DISEÑO E IMPLEMENTACIÓN DE UN SERVICIO PARA LA CAPTURA DE LA INTERACCIÓN DE LOS USUARIOS EN DISPOSITIVOS MÓVILES

TESIS DE FIN DE MASTER

Eugenio Bolaños Cárdenas

Septiembre, 2014

Directores:
Vicente Pelechano y Miriam Gil

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISEÑO E IMPLEMENTACIÓN DE UN SERVICIO PARA LA CAPTURA DE LA INTERACCIÓN DE LOS USUARIOS EN DISPOSITIVOS MÓVILES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Nombre Tesis de Máster:

Diseño e implementación de un servicio para la captura de la interacción de los usuarios en dispositivos móviles

Fecha:

Septiembre de 2014

Nombre del Master:

Ingeniería del Software, Métodos Formales y Sistemas de Información

Tesis de máster de:

Eugenio Bolaños Cárdenas

Directores de máster:

Vicente Pelechano Ferragud

Miriam Gil Pascual

Universidad Politécnica de Valencia

Camino de Vera s/n, Edif. 1F

46022 - Valencia, España

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Agradecimientos

Esta tesis está dedicada muy especialmente a mi familia y a María Adelaida Vélez. Gracias a su apoyo, sacrificio y paciencia he podido terminar este ciclo de estudios de Master. Espero que con este gran esfuerzo podamos recolectar y compartir los frutos venideros.

Agradezco a mis amigos, puesto que siempre me han dado ánimo para terminar mis estudios, y me han brindado apoyo para no desistir.

Doy las gracias a mis directores de Tesis Miriam Gil y Vicente Pelechano, porque sin sus aportes, apoyo y asesoría, no hubiese sido posible realizar este trabajo. Además han despertado en mí el interés en desarrollar aplicaciones a nivel laboral que utilicen el contexto para el beneficio de los usuarios.

Para acabar me gustaría agradecer los aportes que he recibido de mis profesores y compañeros del Master, mis compañeros de trabajo y todas aquellas personas que creyeron en mí.

Eugenio Bolaños Cárdenas

Índice de contenidos

Índice de figuras

Índice de tablas

Abreviaciones

ADB Android Debug Bridge
ADT Android Developer Tools
API Application Programming Interface
ARFF Attribute-Relation File Format
BD Base de datos
CSV Comma Separated Values
DRM Day Reconstruction Method
ECA Event-Condition-Action
ESM Experience Sampling Method
GMM Gaussian Mixture Model
HTTP Hypertext Transfer Protocol
HCI Human Computer Interactions
IoT Internet of Things
JSON JavaScript Object Notation
KDD Knowledge Discovery in Databases
MAC Media Access Control
MMS Multimedia Messaging Service
MQTT MQ Telemetry Transport
MVC Model-View-Controller
M2M Machine-to-machine
PC Personal Computer / Ordenador personal
OS Operating System
QoS Quality of Service
RPC Remote Procedure Call
SDK Software development kit
UI User Interface
UPV Universidad Politécnica de Valencia
URI Uniform Resource Identifier
UUID Universal Unique IDentifier
XML eXtensible Markup Language

“First were mainframes, each shared by lots of people. Now we are in the personal computing era, person and machine staring uneasily at each other across the desktop. Next comes ubiquitous computing, or the age of calm technology, when technology recedes into the background of our lives.”

— **Mark David Weiser**, 1952 - 1999

1. Introducción

El uso y la percepción de los ordenadores en las últimas décadas, y más drásticamente en los últimos 10 años, ha cambiado vertiginosamente. Inicialmente, múltiples usuarios compartían los recursos computacionales, donde a cada uno se le asignaba un intervalo de tiempo para su uso. Luego vinieron los PCs, que nos permitieron tener un equipo propio a nuestra disposición. Con la evolución de la tecnología, aparecieron los ordenadores móviles; inicialmente como ordenadores portátiles y netbooks, donde luego toman la forma de tabletas, teléfonos inteligentes y otros dispositivos móviles híbridos. Todas estas clases de dispositivos móviles que caben en un bolsillo, se han proyectado en ventas de 2.591.753 de unidades para el año 2015[1].

Con esta gran penetración de dispositivos y ordenadores de todo tipo, los términos computación y conectividad, se han extendido en la humanidad hasta el punto que nos pone a pensar en una redefinición de la pirámide de Maslow, al ser incluidos en ella. Uno de los grandes potenciadores de esto, son los dispositivos móviles, los cuales están integrados cada vez más a las actividades de nuestra vida diaria, y es por ello que se han vuelto la interfaz, por defecto, para la interacción con sistemas o servicios de información. De allí que este tipo de dispositivos son una fuente de datos importante, la cual podemos aprovechar para obtener información del contexto de usuario (sensores, localización, conectividad, personalización, etc.).

Adicional a la capacidad de los dispositivos y su penetración, el desarrollo de aplicaciones y servicio móviles van velozmente en aumento (1,371,453 aplicaciones Android en Google Play en septiembre de 2014, Figura 1 - Número de Apps Android en Google Play [2]), generando mayor dependencia e inercia, por parte de los usuarios a usar los dispositivos; es por ello que demandar su atención, de forma poco o muy perceptible, debe depender del contexto en el que se encuentren; en contextos donde solo debe ser demanda la atención del usuario cuando se requiera (ya que este es un recurso valioso y limitado), el servicio o aplicación móvil (app, como es más conocida en la actualidad), debe ser capaz de capturar y procesar el

[1] Gartner (Junio 2014)

[2] AppBrain Stats, Number of Android applications http://www.appbrain.com/stats/number-of-android-apps

contexto para adaptarse a sus necesidades. - El aprendizaje de adaptación de estos sistemas se puede lograr a través del aprendizaje automático o machine learning; el cual tiene muchas áreas de aplicación, tales como localización, interfaces adaptables, observación y reconocimiento humano, reproductores de música, etc.

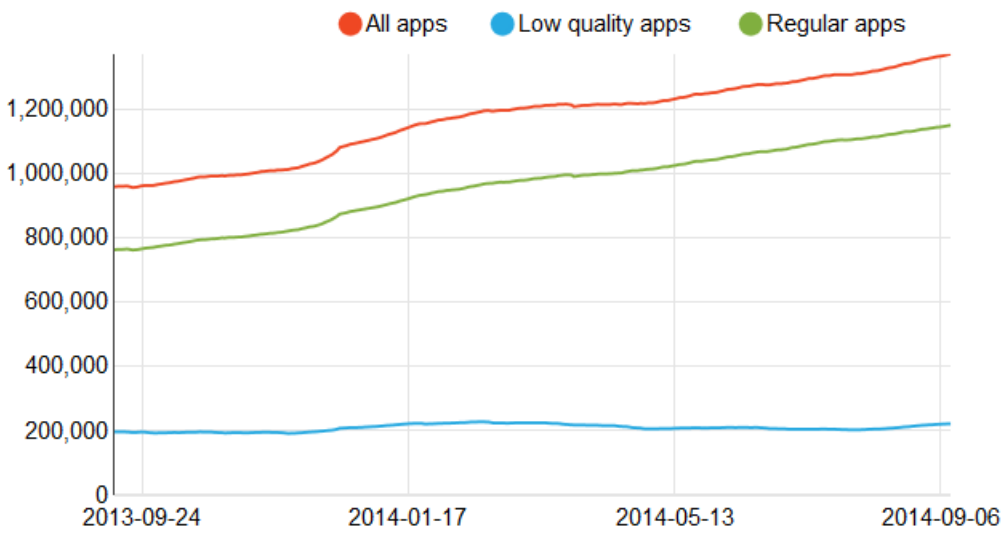


Figura 1 - Número de Apps Android en Google Play

Por ultimo podemos afirmar que los teléfonos móviles son inherentemente personales y el potencial para percibir el entorno del usuario, o en otras palabras, el contexto del usuario, es atractivo para la creación de nuevos servicios y aplicaciones. La comodidad y la disponibilidad de los teléfonos móviles y tiendas de aplicaciones hacen que sea más fácil la distribución y masificación de las aplicaciones. Más importante aún, los teléfonos móviles tienen varios sensores integrados (por ejemplo acelerómetro, sensor de proximidad, giroscopio, etc.). Estos sensores móviles son utilizados principalmente por el sistema operativo móvil para mejorar la experiencia del usuario, tales como la funcionalidad de la aplicación o la interacción del usuario del teléfono móvil (por ejemplo vibración, detección de orientación de la pantalla). Pero no solo es el sistema operativo quien usa estos datos, cada vez es más significativo que en el momento de diseñar servicios o aplicaciones sea usada esta información para adaptar configuraciones, interfaces de usuario o eventos del sistema a las circunstancias.

1.1 Motivación

Los dispositivos móviles son actualmente las herramientas informáticas más extendidas en todo el mundo; es por ello que con el aumento de usuarios móviles demandando sistemas más inteligentes, con mayores niveles de personalización, autoadaptables, sensibles a su contexto y a sus interacciones, conlleva nuevas formas de diseñar las aplicaciones. Por este motivo, surgen retos como adaptación, generación de conocimiento, escalabilidad, confiabilidad, personalización, reutilización, etc., que requieren de un esfuerzo de colaboración para gestionar los ciclos de vida de las aplicaciones. Para lograrlo se requiere aplicar conceptos de diferentes áreas de investigación. Un paso interesante, es hacer frente a los retos de la reutilización del contexto y generación de conocimiento. Por tanto, la siguiente pregunta logra enmarcar el desarrollo de esta tesis: "¿Cómo es posible capturar la información asociada al usuario (contexto e interacciones), para luego generar conocimiento sobre el mismo (inferencias)?".

Con la implementación y análisis de este proyecto se busca aplicar conceptos propuestos en las múltiples investigaciones que se están realizando en el ámbito de context-awareness y machine learning en entornos móviles, todo esto por medio del uso de metodologías, tecnologías y frameworks que actualmente se encuentran disponibles y que han sido estudiados a lo largo del Master de Ingeniería de software, métodos formales y sistemas de información. La selección de las tecnologías y plataformas que se describirán más adelante, son el resultado de la consulta y lectura sobre el estado del arte del tema en cuestión. Como ya se ha mencionado anteriormente, el hecho de tener servicios que posibiliten el acceso a la información contextual de los usuarios de dispositivos móviles en tiempo real, es cada vez más importante para la implementación de aplicaciones o servicios útiles, proactivos y personalizados.

1.2 Problema

Cada vez las aplicaciones deben ser más flexibles y auto adaptables. Esto quiere decir que estas aplicaciones deben tener como entrada (input), un mayor número de variables para así generar una óptima interacción con el usuario u otros sistemas. Dentro de esas tantas variables, el contexto al cual están expuestas y son manipuladas las aplicaciones juega un papel importante. Pero esto no se queda allí, también es relevante considerar variables relacionadas con las interacciones y las preferencias de usuario, que pueden ser definidas dentro de la misma aplicación o en el mismo sistema operativo que corre en el dispositivo móvil. Por esta razón existe la necesidad de implementar un servicio que permita obtener datos de usuario, para luego, a través de un mecanismo de análisis de datos, se pueda generar conocimiento para una posterior adaptación de la interfaz de usuario de la aplicación (trabajos previos). En particular, nos centraremos en usuarios de dispositivos Android, por ser una plataforma software en crecimiento, de código abierto, multi tarea, es soportada por una amplia gama de equipos y permite la implementación de servicios que corren en background. Por otra parte, soporta una gran cantidad de sensores y capturas de interacción.

En este trabajo se presenta una aplicación móvil que busca dar a los usuarios finales la clasificación de su estado (perfil) sobre un conjunto predefinido de estos, dependiendo de las condiciones de su entorno y las preferencias del mismo; donde luego esta información podría ser usada para generar auto adaptaciones de la configuración o generar interfaces de usuario sensibles al contexto. Como se ha comentado en apartados anteriores, este tipo de comportamientos es necesario y puede ser cada vez más demandado por los usuarios. Es en este punto, donde se evidencia la existencia de una problemática, ya que hay bastante teoría al respecto, pero pocas soluciones o ejemplos prácticos que proporcionen una guía para futuros desarrollos.

1.3 Objetivo de la tesis

Implementar un servicio móvil para la plataforma Android que permita capturar interacciones e información de contexto del usuario para luego ser procesada a través de técnicas de aprendizaje automático las cuales, permiten inferir conocimiento que, posteriormente puede ser usado para adaptación de un sistema o aplicación.

Objetivos específicos

- Estudiar los distintos tipos de eventos e información de contexto que se pueden capturar en la plataforma móvil Android
- Proporcionar una implementación de un servicio móvil para capturar las interacciones del usuario e información de contexto.
- Crear un repositorio donde guardar la información capturada.
- Inferir conocimiento a partir de la información de usuario almacenada.
- Aplicar el desarrollo a un caso de estudio concreto

1.4 Solución propuesta: Componentes de implementación

Para la implementación del servicio móvil (Android) se propone una arquitectura multicapa que permite reutilizar los componentes de cada una en posteriores trabajos. Se puede mencionar de forma muy general que la capa de sensibilidad al contexto incluye un consumidor de los sensores móviles y las interacciones de usuario, un módulo de pre-procesado y tratamiento de datos; la capa de persistencia permite almacenar los datos de contexto capturados de tal forma que se pueden compartir y la capa de machine learning por su parte se encarga de la generación y uso de un modelo clasificador para inferir datos del usuario.

Para facilitar la implementación, se dividió el proyecto en 3 frentes de trabajo los cuales hacen parte de diferentes ramas de la computación que no se encuentran aisladas.

