extendida y partes específicas del cliente. El *Microkernel* también sirve como punto donde engarzar estas piezas y coordinar su colaboración.

Reflection
Proporciona un mecanismo para cambiar la estructura y el comportamiento del sistema dinámicamente. La aplicación se divide en dos partes: un meta-nivel y un nivel-base. El meta-nivel hace al software autoconsciente.

A.3.8.2 Patrones de Diseño

Un patrón de este tipo identifica, abstrae y nombra los aspectos elementales de una estructura de diseño, donde los componentes, son las clases y objetos, y sus mecanismos de interacción son mensajes.

Cada patrón prescribe una estructura de clases, sus roles y colaboraciones, y una adecuada asignación de métodos para resolver un problema de diseño en una manera flexible y adaptable.

La aplicación de los patrones en un diseño consiste en identificar el patrón que resuelve el problema de diseño encontrado y aplicar la solución abstracta prescrita por el patrón a dicho problema.

Uno de los beneficios de utilizar patrones es el entendimiento y documentación de diseños orientados a objetos. Los patrones, también, mejoran el mantenimiento de sistemas ya que proveen una especificación explícita de clases e interacción entre objetos. Además, los patrones proveen un vocabulario para discutir y comunicar decisiones de diseño en término de estructuras de clases en lugar de objetos.

Los patrones de diseño ayudan a elegir diseños alternativos que hacen un sistema reutilizable y evitan alternativas que comprometan la reutilización

El libro que ha popularizado los patrones de diseño es el conocido "*Design Patterns: Elements of Reusable Object Oriented Software*". En este se recopilan 23 patrones agrupados en tres categorías: de creación, de estructura y de comportamiento.

A continuación se listan los patrones de este catalogo, con una breve descripción del propósito de cada uno de ellos:

Abstract Factory
Proporciona una interfaz para crear familias de objetos relacionados sin especificar sus clases concretas

Adapter
Convierte la interfaz de una clase en otra distinta, que es la que esperan los clientes

Bridge
Desacopla una abstracción de su implementación, de manera que puedan variar de forma independiente

Builder
Separa la construcción de un objeto complejo de su representación

Chain of Responsibility
Evita acoplar el emisor de una petición a su receptor, al dar a más de un objeto la posibilidad de responder a la petición.

Command
Encapsula una petición en un objeto, permitiendo así entre otras cosas parametrizar a los clientes con distintas peticiones, encolar o llevar un registro de las mismas.

Composite
Combina objetos en estructuras de árboles para representar jerarquías de parte-todo.

Decorator
Añade dinámicamente nuevas responsabilidades a un objeto. Alternativa de la herencia

Facade
Proporciona una interfaz unificada para un conjunto de interfaces de un subsistema.

Factory Method
Define una interfaz para crear un objeto, pero deja que sean las subclases las que decidan que clase instanciar.

Flyweight
Usa el compartimiento para permitir un gran numero de objetos de grano fino de forma eficiente

Interpreter
Dado un lenguaje, define una representación de su gramática junto con un intérprete que usa dicha representación para interpretar las sentencias del lenguaje.

Iterator
Proporciona un modo de acceder secuencialmente a los elementos de un objeto agregado sin exponer su representación interna

Mediator
Define un objeto que encapsula como interactúan un conjunto de objetos.

Memento
Representa y externaliza el estado interno de un objeto sin violar su encapsulación

Observer
Defina una dependencia de uno a muchos entre objetos, de forma que cuando un objeto cambia de estado, se notifican y se actualizan automáticamente todos los objetos que dependen de él.

Prototype
Especifica los tipos de objetos a crear por medio de una instancia prototípica, y crea nuevos objetos copiando este prototipo

Proxy
Proporciona un sustituto o representante de otro objeto para controlar el acceso a este

Singleton
Garantiza que una clase solo tenga una instancia y proporciona un punto de acceso global a la misma.

State
Permite que un objeto modifique su comportamiento cada vez que cambia su estado interno.

Strategy
Define una familia de algoritmos, encapsula cada uno de ellos y los hace intercambiables.

Template Method
Define en una operación el esqueleto de un algoritmo delegando en las subclases algunos de sus pasos

Visitor
Representa una operación sobre los elementos de una estructura de objetos. Permite definir una nueva operación sin cambiar las clases de los elementos sobre los que opera.

A.3.9 ANTIPATRONES

Se puede describir a un antipatrón, como un patrón que invariablemente conduce a una mala solución para un problema. En las palabras de W.J.Brown:

> *"Un antipatrón es una forma literaria que describe una solución comúnmente dada a un problema que genera decididamente consecuencias negativas"*

Los antipatrones se basan en la idea de que puede resultar más fácil detectar a priori fallos que elegir el camino correcto, o lo que es lo mismo, descartar las alternativas incorrectas puede ayudar a la elección de la mejor alternativa.