Context-awareness e interacciones de usuario
El primer frente está relacionado con la información referente al usuario: contexto e interacciones. Los datos de usuario se obtienen de forma cruda (raw) desde diferentes fuentes de datos (que en apartados posteriores veremos), donde luego a partir de dichos datos se hace un pre-procesamiento, para clasificarlos en variables nominales que facilitan y optimizan etapas posteriores de aprendizaje.

Persistencia
Para la persistencia de los datos de contexto en Android, existen diversas opciones, las cuales son mencionadas y explicadas posteriormente. Para este frente de trabajo, el objetivo es persistir la mayor cantidad de datos de usuario generados por el componente de Context-awareness, para su posterior análisis. Todos los datos registrados tienen como llaves el timestamp y el id del dispositivo.

Análisis de datos
Por último, para hacer la inferencia de conocimiento existen múltiples opciones. Inicialmente el trabajo estaba encaminado al uso de ontologías para el almacenamiento y posterior generación de conocimiento, pero debido a la complejidad de las mismas y al bajo nivel de madurez de las APIs en dispositivos móviles, que permiten su manipulación, fueron descartadas. Así que, la implementación se enfocó al uso de técnicas de minería de datos y machine learning. En este caso se evaluó y utilizó la librería para Java que proporciona el proyecto Weka. En posteriores apartados se detalla el uso de esta librería dentro del proyecto.

Por último es bueno aclarar, que el desarrollo del proyecto permite demostrar los diferentes componentes a utilizar, en un enfoque local al dispositivo móvil, para generar aplicaciones sensibles al contexto y a las interacciones con el usuario, proporcionando información que el mismo sistema tendrá en cuenta para aprender. Esta retroalimentación permite a las aplicaciones ir afinando su nivel de interacción con los usuarios para adaptarse a los diferentes momentos de uso.

1.5 Contexto de la tesis

El proyecto presentado en esta tesis se desarrolla en el ámbito de machine learning en conjunto con context-awareness, concretamente capturando datos que rodean al usuario (interacciones y contexto) para luego, a través de procesos computacionales, obtener conocimiento sobre el usuario y su entorno.

Con esta tesis, se busca generar un trabajo práctico y aplicado que proporcione algún aporte a futuras investigaciones dentro de PROS (Centro de Investigación en Métodos de Producción de Software) de la Universidad Politécnica de Valencia, y en especial para la tesis doctoral "Adapting Interaction Obtrusiveness: Making ubiquitous interactions less obnoxious", presentada por Miriam Gil. (M. Gil Pascual, 2013).

1.6 Estructura del documento

La estructura del documento está dividida en siete capítulos, incluida esta introducción. A continuación se detalla brevemente el contenido de cada uno, como guía de la organización del resto de la tesis:

Capítulo 2. Conceptos generales

Introduce los conceptos generales relacionados con esta tesis, para dar al lector los conocimientos básicos necesarios para su mejor comprensión.

Capítulo 3. Contexto Tecnológico

Presenta las tecnologías que han sido usadas o relacionadas en este proyecto, tales como el desarrollo de aplicaciones para Android en Java, el entorno de desarrollo Eclipse y el framework para el análisis de conocimiento Weka, entre otros.

Capítulo 4. Trabajos relacionados

Introduce y presenta algunos de los trabajos relacionados con la captura de datos de contexto e interacciones de usuario, y muy detalladamente el framework AWARE, además presenta la implementación de un framework para facilitar el uso Weka en dispositivos móviles Android.

Capítulo 5. Desarrollo de la Propuesta

Expone aspectos de implementación y desarrollo de la propuesta, presentando la definición de requisitos, el análisis y diseño, la definición de la arquitectura general y algunos detalles de la implementación: persistencia, context-awareness y machine learning.

Capítulo 6. Caso de estudio

En este capítulo se expone casos de estudio para una mejor compresión de la aplicabilidad y uso de la solución presentada.

Capítulo 7. Conclusiones y trabajo futuro

Plantea las conclusiones extraídas durante la realización de esta tesina y presenta algunos retos para la realización de trabajos futuros.

2. Conceptos generales

A continuación se ilustran y explican los conceptos generales, tales como context awareness, machine learning (aprendizaje automático) y sistemas de auto-adaptación UI; los cuales son la base para el desarrollo del proyecto. El análisis y estudio de estos conceptos, da la posibilidad de entender como pueden ser aplicados en la implementación y proveen un soporte teórico sobre las decisiones que se tomen en el momento de desarrollar la solución al problema planteado.

2.1 Context-Awareness

En esta sección, comenzaremos haciendo alusión a la definición de ***contexto*** dada por G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, P. Steggles (1999): "Es cualquier información que se puede utilizar para caracterizar la situación de una entidad (es decir, una persona, lugar u objeto) y que se considera relevante para la interacción entre un usuario y una aplicación, incluyendo el usuario y aplicaciones entre sí. El contexto es típicamente la ubicación, la identidad y el estado de personas, grupos y objetos computacionales y físicos.”. Con esto identificamos claramente que el contexto es una fuente de información basta y amplia, que posibilita el estudio e implementación de múltiples aplicaciones. Con este concepto claro, podemos introducir un nuevo termino context-awareness, el cual puede definirse como: "El uso del contexto para proporcionar información y/o servicios de interés para el usuario, donde la relevancia depende de la tarea del usuario". Este tipo de comportamientos aplicados a un sistema o aplicación pueden mejorar la experiencia de usuario, la usabilidad o el nivel de personalización de los mismos, según los describen Alf Inge Wang1, Qadeer Khan Ahmad2 (2014).

El contexto se adquiere de diversas fuentes (por ejemplo sensores) para clasificar y describir el entorno de las entidades. Como podemos observar en Ferreira, D. (2013) y A. K. Dey, G. D. Abowd, and D. Salber (2001), hay cuatro categorías o características esenciales de información de contexto:

- Identidad: capacidad de asociar un único identificador de una entidad
- Ubicación: es cualquier información geográfica asociada con una entidad (coordenadas GPS, proximidad, elevación, etc.)
- Estado (o actividad): recolección de toda la posible información de la entidad, detectada o capturada por sensores.
- Tiempo: es el registro histórico de información de contexto

En F. González, C. Antonio (2010), se categorizan las fuentes de datos de contexto en cuatro clases principales (como se muestra en la Figura 2 - Fuentes de datos de contexto).

- La interacción del usuario con el dispositivo, que incluye las actividades del uso de los dispositivos, biometría, comunicaciones móviles o la información proveniente de servicios basados en internet tales como la web 2.0 o la computación en nube.
- Los datos de localización relacionados con lo que está sucediendo en el entorno del dispositivo móvil o información ambiental relacionada con el mundo físico.
- Información sobre medio ambiente, tales como el calor, la luz, los niveles de sonido o la velocidad.
- Información del dispositivo interno

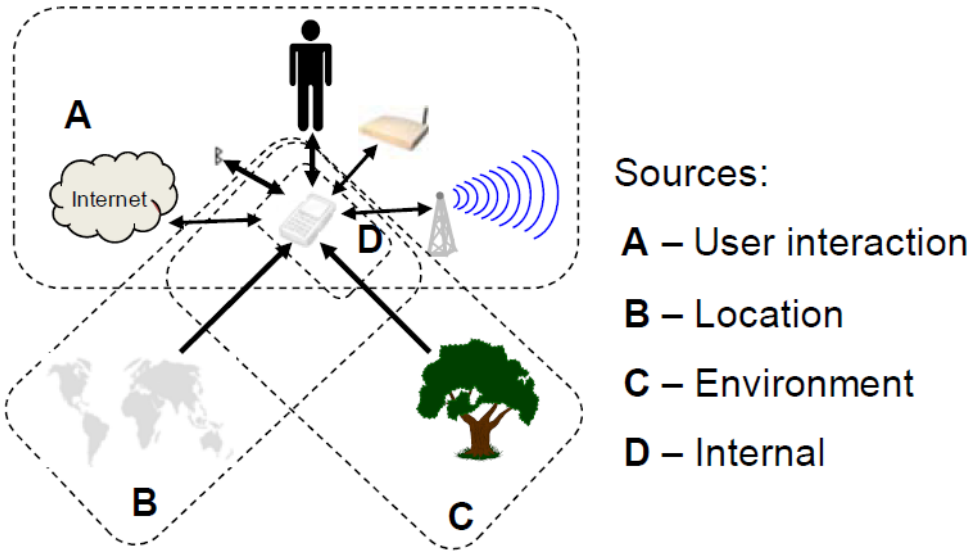


Figura 2 - Fuentes de datos de contexto

Teniendo en cuenta estas clasificaciones de fuentes y características del contexto podemos nombrar los siguientes ejemplos de datos, su contexto, alguna aplicación y la información que se puede inferir a partir de los datos (Y. Lee, S.S. Iyengar, C. Min, Y. Ju, S. Kang, T. Park, J. Lee, Y. Rhee, J. Song, 2012):

Categoría	Contexto	Aplicación	Sensor
Velocidad	Caída	Protección de estado de aplicación y de componentes del teléfono.	Acelerómetro, giroscopio, micrófono
	Movimiento	Desplazamientos de lugar	GPS, acelerómetro
Localización	Exterior	Navegación	GPS, acelerómetro
	Interior		Wifi, sensor de ultra sonido
Lugar	Humor, crowdedness, ruido, iluminación	Recomendaciones, social networking	micrófono, cámara, sensor de luz, GPS, WiFi
Actividad/Estado	Actividad	Activity-based service initiation	Acelerómetro, giroscopio, cámara, RFID (object-attached), GPS
	Gestos (del cuerpo)	Interacciónes de usuario	Acelerómetro, giroscopio, camera
	Patrón de sueño	Monitor de sueño	Acelerómetro, micrófono
	Reunión de trabajo	Control de sonidos y alertas, trabajo colaborativo	Micrófono, GPS, agenda
	Reunión informal	Visibilidad y alertas	Micrófono, GPS, agenda
Ambiente	Clima	Monitor del cliema (Weather monitoring)	Termómetro, higrómetro, anemómetro
	Temperatura	Monitor de temperatura	Termómetro, servicio del clima
Compañía	Numero de relaciones	Advertising, groupware, Trending topics	Micrófono, sensor de proximidad

Tabla 1 Datos de contexto y su aplicación

Al hacer referencia a la visión de Mark Weiser en 1999 sobre el ordenador del siglo 21, podemos resaltar la importancia que da a la forma como el ordenar puede entender el contexto de los usuarios para proporcionar servicios o aplicaciones relevantes. Esta idea ha permitido encaminar muchos trabajos de investigación a resolver el reto de desarrollar aplicaciones de este tipo y la manera óptima sobre cómo obtener y representar la información del contexto. En Ferreira, D. (2013) y haciendo referencia al Kit de herramientas de contexto (A. K. Dey, G. D. Abowd, and D. Salber, 2001) se describe un marco conceptual de referencia para desarrollo de aplicaciones sensibles al contexto. Muy claramente se define la separación de la forma como adquirir y representar el contexto, sobre su utilización en las aplicaciones sensibles al contexto. Los requisitos para tratar con el contexto son los siguientes:

- La separación de problemas: separar la forma en que se adquiere el contexto de cómo se utiliza.
- Interpretación de contexto: uso de múltiples capas de contexto debe ser transparente y proporcionado por la arquitectura.
- Comunicaciones transparentes y distribuidas: comunicación transparente entre los sensores de contexto y las aplicaciones.
- Disponibilidad permanente para la adquisición de información del contexto: los componentes que adquieren contexto se encuentre ejecutando independientemente de las aplicaciones que los utilizan.
- Almacenamiento Contexto: historia contexto se puede utilizar para establecer tendencias y predecir valores futuros de contexto.
- Descubrimiento de recursos: para una aplicación se comunique con un sensor, debe saber qué tipo de información que el sensor puede proporcionar, en el que se encuentra y cómo comunicarse con él.

También es fundamental la reutilización de contexto. Para este propósito, el marco contextual, anteriormente mencionado, propone los siguientes componentes:

- Widgets de contexto: son reutilizables bloques de construcción para la información de contexto. Esconden la complejidad de los sensores reales utilizados en la aplicación y el resumen de información del contexto que se adaptan a las necesidades previstas de la aplicación.
- Intérpretes: toman información de uno o más fuentes de contexto y producen una nueva pieza de información de contexto.
- Agregadores: recoger múltiples piezas de información de contexto relacionados de manera lógica en un repositorio común de contexto.
- Los servicios: son componentes en el marco que ejecutar acciones en nombre de las aplicaciones, sincrónica o asincrónica.
- Descubridores: son responsables de mantener un registro de lo que existen capacidades en el marco.

2.2 Machine Learning (aprendizaje automático)

Machine Learning es una rama de la inteligencia artificial que tiene como objetivo construir y estudiar los sistemas que pueden aprender a partir de datos. Como asevera Tom M. Mitchel (1997), el campo de machine learning es concerniente a la pregunta sobre cómo construir programas que automáticamente mejoren con experiencia. Para lograr esto existen tres tipos de algoritmos de aprendizaje automático: supervisado, no supervisado y aprendizaje por refuerzo (A. P. Angelbrecht ,2007).

- Las Técnicas de aprendizaje supervisado: toman ejemplos de entrada que proporcionan salidas deseadas para las entradas dadas (conjunto de entrenamiento).
- Técnicas de aprendizaje no supervisado: no requieren ejemplos de datos clasificados. El modelo se genera a partir de datos no clasificados utilizando algunos modelos matemáticos que son propensos a seguir los datos.
- Técnicas de aprendizaje por refuerzo: aprenden utilizando una indicación de la corrección al final de un razonamiento. Las siguientes secciones tratan algunas técnicas de aprendizaje supervisado a menudo se utilizan para crear modelos para predecir las clases. Esto incluye los árboles de decisión, redes neuronales, redes bayesianas, naive Bayes, vecinos más cercanos, y las máquinas de vectores soporte.

Bajo el concepto de machine learning se engloban un conjunto de técnicas y algoritmos para extraer información de unos datos o bien para estimar las dependencias o estructura conocida de un sistema, utilizando un número limitado de observaciones de entrada y salida (G. Pajares, J.M. de la Cruz, 2010). Cada uno de los algoritmos existentes es usado para resolver problemas específicos. A continuación se listan los diferentes tratamientos según las características del problema:

- Clasificación
- Regresión
- Probabilidad

Algunos de los algoritmos que logran resolver este tipo de problemas son los siguientes:

• Redes bayesianas: las redes bayesianas son grafos dirigidos acíclicos, cuyos nodos representan variables aleatorias en el sentido de Bayes: las mismas pueden ser cantidades observables, variables latentes, parámetros desconocidos o hipótesis. Las

aristas representan dependencias condicionales; los nodos que no se encuentran conectados representan variables las cuales son condicionalmente independientes de las otras. Cada nodo tiene asociado una función de probabilidad que toma como entrada un conjunto particular de valores de las variables padre del nodo y devuelve la probabilidad de la variable representada por el nodo. Por ejemplo, una red bayesiana puede representar las relaciones probabilísticas entre enfermedades y síntomas. Dados los síntomas, la red puede ser usada para computar la probabilidad de la presencia de varias enfermedades.

• Clustering: consiste en asociar la información basada en las similitudes encontradas en sus propiedades. La información que comparte las mismas características se encuentra ubicada muy cerca formando un grupo llamado cluster. Por ejemplo, contenidos en las páginas web pueden ser agrupados de acuerdo con el acceso de sus usuarios.

• Árboles de decisión (DT): es un modelo de predicción que sirve para representar y categorizar una serie de condiciones que ocurren de forma sucesiva; para la resolución de un problema. Están compuestos por un conjunto de reglas organizadas jerárquicamente de forma estratégica. Ayudan a la representación, la selección y la clasificación de los datos.

• Lógica Difusa: recomendado para tratar con datos difusos. Se basa en lo relativo de lo observado como posición diferencial. Este tipo de lógica toma dos valores aleatorios, pero contextualizados y referidos entre sí. Así, por ejemplo, una persona que mida 2 metros es claramente una persona alta, si previamente se ha tomado el valor de persona baja y se ha establecido en 1 metro. Ambos valores están contextualizados a personas y referidos a una medida métrica lineal.

• Algoritmos genéticos: en algunas investigaciones se recomiendan los algoritmos genéticos en la adaptación de los sistemas complejos, ya que son capaces de tratar problemas altamente restringidos. Son escalables para altas dimensiones y también porque en tareas complejas el espacio de características puede crecer exponencialmente con el número de características.

• Modelos de Markov: En un modelo de Markov sencillo, cada nodo enumera los posibles estados del sistema, y las transiciones reflejan probabilidades de cambiar de un estado a otro. Sólo el último estado (acción del usuario) se considera; sin embargo, en los modelos de segundo orden los dos últimos estados (acciones del usuario) también cuentan.

• Redes Neuronales: son un paradigma de aprendizaje y procesamiento automático inspirado en la forma en que funciona el sistema nervioso de los animales. Se trata de un sistema de interconexión de neuronas que colaboran entre sí para producir un

estímulo de salida. Por ejemplo, se aplican las redes neuronales para compilar un perfil de usuario mediante el uso de elementos de procesamiento para conectar nodos de entrada con nodos de salida.

• Reglas de inducción: consiste en la extracción de reglas formales basadas en la observación de los datos o los patrones locales. Aunque es el método más común adoptado para la adaptación de sensibilidad al contexto dada su simplicidad, también se ve reducida su expresividad.

Por último, cabe anotar que existen implementaciones e investigaciones sobre modelos que puede aprender (entrenarse) incrementalmente de datos frescos, sin necesidad de usar datos anteriores. Este tipo de modelos se les conoce como Clasificadores Actualizables (Updatable Classifiers). Algunos de los modelos que pueden modificarse para que cumplan esta propiedad, están NaverBayes, Markov Model, Interpolated Markov Model (H. T. Lin, N. Koul, 2011), entre otros.

Machine learning en combinación con Context-aware

Desde principios de los años noventa se ha intentado aplicar machine learning para apoyar diferentes estudios de sistemas sensibles al contexto (context-awareness). Aunque estos trabajos se han dedicado a explorar múltiples aplicaciones del machine learning para la adaptación de sensibilidad al contexto, estos se encuentran muy dispersos y cada uno se ha centrado en una aplicación específica de la adaptación, sin una visión unificada de sus beneficios potenciales (V. Genaro Motti, N. Mezhoudi, J. Vanderdonckt, 2012). Estos beneficios que proporciona machine learning se pueden evidenciar durante dos fases distintas:

1. Inferencias acerca de la información de contexto. Por ejemplo, búsqueda de patrones en las historias de interacción de usuario y datos de agrupamiento, como los perfiles de los usuarios.
2. Decisiones de diseño de adaptación de sensibilidad al contexto. Por ejemplo: predicción del comportamiento de los usuarios, automatización de tareas, aprendizaje de preferencias del usuario y la continua evolución del motor.

Es importante investigar cómo cada algoritmo de machine learning soporta fases especificas del proceso de adaptación en profundidad; hasta el momento no existe un criterio unificado que apoye la aplicación de este tipo de técnicas en context awareness de una manera amplia, cubriendo las implementaciones de propósito general. En V. Genaro Motti, N. Mezhoudi, J.

Vanderdonckt, (2012), se resumen las aplicaciones potenciales, se definen los requisitos comunes, y se destacan las principales ventas y desventajas.

2.3 Sistema de auto-adaptación de UI

En informática, un sistema auto-adaptativo, es un sistema que puede desarrollarse y aprender por sí mismo y en el que tanto sus parámetros como su estructura se adaptan al entorno en tiempo real. Este tipo de sistemas están normalmente asociados con sistemas que reciben gran cantidad de datos y cuyo modo de operación es en tiempo real (ESDIS '09, 2009). Durante las últimas décadas, la información que muchos sistemas tienen que recopilar, almacenar y tratar, está incrementando de forma exponencial. Una de las características importantes de los sistema auto-adaptativos consiste en poder ser capaces de interpretar los datos de forma eficiente. Es decir, obtener un modelo que pueda ser interpretado. Para esta interpretación suele ser necesario conocer el entorno en el que se está aplicando, pero el modelo obtenido se pretende que nos dé información sobre por qué actúa de esa manera.

Al centrar esta definición en el área de las interfaces de usuario (UI) e interacciones, encontramos que la auto-adaptación de interfaces de usuarios centrada en la atención de los usuarios, hace uso de los conceptos de tres áreas de investigación: Interacción persona-Computador (HCI), Context-Aware Computing y Considerate Computing, estando su ámbito englobado dentro de la intersección de las tres, tal como lo presenta M. Gil Pascual (2013). Ver Figura 3 - Disciplinas relacionadas.

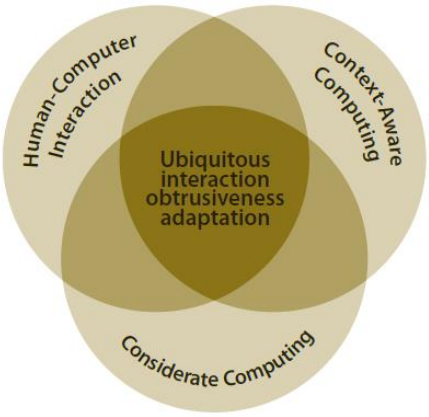


Figura 3 - Disciplinas relacionadas

En este planteamiento, la computación y los servicios deben comportarse de una manera "considerada", requiriendo la atención del usuario estrictamente cuando sea necesario, ya que la atención del usuario es un recurso valioso y limitado. Para definir su comportamiento de la interacción y la adaptación en base al nivel de molestia, se definen modelos de intromisión e interacción independientes de la tecnología. Estos modelos, posteriormente conducen la adaptación de la interacción dinámicamente por medio de una infraestructura autónoma que los usa en tiempo de ejecución. Esta infraestructura es capaz de detectar cambios en la situación del usuario (por ejemplo su localización, su actividad, etc.), planear y ejecutar modificaciones en la interacción de los servicios. Cuando se detecta un cambio del contexto del usuario, los servicios se auto-adaptan para usar los componentes de interacción más apropiados de acuerdo a la nueva situación y no molestar al usuario.

Además, como las necesidades y preferencias de los usuarios pueden cambiar con el tiempo, también es utilizada la estrategia del aprendizaje por refuerzo para ajustar los modelos de diseño iniciales, de forma que se logra maximizar la experiencia del usuario. El diseño inicial de la interacción, basado en el nivel de molestia, asegura un comportamiento inicial consistente con las necesidades de los usuarios en ese momento. Luego, este diseño se va refinando de acuerdo al comportamiento y preferencias de cada usuario por medio de su retroalimentación a través de la experiencia de uso.

Diseñando el espacio de adaptación del nivel molestia

Se define un espacio de adaptación del nivel molestia como un espacio constituido por los diferentes niveles de molestia por donde puede transitar un servicio, posicionando en un eje en 2D, las dimensiones iniciativa (en la vertical) y atención (en la horizontal).

Tal como se puede apreciar en la figura, el sistema subdivide el eje de iniciativa en dos:

- Reactivo: El usuario es quien iniciará la interacción, bien explícitamente o bien implícitamente.
- Proactivo: Los servicios iniciarán la interacción, suministrando información no solicitada.

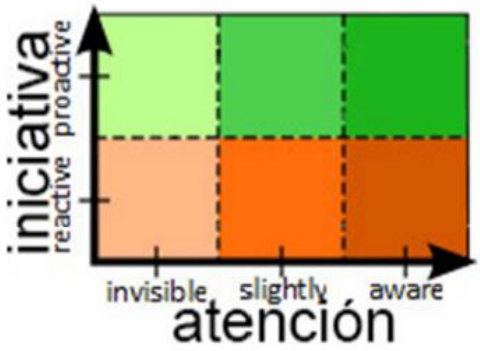


Figura 4 - Atención vs Iniciativa: espacio de adaptación de molestia

También divide el eje de atención en tres segmentos asociados con las siguientes formas de interacción:

- Invisible: El usuario no percibirá la interacción.
- Ligeramente apreciable (slightly-appreciable): El usuario puede no percibir la interacción, a menos que haga algún esfuerzo.
- Plenamente consciente (completely-aware): El usuario será plenamente consciente de la interacción, aun incluso cuando esté realizando otras tareas en el sistema.

Cada una de las intersecciones de los ejes (conformando una especie de "celda"), se corresponde con un nivel de molestia que tendrá asociada una configuración de iteración diferente. Así, dependiendo de dónde sitúen los diseñadores un servicio (en qué nivel de molestia), este interaccionara con el usuario de una forma u otra. Por lo tanto, en la etapa de diseño del sistema, los diseñadores, además de decidir qué servicios son necesarios para cada usuario según su perfil, posicionarán cada uno de estos servicios dentro de ese espacio de adaptación de molestia, en un nivel acorde con la forma de iteración deseada por el usuario, según sus necesidades y preferencias iniciales.

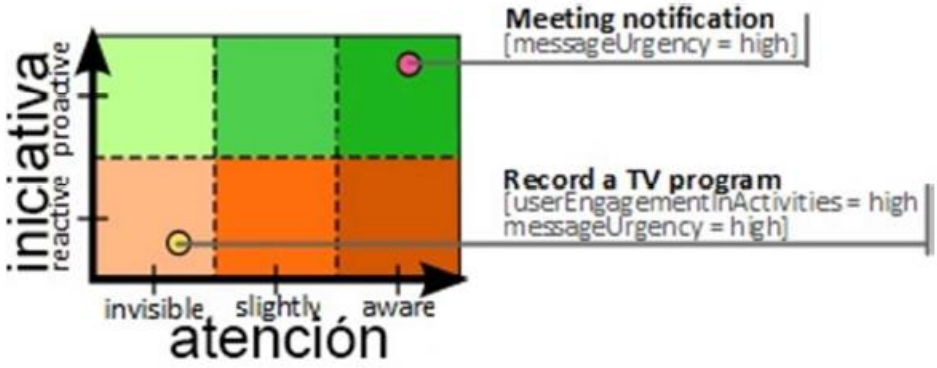


Figura 5 - Ejemplos espacio de adaptación de molestia

Por ejemplo, tal como se ilustra en la figura, un usuario puede preferir inicialmente que el servicio encargado de grabar programas de televisión, una vez que le ha encomendado una tarea de grabación, no le moleste con ningún tipo de mensaje, que simplemente haga su trabajo en un segundo plano. Para este usuario, dicho servicio se posicionaría inicialmente en los niveles de iniciativa reactiva y atención invisible. Sin embargo, el mismo usuario prefiere que su servicio de notificación de citas le avise automáticamente (por sí mismo) cada vez que tenga información que notificar y en primer plano, de modo que sea totalmente consciente para el usuario, incluso aunque este estuviera haciendo alguna otra tarea en el sistema. Por lo tanto, dicho servicio se posicionaría inicialmente en los niveles de iniciativa proactiva y atención.

Sistema de auto-adaptación de preferencias

El contexto, las preferencias y las necesidades de los usuarios evolucionan a lo largo del tiempo, siendo necesaria una auto-adaptación del nivel de molestia de cada servicio que sea coherente con dichos cambios, de manera que garantice que los servicios interactuarán con el usuario de manera satisfactoria para este. El sistema de auto-adaptación de preferencias, permite solucionar la adaptación a los cambios de contexto mediante el uso de reglas de adaptación, donde se especifica la condición que hace que un servicio cambie de un nivel a otro dependiendo del contexto del usuario.