Al documentarse los antipatrones, además de los patrones de diseño, se dan argumentos a los diseñadores de sistemas para no escoger malos caminos, en base a documentación disponible en lugar de simplemente la intuición.

Los patrones de diseño se concentran en la definición y generalización de una solución positiva a problemas que ocurren comúnmente. Los antipatrones, por el contrario, consisten de dos soluciones: el punto de partida del proceso de definición de un antipatrón es la existencia de una solución problemática. Esta solución antipatrón genera consecuencias negativas para la administración y desarrollo de software. La segunda solución, *refactored solution,* es el objetivo del proceso de aplicar el antipatrón y provee el medio para que el proyecto alcance el éxito. Los elementos fundamentales de un antipatrón son los siguientes:

- *Name* (Nombre). El nombre del antipatrón define la solución en forma compacta e informativa
- *AntiPattern Solution* (Solución Antipatrón). Corresponde al síntoma, consecuencia y una descripción abstracta de la solución problemática
- *Refactored Solution* (Solución Refabricada). Descripción de las consecuencias positivas encontradas después de aplicada la solución refabricada.

La clasificación realizada por Brown agrupa a los antipatrones según: antipatrones de desarrollo, de arquitectura de software y de gestión de proyectos.

- Antipatrones de desarrollo
  Este grupo de antipatrones son encontrados a bajo nivel del desarrollo de software, específicamente en programación y administración de código

  - ***The Blob***: Clases de gran tamaño, una clase que lo hace todo, el resto de las clases solo contienen datos
  - ***Lava Flow***: Código muerto, y escasa información de diseño no permiten realizar cambios
  - ***Functional Decomposition***: Diseño No Orientado a Objetos
  - ***Poltergeists***: No se sabe lo que hacen algunas clases, solo añaden sobrecarga y complejidad innecesaria. Afecta al rendimiento del sistema

- ***Golden Hammer***: Para un martillo todos son clavos, el mismo concepto se aplica de forma obsesiva para cualquier problema
- ***Spaghetti Code***: Software desarrollado a medida (ad-hoc) difícil de extender y optimizar. Resultado de la inexperiencia. Abundan estructuras de decisión condicional tipo IF.
- ***Cut & Paste programming***: cortar y pegar código, el mismo código repetido en varios sitios, difícil de modificar y mantener

- Antipatrones de arquitectura de software
  Este grupo involucra problemas en la definición y mantenimiento en las estructuras de un sistema

  - ***Stovepipe enterprise***: En la empresa se desarrollan varios sistemas de manera independiente y a distintos niveles. Esto dificulta la interoperabilidad, la reutilización e incrementa costos. Se crean islas automatizadas dentro de la misma empresa
  - ***Stovepipe system***: Es análogo al *Stovepipe Enterprise*, pero a nivel de un sistema
  - ***Vendor Lock-In***: Arquitectura dependiente de un fabricante, los productos de diferentes fabricantes no suelen ser compatibles
  - ***Architecture by implication***: No existe documentación de la Arquitectura del sistema. Arquitectura implícita.
  - ***Design by comite***: Diseño por Comité, se diseña a través de las reuniones de un grupo demasiado numeroso o inexperto
  - ***Reinvent the Wheel***: Se supone que se debe desarrollar desde cero, falta información y tecnología reutilizable entre proyectos

- Antipatrones de gestión de proyectos
  Este grupo esta relacionado con la administración de proyectos software y personas.
  - ***Analysis paralysis***: Análisis tan completo que nunca termina, el proyecto muere en la fase de análisis
  - ***Death by planning***: Excesiva planificación. Los procesos de planificación deberían ser incrementales e iterativos
  - ***Corncob***: Personas problemáticas, difíciles obstruyen y desvían en proceso de desarrollo
  - ***Irrational management***: La falta de decisión puede generar decisiones de facto y desarrollos de emergencia
  - ***Project mismanegement***: Solo apagando fuegos, no se cuida el diseño, solo implementación

A.3.10 RESUMEN ANEXO

A lo largo del presente anexo se introdujo el concepto de patrones, partiendo desde sus principios en la Arquitectura tradicional ligado a los trabajos de Christopher Alexander, para luego llegar a las diferentes representaciones de los mismos en el ámbito del desarrollo de Software. Se analizaron las distintas definiciones, sus propiedades y presentaciones, y una breve descripción de lo que ha dado por llamarse “lenguaje de patrones”. Se puso particular interés en los Patrones de Diseño y los Patrones Arquitectónicos. Por ultimo se analizó superficialmente el concepto de antipatrones.