3. Contexto Tecnológico

Este apartado busca contextualizar y describir las diferentes tecnologías utilizadas en el proyecto. Se detallan los diferentes componentes, características e información de las tecnologías usadas, así como las adaptaciones que se realizaron para la implementación.

3.1 Java

Es un lenguaje de programación, originalmente desarrollado por James Gosling de Sun Microsystems (la cual fue adquirida por la compañía Oracle) y publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems. Su sintaxis deriva en gran medida de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java son generalmente compiladas a bytecode (clase Java) que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente.

Es un lenguaje de programación de propósito general, concurrente, orientado a objetos y basado en clases que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como WORA, o "write once, run anywhere"), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra. Java es, a partir de 2012, uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de cliente-servidor de web, con unos 10 millones de usuarios reportados.

Hay que aclarar que el desarrollo de programas para Android se hace habitualmente con el lenguaje de programación Java y el conjunto de herramientas de desarrollo, pero que Android no utiliza los estándares establecidos de Java, i.e. Java SE y ME. En Android sólo se utiliza la sintáxis y la semántica de Java, pero no incorpora en su totalidad las bibliotecas de clases de Java y APIs que acompañan a Java SE o ME.

3.2 Eclipse

Eclipse es un entorno de desarrollo integrado, compuesto por un conjunto de herramientas de programación de código abierto multiplataforma. Es una potente y completa plataforma de programación, originalmente desarrollada para Java pero que, gracias a su ampliación con plugins, se ha extendido a otros lenguajes de programación como C/C++ o Python. Debido a su enorme flexibilidad, ha dado lugar a otros IDEs especializados como intelliJ IDEA o Android Studio. Adicionalmente está soportado por una gran comunidad que permite extender constantemente las áreas de aplicación y está en constante evolución.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge, pero ahora es desarrollado por la Fundación Eclipse, la cual es una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios, según se puede observar en su sitio oficial[3]. Eclipse fue liberado originalmente bajo la Common Public License, pero después fue re-licenciado bajo la Eclipse Public License.

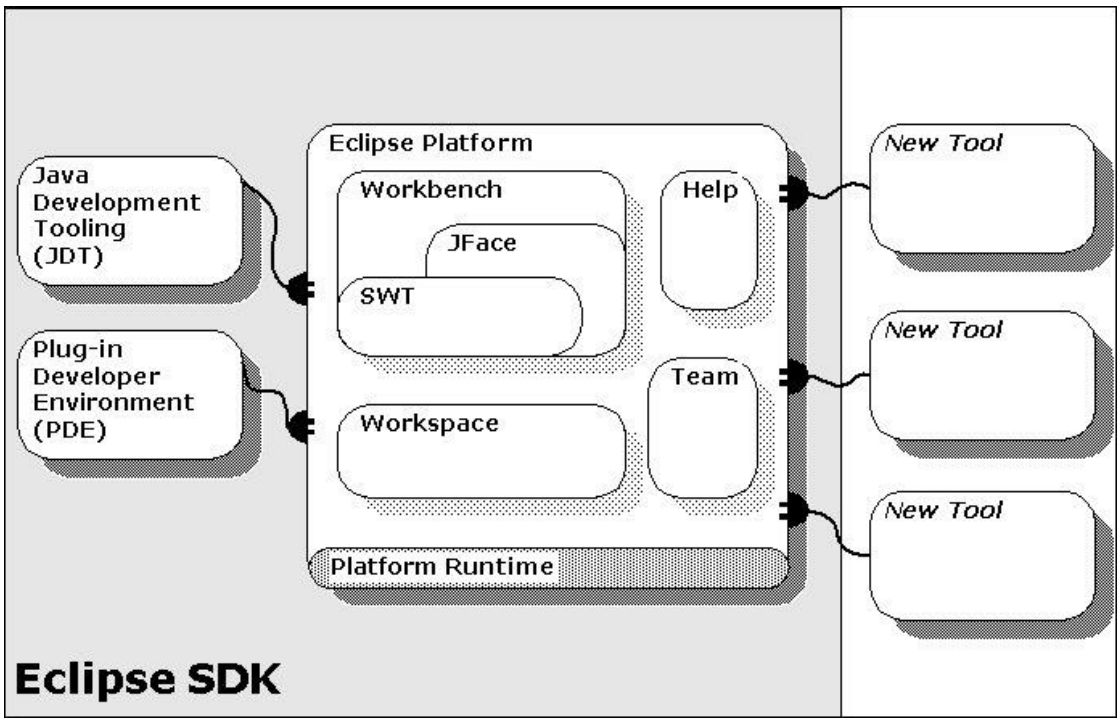


Figura 6 - Arquitectura de Eclipse [4]

[3] http://www.eclipse.org/

[4] http://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Fguide%2Farch.htm

Como se puede observar en la Figura 6 - Arquitectura de Eclipse que describe la arquitectura del IDE, Eclipse es altamente desacoplado, es configurable y se puede integrar con diversos plugins o SDK's desarrollados por terceros. En el caso del desarrollo e implementación del proyecto, se utilizó Eclipse como IDE en conjunto con el SDK de Android, del cual posteriormente se hablará.

3.3 Android

Android (Burnete, 2009) es un sistema operativo creado por Google orientado principalmente a dispositivos móviles aunque cada vez son más intensos los rumores que van a aparecer en otros dispositivos como ordenadores. El proyecto no fue creado por Google, sino por la compañía Android Inc., que posteriormente fue comprada por Google. Android fue presentado en el 2007 junto a la fundación Open Handset Alliance (OHA)5. El día 23 de Septiembre del año 2008 Google lanzó su primera versión del sistema operativo, basado en una versión del Kernel 2.6 de Linux y un mes más tarde se vendió el primer móvil con este sistema operativo. El sistema operativo está diseñado principalmente para dispositivos móviles con pantalla táctil como Smartphone o tablets.

La licencia del sistema operativo es software libre. Además de esto es open source, con lo cual es posible acceder al código fuente y modificarlo creando así las ROMs personalizadas, utilizadas por usuarios expertos para personalizar sus dispositivos, o por las distintas compañías que trabajan con el creando una capa superior al núcleo para así intentar sacar el rendimiento máximo a los dispositivos.

El desarrollo del sistema no es sólo asunto de Google, también colabora OHA. Este consorcio está formado por multinacionales en el mundo de la tecnología como Samsung, LG, Nvidia o HTC. El objeto principal de este es desarrollar estándares del campo de la telefonía y Android es uno de ellos.

Para desarrollar en Android es necesario reunir todos los elementos de entorno y el control de las aplicaciones. Como lenguaje de desarrollo se utiliza Java y se proporciona un plugin para eclipse que contiene todas las herramientas necesarias. En este plugin se incluye un SDK que facilita al máximo el trabajo e intenta evitar las malas prácticas dentro de lo posible. Una de las principales ventajas es que cualquier dispositivo Android puede ejecutar cualquier aplicación (siempre y cuando la versión instalada sea compatible) ya que es un sistema operativo multiplataforma.

Arquitectura Android

Para empezar a desarrollar aplicaciones debemos conocer como es la estructura de este sistema operativo (ver Figura 7 - Arquitectura Android). Para empezar diremos que está formado por varias capas o niveles, facilitando así la tarea al programador. Además, esta distribución permite acceder a las capas más bajas mediante librerías excluyendo así la necesidad de programar a bajo nivel las funcionalidades necesarias para utilizar las componentes hardware del dispositivo.[5]

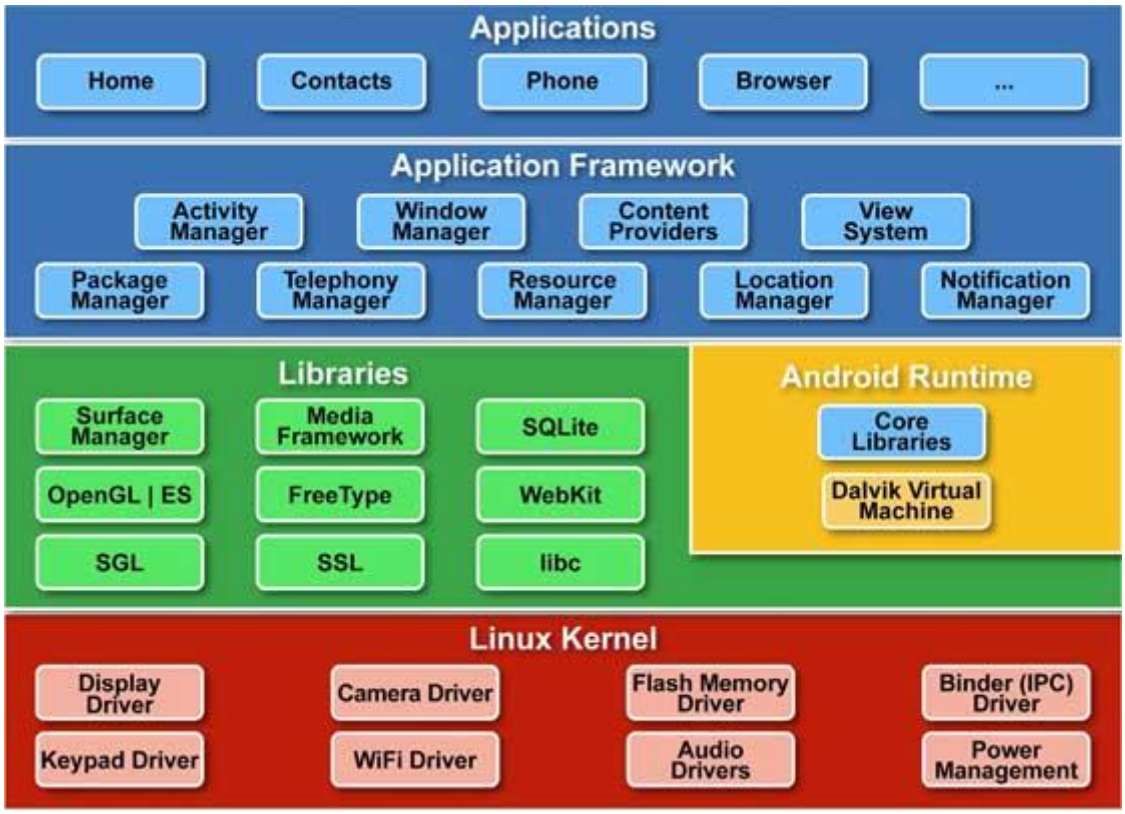


Figura 7 - Arquitectura Android

Cada capa utiliza elementos de la capa inferior para realizar funciones, este tipo de arquitectura es conocida como arquitectura pila, (Brahler, 2010).

A continuación se van a explicar las diferentes capas siguiendo una dirección ascendente:

[5] http://developer.android.com/intl/es/guide/components/index.html

1. Kernel de Linux: El núcleo actúa como capa de abstracción entre el hardware y el resto de las capas. El desarrollador no accede directamente a esta capa, sino se deben utilizar librerías disponibles en capas superiores. Uno de los motivos de utilizar estas librerías es la de evitarnos el hecho de conocer las características precisas de cada teléfono. Para cada elemento de hardware existe un driver (controlador) dentro del kernel que permite su utilización. El kernel también se encarga de gestionar los diferentes recursos del teléfono y del sistema operativo en sí.

2. Bibliotecas: Es la capa que siguiente al kernel. Esta capa está formada por las capas nativas de Android. Están escritas en C, C++ y compiladas para la arquitectura hardware especifica del teléfono. El objetivo de estas es proporcionar funcionalidad a las aplicaciones para tareas que se repiten con frecuencia, evitando que se tengan que codificar una y otra vez garantizando una mejor mantenibilidad y una mejor eficiencia.

Entre las librerías se encuentra:

- OpenGL: motor gráfico 3D basado en las APIs de OpenGL. Si el teléfono proporciona aceleración gráfica es utilizada, en caso contrario se utiliza un motor software altamente optimizado.
- Gestor de superficies (Surface Manager): se encarga de componer las imágenes que se muestran en la pantalla a partir de capas gráficas 2D y 3D. Cada vez que la aplicación pretende dibujar algo en la pantalla, la biblioteca no lo hace directamente sobre ella. En vez de eso, realiza los cambios en imágenes que almacena en memoria y que después combina para formar la imagen final que se envía a la pantalla.
- Bibliotecas Multimedia: basadas en OpenCore, permiten visualizar, reproducir e incluso grabar numerosos formatos de imagen, video y audio.
- Webkit: motor web utilizado por el navegador. Es el mismo motor que utiliza Google Chrome y Safari.
- SSL (Secure Sockets Layer): proporciona la seguridad necesaria al acceder a Internet por medio de criptografía.
- FreeType: permite mostrar fuentes tipográficas, basadas tanto en mapas de bits como en vectores.
- SQLite: motor de BBDD relacional, disponible para todas las aplicaciones.
- SGL: desarrollada por Skia, utilizada tanto en Android como en Google Chrome, se encarga de representar elementos de dos dimensiones, Es el motor gráfico 2D de Android.
- Biblioteca C de sistema (libc): está basada en la implementación de Berkeley Software Distribución (BSD), pero optimizada para sistemas Linux embebidos. Proporciona funcionalidad básica para la ejecución de las aplicaciones.

3. Entorno de ejecución: Como podemos ver en la Figura 7 - Arquitectura Android, el entorno de ejecución se encuentra dentro la capa de librerías. En este lugar se encuentran las librerías con las funcionalidades habituales de Java así como otras específicas.

El componente principal del entorno es la máquina virtual Dalvik. Las aplicaciones se codifican en Java y son compiladas en un formato específico para que la máquina virtual las ejecute. La ventaja de esto es que las aplicaciones se compilan una única vez y de esta forma estarán listas para distribuirse con la garantía de serán ejecutables en cualquier dispositivo que disponga de una versión Android compatible.

Dalvik es una variación de la máquina virtual de Java, por lo que no es compatible con el bytecode Java. Java se usa como lenguaje de programación, y los ejecutables que genera el SDK de Android tienen la extensión .dex que es específico para Dalvik, por ello no se pueden ejecutar aplicaciones Java en Android ni viceversa.

4. Framework de aplicaciones: La siguiente capa está formada por todas las clases y servicios que utilizan directamente las aplicaciones para realizar sus funciones. La mayoría de los componentes son librerías Java que acceden a través de la máquina virtual.
Siguiendo el diagrama encontramos:

- Activity Manager: Encargado de administrar la pila de actividades de nuestra aplicación así como su ciclo de vida.
- Windows Manager: Se encarga de organizar lo que se mostrará en pantalla que posteriormente pasarán a ser ocupadas por las actividades.
- Content Provider: Esta librería es la encargada de crear una capa de encapsulación que se compartirá entre aplicaciones para tener control sobre cómo se accede a la información.
- Views: Son las encargadas de construir las interfaces de usuarios, están formadas por elementos simples como botones, cuadros de texto, listas o elementos más complejos como navegadores web o visor de Google Maps.
- Notification Manager: Engloba los servicios para notificar al usuario cuando algo requiera su atención mostrando alertas en la barra de notificaciones. Esta biblioteca permite utilizar sonidos, activar las vibraciones, o los LEDs de notificación si el dispositivo dispone de ellos.
- Package Manager: Esta biblioteca da la información sobre los paquetes instalados en el dispositivo además de gestionar la instalación de nuevos. Estos paquetes tienen el archivo .apk, que a su vez incluyen los archivos .dex con todos los recursos y archivos adicionales que necesite la aplicación, para facilitar su descarga e instalación.
- Telephony Manager: Librería encargada de tratar todo lo relacionado con las llamadas, los SMS/ MMS, esta librería no permite reemplazar o eliminar la actividad que se muestra cuando una llamada está en curso.

- Resource Manager: Con esta librería gestionamos todos los elementos que forman la aplicación pero están fuera del código, es decir, traducciones, imágenes, sonidos, layouts.
- Location Manager: Permite determinar la posición geográfica del dispositivo mediante GPS, triangulación u otras técnicas para poder trabajar con mapas.
- Sensor Manager: permite manipular los elementos de hardware del teléfono como el acelerómetro, giroscopio, sensores de luminosidad, campo magnético, presión, proximidad, temperatura….
- Cámara: Con esta librería podemos hacer uso de todas las cámaras que tenga el dispositivo, para tomar instantáneas o grabar videos.
- Multimedia: Permiten reproducir y visualizar audio, video, imágenes.

5. Aplicaciones: En la última capa se incluyen todas las aplicaciones del dispositivo, tanto las que tienen interfaz como las que no, las nativas y las administrativas, las que vienen preinstaladas en el dispositivo y aquellas que el usuario ha decidido instalar.

En esta capa encontramos también la aplicación principal del sistema: el lanzador (launcher), porque es la que permite ejecutar otras aplicaciones mediante una lista y mostrando diferentes escritorios donde podemos colocar accesos directos a aplicaciones o widgets.

Como podemos observar esta separación por niveles proporciona un entorno sumamente poderoso para que podamos programar aplicaciones que hagan cualquier cosa. Android es accesible y podemos jugar siempre con las aplicaciones de nuestro teléfono para optimizar cualquier tarea. Uno de los potenciales de Android es que permite al usuario el control total de su dispositivo para poder crear un entorno a medida.

Activities

Una Activity es cada una de las interfaces de usuario que podemos encontrarnos en una aplicación Android, es decir, es como los sistemas Android representan las interfaces graficas con las que el usuario del sistema podrá interactuar. Para crear una Activity únicamente debemos extender de la clase Activity proporcionada por Android.
Con lo que hemos visto anteriormente podemos decir qué una aplicación está formada por un grupo o un conjunto de Activities relacionadas entre sí. Cuando tenemos una aplicación disponemos de una Activity principal que es la primera que se lanza cuando ejecutamos la aplicación. Esta Activity principal podrá darnos acceso a otras. Es muy importante saber que cuando se inicializa una Activity se almacena en una pila con el algoritmo LIFO6y la Activity anterior pasa a un estado parada.

Como hemos visto una Activity dispone de muchos estados, por ello existe un conjunto de métodos que se pueden sobrecargar en cualquier momento, para realizar acciones en cada uno de los estados en los que se encuentre.

Una Activity tiene un ciclo de vida definido, el cual comprende los siguientes estados (ver Figura 8 - Ciclo de vida de un Activity en Android):

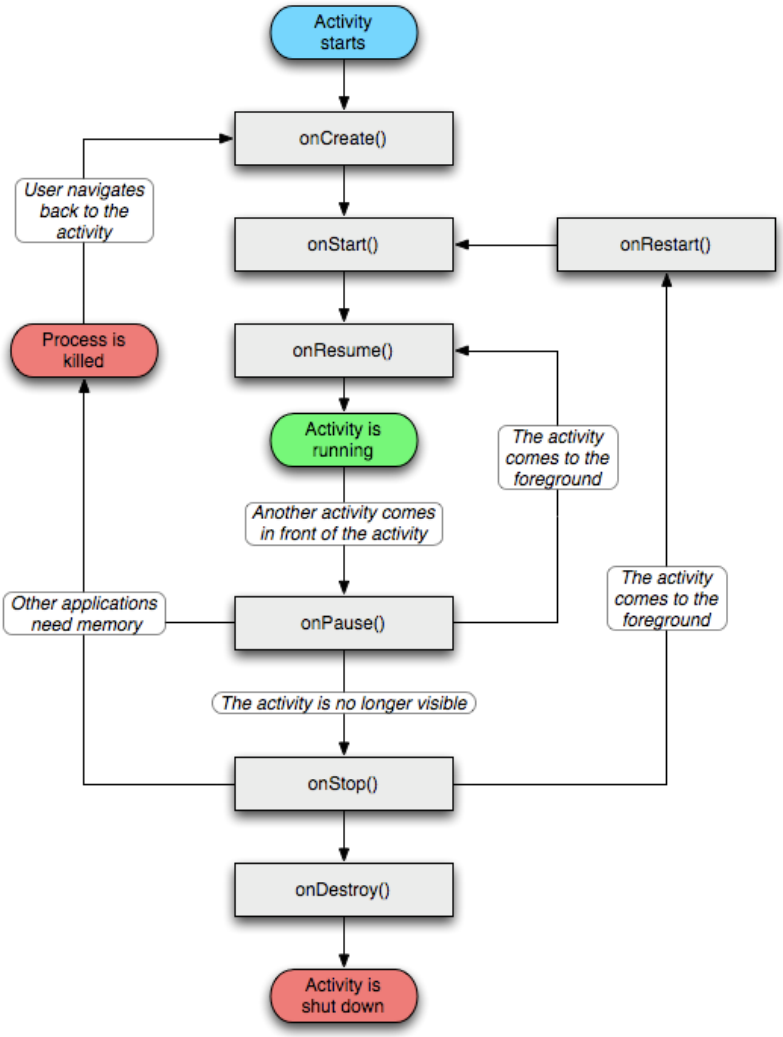


Figura 8 - Ciclo de vida de un Activity en Android

- onCreate(): Se llama cuando se ejecuta una Activity por primera vez.
- onRestart(): Se llama después de que una Activity haya sido parada, justo antes de que se inicialice de nuevo, y siempre seguido de onStart().

- onStart(): Se llama siempre antes de que la Activity se haga visible al usuario. Seguida por onResume(); si la Activity a primer plano o por onStop(); si va a permanecer oculta.
- onResume(): Se llama justo antes de que la Activity comience la interacción con el usuario. En este punto, la Activity está en la cima de la pila. Siempre va seguida de onPause().
- onPause(): Se llama cuando el sistema va a reanudar otra Activity. En este método se guardan datos, se paran animaciones u otras cosas que puedan consumir CPU. Debe ser un método rápido ya que la siguiente Activity no se reanuda hasta que esta no termine sus acciones. Es seguida por onResume(); si va a primer plano, o por onStop(); si se hace invisible al usuario.
- onStop(): Se llama cuando la Activity y no va a ser visible al usuario bien por que vaya a ser destruida o bien porque va a pasar a un segundo plano. Va seguida de onRestart(); si vuelve a ser visible para el usuario, o por onDestroy(); si va a desaparecer.
- onDestroy(): Se llama antes de que la Activity se destruya. Es la última llamada que recibe una Activity. Puede llamarse por dos motivos, que la Activity haya sido finalizada, o por que el sistema necesite espacio y decida eliminarla.

Services

Un servicio es un componente que mejora el rendimiento de operaciones largas en segundo plano y no provee una interfaz de usuario.
Otro componente puede arrancar un servicio y este continuara en segundo plano incluso si el usuario cambia a otra aplicación. Además un componente puede enlazar con un servicio para interactuar con el incluso realizar interprocesos de comunicación. Por ejemplo un servicio puede soportar transacciones de red, reproducir música, todo desde el backgroud.

Un servicio puede adoptar dos formas:

- Started: Un servicio esta “started” cuando un componente lo lanza ejecutando una llamada a “startService”. Una vez lanzado, un servicio puede correr en segundo plano indefinidamente, incluso si el componente que lo ha lanzado se destruye. Normalmente, un servicio en ejecución realiza operaciones simples y no devuelve ningún resultado al componente que lo ha inicializado.
- Bound: Un servicio esta “bound”, limitado, cuando un componente se une a él haciendo una llamada al método bindService().

Content Provider

Un Content Provider, es el encargado de gestionar un conjunto de datos de la aplicación, que se han de compartir. Los datos pueden ser almacenados en: un sistema de archivos, una base de datos SQLite, en la Web o en cualquier otro lugar persistente, a la que la aplicación pueda acceder.
Mediante un Content Provider, otras aplicaciones pueden hacer consultas o incluso modificar los datos (si el Content Provider lo permite). Por ejemplo, el sistema Android proporciona un Content Provider que gestiona la información de los contactos del usuario; por tanto, una aplicación que tenga los permisos adecuados, podrá consultar parte del Content Provider (como ContactsContract.Data), para leer y escribir información sobre una determinada persona. Los Content Provider también son adecuados para leer y escribir datos que son propios de una aplicación, y no deben ser compartidos.
Un Content Provider se ha de implementar como una subclase de ContentProvider, y debe implementar un conjunto estándar de APIs, que habilitan otras aplicaciones para ejecutar transacciones.
Cuando se desea acceder a los datos de un Content Provider, se ha de hacer mediante el uso de un objeto Content Resolver, que esté dentro del contexto de la aplicación, para realizar la comunicación con el Provider como un cliente. El funcionamiento es el siguiente: el objeto ContentResolver se comunica con el objeto Provider (una instancia de la clase que implementa ContentProvider). Por su parte el objeto Provider recibe las solicitudes de los datos desde los clientes, ejecuta las acciones solicitadas y devuelve los resultados.

Broadcast Receivers

Un Broadcast Receiver, es un componente que responde a la emisión de anuncios/notificaciones en todo el sistema. Existen muchos Broadcasts que los origina el sistema, como por ejemplo: Broadcasts que avisan que la pantalla se ha apagado, que la batería está baja, o que se ha capturado una imagen. Las aplicaciones por su parte también pueden iniciar Broadcasts, por ejemplo, para informar a otras aplicaciones que ya se han descargado los datos en el dispositivo, y pueden ser usados.
Aunque los Broadcast Receivers no muestran una interfaz de usuario, estos pueden crear una notificación StatusBar, para alertar al usuario cuando ocurra un evento Broadcast. Sin embargo, los Broadcast son usados más comúnmente, como una puerta (“Gateway”) a otros componentes, y tienen como propósito, realizar el menor trabajo posible. Por ejemplo, este puede instanciar un Service, para que ejecute algún trabajo basado en el evento.
Un Broadcast Receiver se implementa como una subclase de BroadcastReceiver, y cada Broadcast se envía mediante un Intent.

Manifest

El manifest en un fichero XML que se ha de crear para cada aplicación, dicho fichero se debe llamar: AndroidManifest.xml. Éste contendrá, la declaración de cada uno de los componentes de los que está compuesta la aplicación a la que pertenece. Otras de las cosas que se han de declarar en el manifest son:

- Todos los permisos de usuario que requiera la aplicación, por ejemplo, acceso a Internet, acceso para la lectura de los contactos del usuario, etc.
- El API Level mínimo requerido por la aplicación, esto es, la versión mínima de Android, para la cual la aplicación es operativa, Código 3.5.
- Las características de hardware y software, requeridos por la aplicación (Cámara, Bluetooth, etc.).
- Librerías API que se deben enlazar con la aplicación (diferentes al API de Android), como por ejemplo, las librerías de Google Maps.

Fragment

Como es conocido en los últimos años han aparecido dispositivos de gran tamaño, tipo tablets, por ello el equipo de desarrollo de Android tuvo que solucionar el problema de la adaptación de la interfaz gráfica de las aplicaciones a ese nuevo tipo de pantallas. Una interfaz de usuario diseñada para un dispositivo móvil no se adaptaba a pantallas de 7 pulgadas o superiores. La solución a esto vino en forma de un componente, llamado Fragment.

Un Fragment podría definirse como una porción de la interfaz de usuario que puede añadirse o eliminarse de una interfaz de forma independiente al resto de elementos de la actividad, y que por supuesto puede reutilizarse en otras actividades. Esto permite poder dividir nuestra interfaz en varias porciones de forma que podamos diseñar diversas configuraciones de pantalla, dependiendo de su tamaño y orientación, sin tener que duplicar código en ningún momento, sino tan solo utilizando o no los distintos fragmentos para una de las posibles configuraciones.

Android SDK

El SDK (Software Development Kit) de Android, incluye un conjunto de herramientas de desarrollo. Comprende un depurador de código, biblioteca, un simulador de teléfono basado en QEMU, documentación, ejemplos de código y tutoriales. Las plataformas de desarrollo soportadas incluyen Linux (cualquier distribución moderna), Mac OS X 10.4.9 o superior, y

Windows XP o superior. La plataforma integral de desarrollo (IDE, Integrated Development Environment) soportada oficialmente es Eclipse junto con el complemento ADT (Android Development Tools plugin), aunque también puede utilizarse un editor de texto para escribir ficheros Java y Xml y utilizar comandos en un terminal (se necesitan los paquetes JDK, Java Development Kit y Apache Ant) para crear y depurar aplicaciones. Además, pueden controlarse dispositivos Android que estén conectados (e.g. reiniciarlos, instalar aplicaciones en remoto).

Las actualizaciones del SDK están coordinadas con el desarrollo general de Android. El SDK soporta también versiones antiguas de Android, por si los programadores necesitan instalar aplicaciones en dispositivos ya obsoletos o más antiguos. Las herramientas de desarrollo son componentes descargables, de modo que una vez instalada la última versión, pueden instalarse versiones anteriores y hacer pruebas de compatibilidad.

Una aplicación Android está compuesta por un conjunto de ficheros empaquetados en formato .apk y guardada en el directorio /data/app del sistema operativo Android (este directorio necesita permisos de superusuario, root, por razones de seguridad). Un paquete APK incluye ficheros .dex (ejecutables Dalvik, un código intermedio compilado), recursos, etc.

3.4 Weka

Weka (Waikato Environment for Knowledge Analysis, en español “entorno para análisis del conocimiento de la Universidad de Waikato”) es una plataforma de software para el aprendizaje automático y la minería de datos escrito en Java y desarrollado en la Universidad de Waikato. Weka es software libre distribuido bajo la licencia GNU-GPL (Witten, I., Frank, E., 2005).

Este software tiene como base el formato ARFF (attribute-relation file format), el cual separa cada uno de los atributos por coma, por lo que los espacios en blanco que originalmente tenía la BD los sustituimos por una coma, además agregamos la siguientes líneas en la cabecera de la BD.

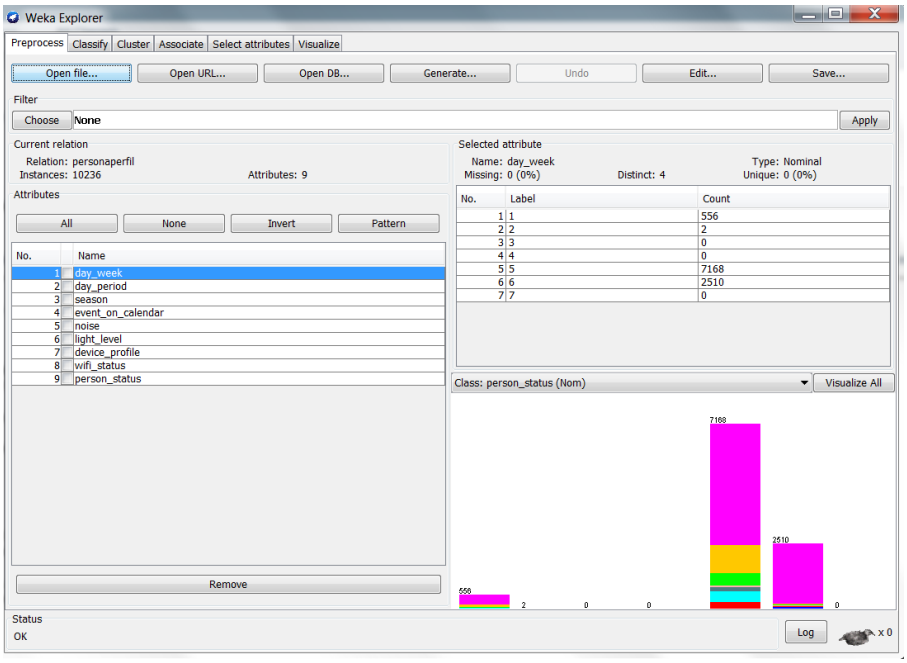


Figura 9 - UI Explorer Weka

Algunas de las características importantes de Weka son:

- Está disponible libremente bajo la licencia pública general de GNU.
- Es muy portable porque está completamente implementado en Java y puede correr en casi cualquier plataforma.
- Contiene una extensa colección de técnicas para pre-procesamiento de datos y modelado.
- Es fácil de utilizar por un principiante gracias a su interfaz gráfica de usuario.

Weka soporta varias tareas estándar de minería de datos, especialmente, pre-procesamiento de datos, clustering, clasificación, regresión, visualización, y selección. Todas las técnicas de Weka se fundamentan en la asunción de que los datos están disponibles en un fichero plano (flat file) o una relación, en la que cada registro de datos está descrito por un número fijo de atributos (normalmente numéricos o nominales, aunque también se soportan otros tipos). En la interfaz “Explorer” provista por Weka, se puede tener acceso a estás funcionalidades (Ver Figura 9 - UI Explorer Weka).

Weka también proporciona acceso a bases de datos vía SQL gracias a la conexión JDBC (Java Database Connectivity) y puede procesar el resultado devuelto por una consulta hecha a la base de datos. No puede realizar minería de datos multi-relacional, pero existen aplicaciones

que pueden convertir una colección de tablas relacionadas de una base de datos en una única tabla que ya puede ser procesada con Weka.

Weka proporciona implementaciones de algoritmos que pueden ser fácilmente aplicados a un conjunto de datos de aprendizaje. También incluye una variedad de herramientas para la transformación de conjuntos de datos, como los algoritmos para discretización, que son conocidos como unas de las más importantes tareas de pre-procesamiento de datos en la minería de datos. Es posible con WEKA procesar previamente un conjunto de datos, alimentar un esquema de aprendizaje y analizar el rendimiento del clasificador resultante.

El API del Weka está organizado en una estructura jerárquica, y los algoritmos están delimitados por su relevancia para las clases de tareas de minería de datos tales como se presenta en la Figura 10 - Taxonomía de los algoritmos del API de Weka.

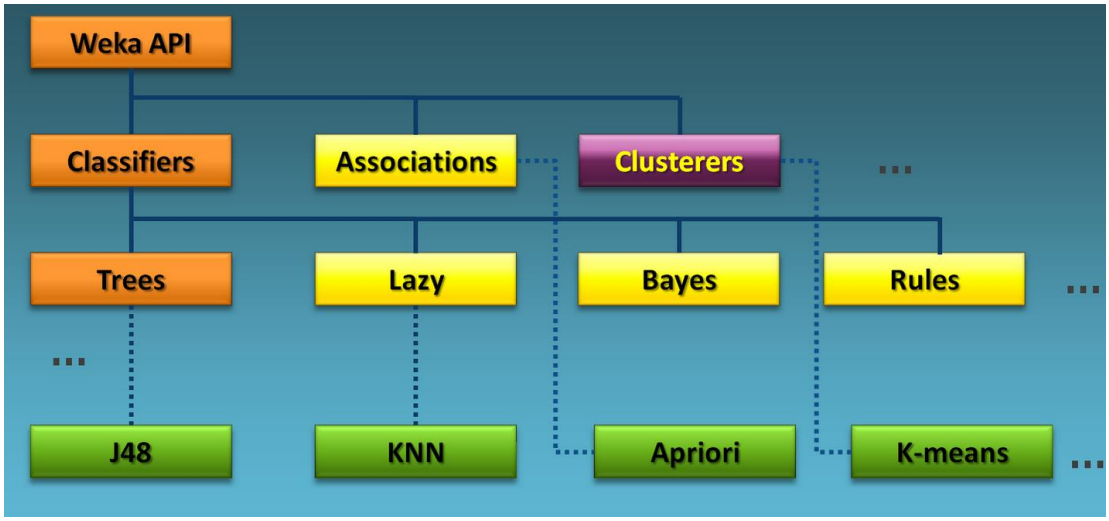


Figura 10 - Taxonomía de los algoritmos del API de Weka[6]

Adicionalmente a todo esto, Weka provee un módulo llamado "Knowledge Flow" que habilita una interfaz gráfica para diseñar flujos de trabajo e implementar el proceso de data mining de una manera integrada. Además es posbile integrarlo con tareas ETL o de transformación de datos. Ver Figura 11 - UI de Knowledge Flow de Weka.

[6] http://blog.khaledtannir.net

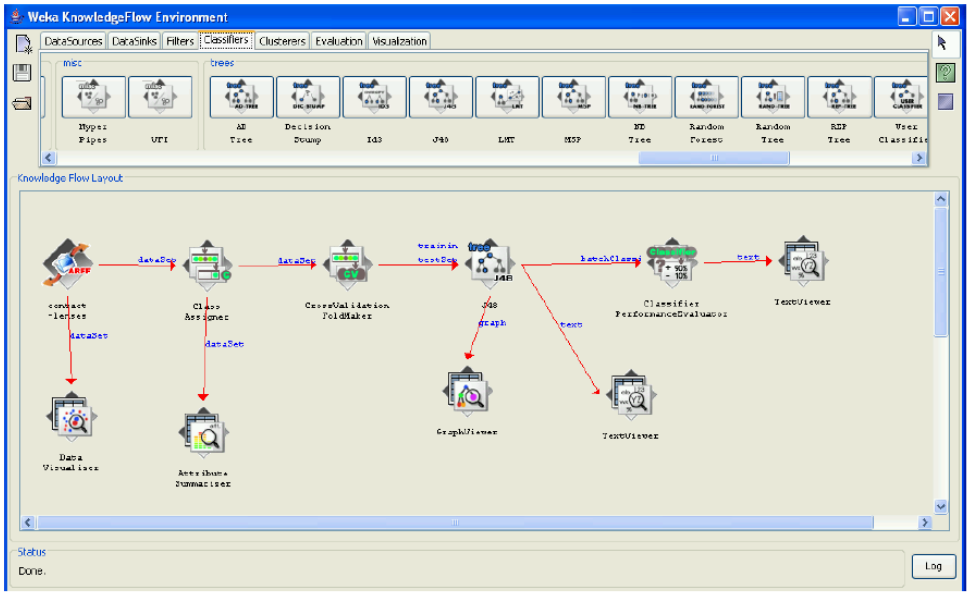


Figura 11 - UI de Knowledge Flow de Weka

4. Trabajos relacionados

Presentados los conceptos generales y las tecnologías involucradas en el proyecto, los cuales son importantes para entender el desarrollo del mismo, en este capítulo se presentan algunos resultados de investigación y trabajos desarrollados en los ámbitos de Mobile Context-awareness y el uso de machine learning en dispositivos móviles. También se analizan las funcionalidades y bondades del framework AWARE que es base importante de la solución planteada.

4.1 Mobile Context-awareness

La construcción de sistemas sensibles al contexto es una tarea compleja y requiere mucho tiempo debido a la falta de apoyo a la infraestructura adecuada (G. Chen, D. Kotz, 2000). La revisión de la literatura demuestra una vez más cómo la investigación sobre context-awareness se encuentra muy fragmentada, ya que los investigadores se centran en un dominio de experiencia específica. Los desarrolladores de aplicaciones tienen dificultades para recoger y reutilizar el contexto de alto nivel, y los usuarios experimentan aplicaciones sensibles al contexto mal diseñadas.

La trabajos existentes revisados, sugieren que los datos de contexto pueden estar basados en hardware, software o directamente provistos por los humanos (Ferreira, D., 2013). Los datos que provienen del harware, generalmente son provenientes de sensores incorporados en los dispositivos móviles o sensores externos que se encuentran en el ambiente. La información proveniente de software puede ser provista también por sensores incluidos en la red, algoritmos basados en sensores, derivaciones de la fusión de otros sensores (combinación de datos) o configuraciones/preferencias provenientes del sistema operativo. Por último, la fuente de datos basada en humanos, es tradicionalmente datos cualitativos o de captura que generalmente son difíciles o imposibles de capturar por los sensores de hardware o software.

Ferreira, D. (2013), hace un resumen general de las herramientas de investigación y trabajos disponibles en materia de gestión de contexto móvil (Tabla 2 Trabajos relacionados con

Context-awareness). Así mismo, se destacan las herramientas más relevantes y se resume las habilidades en materia de gestión del contexto.

Tabla 2 Trabajos relacionados con Context-awareness

Middleware	Audience			Sensing			Management		Properties		
	R	D	U	HW	SW	H	Centralized	Distributed	Shared	Dynamic	Scalable
CORTEX	x			x	x		x		x		x
ContextStudio			x	x	x	x	x		x	x	
ContextPhone	x		x	x	x	x		x	x		
AWARENESS	x			x	x			x	x	x	
Momento	x			x	x	x		x		x	
MyExperience	x			x	x	x		x			
CenceMe			x	x	x			x		x	
EmotionSense	x			x	x		x			x	
Empath	x		x	x	x			x			x
Funf	x		x	x	x		x		x		x
Ginger.io	x		x	x	x	x		x	x		
SystemSens	x			x	x			x			x
ohmage	x		x	x	x	x	x		x		
ODK Sensors		x		x				x	x	x	x
AWARE	x	x	x	x	x	x	x	x	x	x	x

R – Researcher, D – Developer, U – User, HW – Hardware, SW – Software, H – Human

Por otro lado, en A. K. Dey, G. D. Abowd, and D. Salber (2001) se describe un kit de herramientas de contexto, el cual es un marco de trabajo para la creación de prototipos de aplicaciones sensibles al contexto (context-aware). En su Tesis Doctoral, define una arquitectura para la creación de prototipos de aplicaciones sensibles al contexto, basándose en un corredor de contexto que utiliza ontologías para la representación y modelación del contexto. Esto consiste en una base de conocimientos del contexto, un motor de razonamiento del contexto, un módulo de adquisición del contexto y otro de manejo de la privacidad.

En M. Baldauf, S. Dustdar, F. Rosenberg (2007) se muestra un estudio sobre los distintos sistemas sensibles al contexto que han derivado principios arquitectónicos comunes dependiendo de factores como:

- Tipos de sensores: locales o remotos,
- Cantidad de usuario: sistema tiene muchos o pocos usuarios
- Recursos disponibles en el dispositivo
- Extensibilidad del sistema.

En este estudio se propone una estructura de capas para soportar los sistemas sensibles al contexto el cual consiste (desde abajo hacia arriba) en una en una "*Sensor layer" o* capa de sensores que gestiona los sensores físicos, virtuales y lógicos, una "*Raw data retrieval layer* "

(capa de recuperación de datos sin procesar) responsable de proporcionar la capa anterior con los datos del sensor, una "*Preprocessing layer*" (capa de pre-procesamiento), que es opcional, y permite para mejorar la calidad de los datos del sensor a través del razonamiento y la interpretación, una "*Storage and management layer*" (capa de almacenamiento y gestión) encargada de organizar y proporcionar los datos recogidos a clientes a través de una interfaz pública y una "*Application Layer*" (capa de aplicación) que proporciona una API de alto nivel para aplicaciones sensibles al contexto. Esta arquitectura se puede observar en la Figura 12 - Arquitecturas por capas de Baldauf y Miraoui.

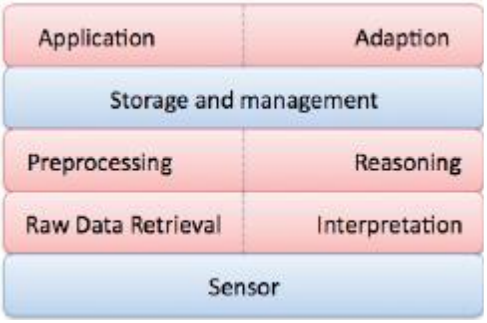


Figura 12 - Arquitecturas por capas de Baldauf y Miraoui

Por otra parte M. Miraoui, C. Tadj, C. b. Amar (2008), llevó a cabo un estudio similar con especial atención a la computación generalizada, enfocándose en el nivel de abstracción del contexto, el modelo de comunicación, el sistema de razonamiento, la extensibilidad y la reutilización. También describe una arquitectura de capas similar a la anteriormente mencionada, donde la segunda y tercera capa de la parte inferior se modifican de la forma que la capa de recuperación de datos sin procesar es sustituida por una capa de interpretación que transforma los datos del sensor sin procesar en información útil del contexto, y la capa de pre-procesamiento se sustituye por una capa de razonamiento para interpretar, predecir y deducir la información del sensor de la capa de abajo y producir información de contexto en un nivel superior. Adicionalmente la capa de Adaptación, que es responsable de la adaptación del sistema al cambiar el entorno, sustituye la capa de aplicación.

4.2 Aware Framework

AWARE es un framework de instrumentación de Android para el registro, el intercambio y la reutilización de contexto móvil, desarrollado por el grupo de investigación "Community Imaging research group" de la Universidad de Oulu en Finlandia. Este framework es abierto, escalable y está soportado por científicos de diversas áreas que lo usan para recolectar

información en campo. AWARE captura datos (Figura 13 - Framework AWARE) de hardware, software y origen humano. Adicionalmente este framework permite abstraer eventos y generar nuevos plugins para capturar nueva información de contexto o interacciones de usuario.

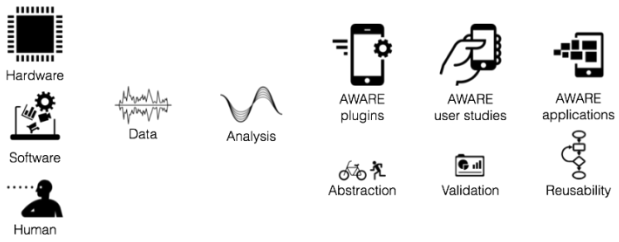


Figura 13 - Framework AWARE[7]

AWARE reduce la carga y complejidad en la construcción de herramientas para la captura y registro de datos de contexto móvil, aumentando la reutilización no solo de código y herramientas previamente desarrolladas (plugins), sino también de los resultados de investigación. Esto se logra mediante la encapsulación de diferentes retos de la investigación en soluciones o herramientas generadas como resultado de investigación, que son reutilizables para aplicaciones sensibles al contexto móvil que se pretendan desarrollar (Ferreira, D., 2013).

La instrumentación es, por definición, una colección de instrumentos considerados y supervisados colectivamente. Middleware es una capa de software que se encuentra por encima del sistema operativo y por debajo de la capa de aplicación y abstrae la heterogeneidad del medio ambiente subyacente. AWARE aprovecha el hardware, software y sensores humanos en los teléfonos móviles para detectar el medio ambiente, el contexto y las interacciones de su usuario. Más importante aún, entonces comparte este contexto móvil, facilitando así el acceso y la reutilización de contexto móvil a otros investigadores, desarrolladores y para los usuarios con aplicaciones móviles sensibles al contexto.

Podemos concluir que AWARE es un middleware de instrumentación del contexto móvil que se centra en:

- Capturar el contexto (en hardware, en software, sensores basados en humanos),
- Almacenar datos de contexto (repositorios móviles, web y de datos del servidor),
- Compartir contexto para la investigación y para sí mismo (conocimiento compartido)

[7] www.awareframework.com/what-is-aware/

- Habilitar el uso del contexto aplicaciones (para los investigadores, desarrolladores de aplicaciones y usuarios).

Capturar de datos de contexto es un reto. De hecho, el contexto se produce en cualquier momento y en cualquier lugar, por todo y cualquier persona es extremadamente volátil y subjetivo. La siguiente Figura 14 - Contexto del framework AWARE describe cómo AWARE lo trata:

Figura 14 - Contexto del framework AWARE [8]

1. En el día a día, el teléfono utiliza sus sensores internos para recopilar datos. Se ha automatizado el proceso de recopilación de datos de la mayoría de los sensores disponibles en los teléfonos inteligentes Android.
2. La mayor parte de investigaciones previamente realizadas se ha limitado a un escaso número de usuarios y muy poco del trabajo utilizado es reusable: con frecuencia en los estudios de investigación se desarrolla software y herramientas de registro de datos de contexto desde cero, una y otra vez. Aware permite apoyar la reutilización de los resultados de investigación, datos y herramientas, haciendo posible la cooperación entre los científicos, independientemente de su disciplina. AWARE está diseñado como una arquitectura tipo plugin que permite que cualquiera pueda contribuir y ampliar las capacidades de AWARE.

   Los plugins desarrollados para AWARE tienen como objetivo principal recolectar y abstraer datos de sensores generados por contexto. Al reutilizar datos de contexto generados por plugins existentes, es posible desarrollar otros plugins para crear nuevos contextos de nivel más alto. En segundo lugar, también pueden presentar y

[8] http://www.awareframework.com/context/

explicar el contexto. Los plugins permiten que los usuarios se beneficien de las capacidades de contexto añadido y proporcionan una interfaz para la interacción con el usuario o para la presentación de información de contexto.

3. El contexto producido por AWARE es compartido a través de tres estrategias diferentes: broadcasts (emisores), observers (observadores) y providers (proveedores). Cada uno ofrece pros y contras, que se verán a continuación.

   Broadcasts

   Se utiliza principalmente para actualizar rápidamente otros sensores, plugins y aplicaciones de contexto del usuario. Es el camino más ligero para ser notificado de cambios en el contexto. Una aplicación no es necesario que contenga AWARE embebido, todo lo que se necesita, es utilizar los BroadcastReceivers de Android. Se debería utilizar, en situaciones donde una aplicación o servicio debe ser notificado sobre el contexto actual, independientemente de los datos capturados, generalmente estas notificaciones corresponden a cambios. Es posible capturar una o múltiples emisiones de Broadcasts simultáneamente.

   Para su uso, basta con considerar las siguientes líneas de código:

```
private static ContextReceiver contextReceiver = new ContextReceiver();
public static class ContextReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
           //Action for context here.
  }
}
```

   Es recomendable no realizar tareas muy complejas y de alto tiempo de procesamiento dentro del método onReceive(). Los broadcasts deben retornar en menos de 15 segundos (timeout), de lo contrario Android las interrumpirá, generando mensajes ANR (Android no responde).

   Adicionalmente es necesario registrar el BroadcastReceiver en la aplicación o plugin que lo use. Android permite dos métodos para hacerlo: en el Manifest (aplicaciones) o tiempo de ejecución (para los plugins). Para registrar el BroadcastReceiver en tiempo de ejecución, se utiliza el siguiente código en el método onCreate de la aplicación o plugin:

```
@override
protected void onCreate(...) {
  super.onCreate(...);
  IntentFilter contextFilter = new IntentFilter();
  //Check the sensor/plugin documentation for specific context broadcasts.
```

```
  contextFilter.addAction(ACTION_AWARE_CONTEXT_BROADCAST);
  registerReceiver(contextReceiver, contextFilter);
}
```

Providers

Se utiliza para almacenar datos de contexto de sensores sin procesar o de plugins. Se hace por medio de los ContentProviders de Android. Los datos se almacenan localmente en el dispositivo, y si es necesario, de forma remota en un servidor MySQL utilizando el dashboard provisto por AWARE.

Los Providers permiten el acceso a los datos principalmente de manera pasiva. En otras palabras, son los plugin o aplicaciones los responsables de solicitar los datos (es decir, peticiones pull), utilizando un cursor de Android de la siguiente forma:

```
//CONTENT_URI is the sensor's or plugin's table URI
Cursor sensorData = getContentResolver().query(CONTENT_URI,
tableColumns, whereCondition, whereArguments, orderBy);
```

Cada proveedor tiene URIs únicas de contenido; una por cada tabla de almacenamiento. Los proveedores son los archivos de base de datos SQLite, donde cada base de datos tiene sus propias tablas, columnas y esquema. Las URIS específica de cada sensor o plugin se deben mirar en la documentación correspondiente. Común a todos son tres columnas de tabla: _id, marca de tiempo y device_id, que mantienen el identificador de entrada de base de datos (asignado automáticamente a cada inserción), la instancia de la hora del evento y de AWARE ID de dispositivo del dispositivo móvil.

Observers

Se utiliza para realizar seguimiento de los cambios de datos del contexto y de los sensores en tiempo real. Están implementados como ContentObservers de Android. Los Observers proporcionan acceso activo por eventos de contexto (es decir, push). Los observadores son energéticamente eficientes y permiten el etiquetado contexto en tiempo real. Cuando se crea un observador, se debe implementar lo que pasa cuando existen nuevos datos disponibles. La siguiente línea de código nos permite ilustrar como se utiliza:

```
public ContextObserver(Handler handler) {
  super(handler);
```

```
  @Override
  public void onChange(boolean selfChange) {
            super.onChange(selfChange);
            //Update or query the data from the provider's URI.
  }
}
```

Cuando se define el observer, es necesario especificar a qué tabla se realizara el seguimiento, agregando lo siguiente en el método onCreate() del plugin o aplicación:

```
@override
protected void onCreate(...) {
  super.onCreate(...);
  ContextObserver contextObserver = new ContextObserver(new Handler());
  //CONTENT_URI is the sensor's or plugin's table URI you are interested in
  getContentResolver().registerContentObserver(CONTENT_URI, true,
  contextObserver);
}
```

4. Por ultimo es posible descargar directamente desde el cliente móvil de AWARE funcionalidades adicionales. Cualquier sensor adicional, aplicación o plug-in pueden ser compartidos públicamente. Si es necesario, el cliente AWARE notifica automáticamente al usuario de una actualización o le solicita instalar ningún plugin-dependencia de dependencias. El dashboard de AWARE permite implementar a gran escala, estudios distribuidos o estudios de carácter personal.

4.3 Machine learning: Weka en Android

En P. Liu, Y. Chen, W. Tang, Q. Yue3 (2012) presentan y ponen en práctica un framework llamado MobileWeka, basado en Java para extender herramienta de minería de datos Weka a la plataforma móvil. Proporciona una interfaz de usuario gráfica amigable. Además simplifica las funciones de minería de clasificación, clustering y de reglas asociado en plataformas Android.

Este framework propuesto, está diseñado para ejecutar tareas de minería de datos y machine learning en dispositivos móviles. Además habilita el uso de los algoritmos incluidos en Weka. MobileWeka apoya tareas de minería de datos estándar, por ejemplo, clustering, clasificación y reglas de asociación. El principal problema que resuelve el framework es abstraer el flujo de control básico de minería de datos y proporcionar a los usuarios una interfaz gráfica que suministra características de simplicidad y extensibilidad.

El framework está compuesto de cuatro capas según se puede ver en la Figura 15 - Arquitectura MobileWeka. La primera de ellas es el sistema operativo Android; el segundo es Android SDK Java que depende de la primera; la tercera es Weka que está a cargo de los algoritmos de minería de datos. La última es la capa de presentación, que proporciona una interfaz de usuario y la lógica de control amigable.

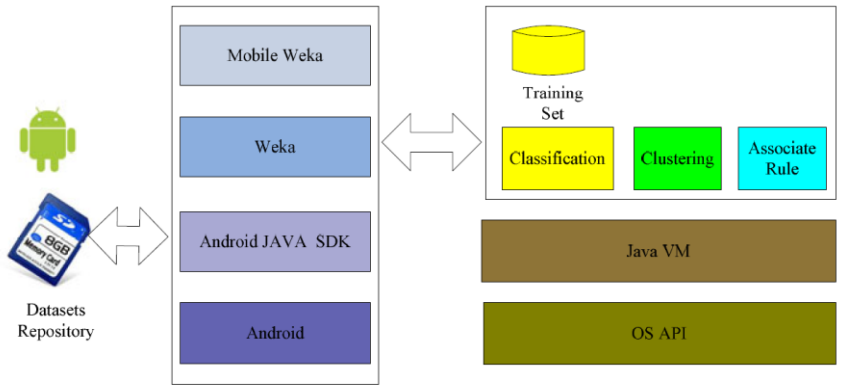


Figura 15 - Arquitectura MobileWeka

Cabe resaltar que la capa de presentación permite al usuario manipular los parámetros de configuración para el entrenamiento y pruebas de los diferentes modelos, ver Figura 16 - UI MobileWeka.

Figura 16 - UI MobileWeka

5. Desarrollo de la propuesta

Para la implementación de la propuesta se utiliza una metodología basada en el análisis, diseño y desarrollo de la solución. Iniciando con la identificación de requisitos, se especifican las funcionalidades, atributos de calidad y los actores del sistema. Luego se definen algunos casos de uso, un diagrama de dominio y se propone la arquitectura general del sistema. Adicionalmente se explica de en forma detallada, como se han implementado cada uno de los módulos y clases que componen la solución desarrollada, exponiendo los aspectos más relevantes y algunas consideraciones sobre la forma como se implementó. En este documento no se incluirá en su totalidad el código fuente de la solución, omitiendo comentarios y secciones relacionadas con el tratamiento de errores. Si es necesario ver el código fuente completo, habrá que remitirse a los archivos de implementación.

El código fuente de la propuesta está escrito en Java, específicamente para la plataforma Android. El entorno de desarrollo (IDE) utilizado es Eclipse para Java, integrado con el SDK de Android.

5.1 Requisitos del software

Los requisitos son descripciones de las necesidades funcionales y no funcionales que el software debe realizar para satisfacer los requisitos de usuario y del negocio. En este apartado se listan cada uno de los requisitos relacionados al proyecto, los cuales si están bien definidos, permiten encaminar el proyecto al éxito, de una manera eficiente y efectiva.

Requisitos funcionales

Estos requisitos son descripciones de los servicios que el sistema debe proporcionar, hacen referencia al conjunto de funcionalidades, comportamientos y características que el sistema debe proveer referente a la funcionalidad del sistema. Como se ha planteado en el objetivo de trabajo, el principal requisito de cara a alcanzar las expectativas del proyecto, es capturar

la información de usuario, para luego generar conocimiento. A continuación se definen los requisitos funcionales del proyecto:

- El sistema debe ser capaz de capturar eventos e información de contexto de usuario en una plataforma móvil
- El sistema debe tener la posibilidad de almacenar eventos capturados para su posterior análisis
- Generar conocimiento sobre el usuario a partir de la información de contexto capturada
- El sistema debe tener la posibilidad de aprender tiempo de ejecución, con los datos que el usuario le retroalimente.
- El sistema debía proporcionar una interfaz gráfica de usuario que de acceso a las funcionalidades del sistema.

Requisitos no funcionales

Un requisito no funcional son las restricciones o atributos de los servicios y funciones ofertadas por el sistema, las cuales especifican criterios para juzgar su operación en lugar de sus comportamientos específicos.

En este proyecto se han identificado algunos atributos de calidad y requisitos técnicos que definen las necesidades no funcionales del sistema así:

Atributos de calidad

- Facilidad de mantenimiento
- Escalabilidad
- Rendimiento
- Reusabilidad

Requisitos técnicos

- Desarrollar el servicio para usuarios del sistema operativo Android 2.3 o superior
- No es necesario disponer de conectividad a internet
- El código debe de estar lo más desacoplado posible unas clases de otras, y definir correctamente la funcionalidad de cada objeto para conseguir una alta cohesión.

5.2 Análisis y diseño de propuesta

Un caso de uso es una secuencia de interacciones que se desarrollarán entre un sistema y sus actores en respuesta a un evento que inicia un actor principal sobre el propio sistema. Los diagramas de casos de uso sirven para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas. En los siguientes apartados se presentarán los actores involucrados en los casos de uso, un diagrama de contexto y por último la especificación de los casos.

Actores

Los actores representan entidades externas que interactúan con el sistema, y pueden ser desde un usuario hasta un sistema externo. A continuación se identifican los actores y se da una breve descripción sobre su interacción:

- Usuario móvil: Representa a la persona que utiliza el dispositivo móvil constantemente y que interactúa con las diferentes aplicaciones instaladas
- Sistema operativo (SO) Android: El Sistema operativo proporciona acceso a datos de servicios, características de configuración, interacciones de usuario o preferencias de configuración del dispositivo móvil.
- AWARE Service: Este actor representa el framework AWARE, anteriormente descrito, el cual brinda información de contexto que es obtenida a través de los diferentes sensores y componentes de hardware con los que cuenta el dispositivo móvil.

Diagrama de dominio

Con el siguiente diagrama de dominio se logran identificar todos los temas relacionados con el problema, describiendo distintas entidades, atributos, papeles, relaciones y restricciones que rigen el dominio del problema.

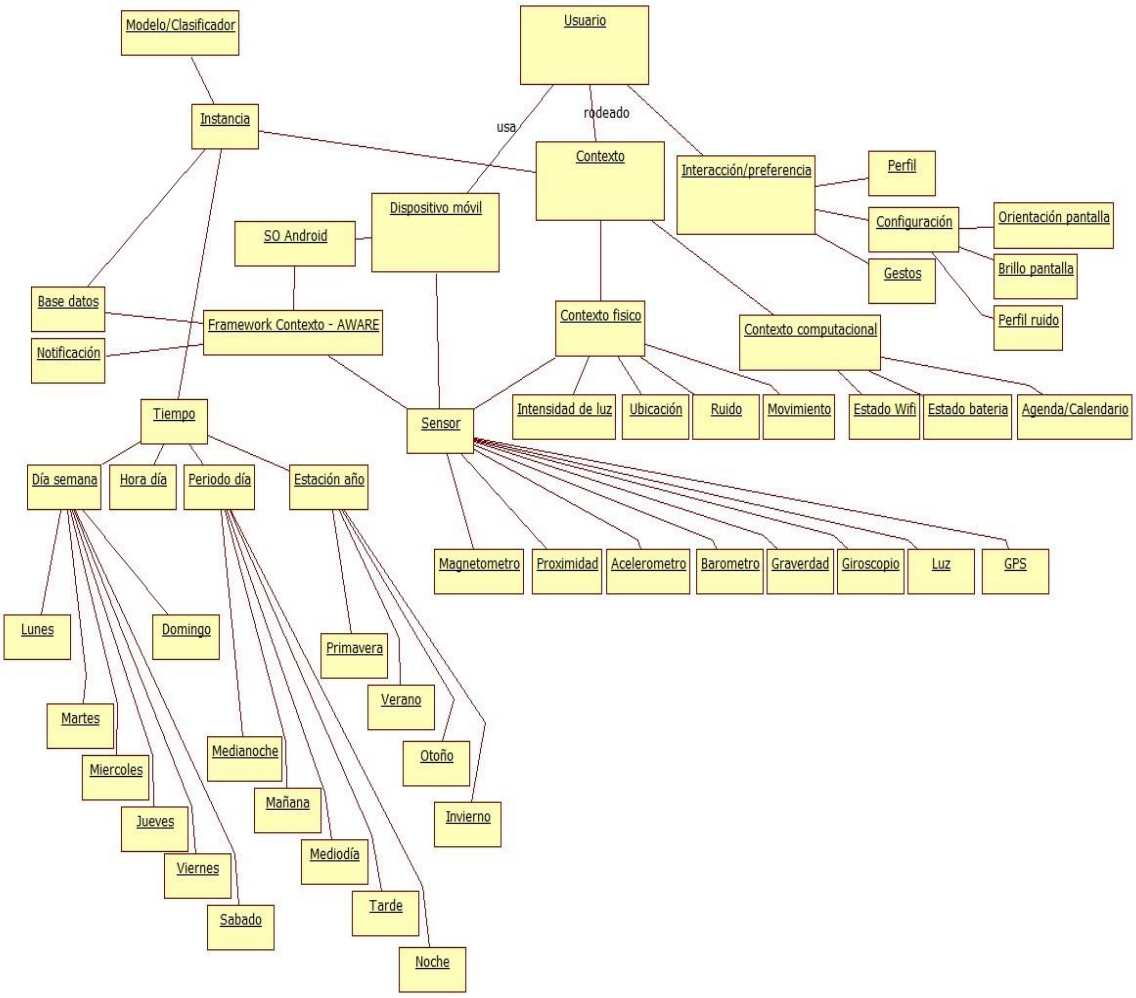


Figura 17 - Diagrama de Dominio

El modelado del contexto es una tarea compleja. Es necesario acotar el problema a escenarios y contextos específicos. En la Figura 17 - Diagrama de Dominio referente al diagrama de dominio del problema, se ha limitado el alcance y se han identificado las entidades, relaciones

y atributos concernientes con el contexto, las cuales son tenidos en cuenta en el desarrollo de la solución.

En términos generales este diagrama describe como el usuario interactúa con el dispositivo móvil, su entorno (contexto), los servicios del sistema operativo y a su vez el framework AWARE. Los dispositivos móviles tienen diferentes sensores que permiten capturar datos de contexto y generar personalizaciones. Adicional a esto, están los servicios prestados por el sistema operativo Android que habilitan la consulta del estado de componentes o funcionalidades. Es por esto que se ha clasificado el contexto en dos:

i) Contexto computacional: Datos que brinda el sistema operativo como el estado de wifi, estado de batería, información de calendario y eventos en agenda, etc.).
ii) Contexto físico: Datos que brindan los sensores del entorno físico que rodea al usuario tales como intensidad de luz, ubicación, velocidad, etc.

No se tuvo en cuenta “Contexto de usuario digital” el cual corresponde a la información referente a cuentas y estado de redes sociales, comunicación, etc.

Por último se puede observar como el contexto es capturado en instancias, que son fotos de contexto en instantes de tiempo dado. Estas instancias son almacenadas en una base de datos, donde luego son usadas para generar modelos o clasificadores, que posibilitan la inferencia de conocimiento acerca del usuario.

Casos de uso

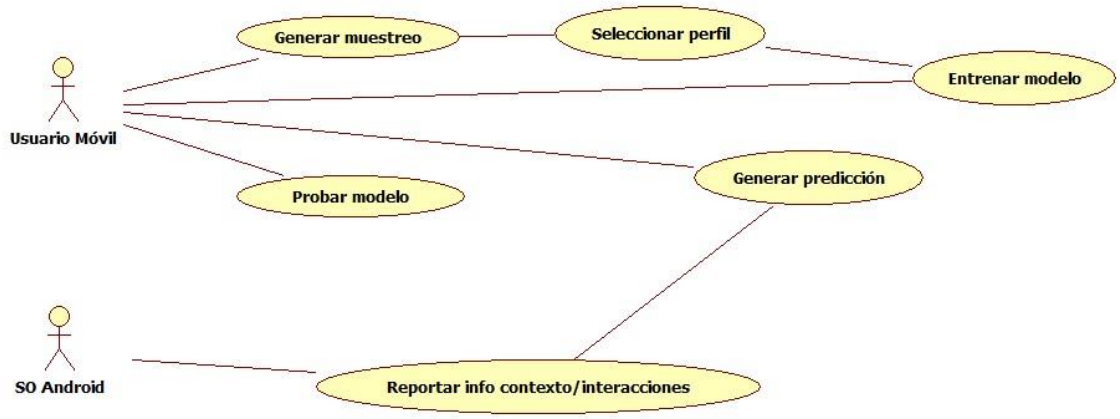


Figura 18 - Casos de uso

Especificación de los casos de uso

Generar muestreo

Descripción	El usuario genera información de muestreo para entrenar el modelo clasificador. Los datos de contexto y de interacción de usuario, así como el perfil seleccionado, son almacenados en BD.
Actor	Usuario móvil
Precondición	Seleccionar uno de los perfiles de usuario dispuestos en la app. La aplicación debe estar abierta para comenzar el muestreo.
Comentarios	Si el usuario no ha selecciona ninguna opción no se debe generar registros. El muestreo se realiza por medio de un timer que se ejecuta cada N segundos. Este valor es configurable por el usuario.

Seleccionar perfil

Descripción	Selección el perfil en el cual se encuentra el usuario. Estos perfiles están previamente definidos.
Actor	Usuario móvil

Precondición	La aplicación debe estar abierta en la pestaña de sampler.
Comentarios	NA

Generar predicción (clasificación)

Descripción	Para probar los modelos con los que cuenta el sistema, existe una opción que permite al usuario generar una predicción con los datos actuales de contexto e interacción.
Actor	Usuario móvil
Precondición	NA
Comentarios	NA

Reportar información de contexto/interacción

Descripción	El framework y el sistema operativo, reportan la información de contexto según la configuración de los mismos. Esta información activa la generación de predicciones o entrenamiento según el modelo.
Actor	SO Android / AWARE Service
Precondición	NA
Comentarios	NA

5.3 Arquitectura general y detalles de la implementación

Para desacoplar el desarrollo del servicio y así cumplir con uno de los requisitos no funcionales definidos previamente, se ha propuesto una arquitectura multicapa; la cual permite principalmente el desarrollo por niveles (en caso de cambios, sólo se afecta al nivel requerido, sin tener que revisar todo el código, o en caso de reutilización se toman las clases o librerías que se vayan a usar). Esta separación por capas es de tipo lógico. En la Figura 19 - Arquitectura del servicio se observa la arquitectura general en la que está centrado el proyecto.

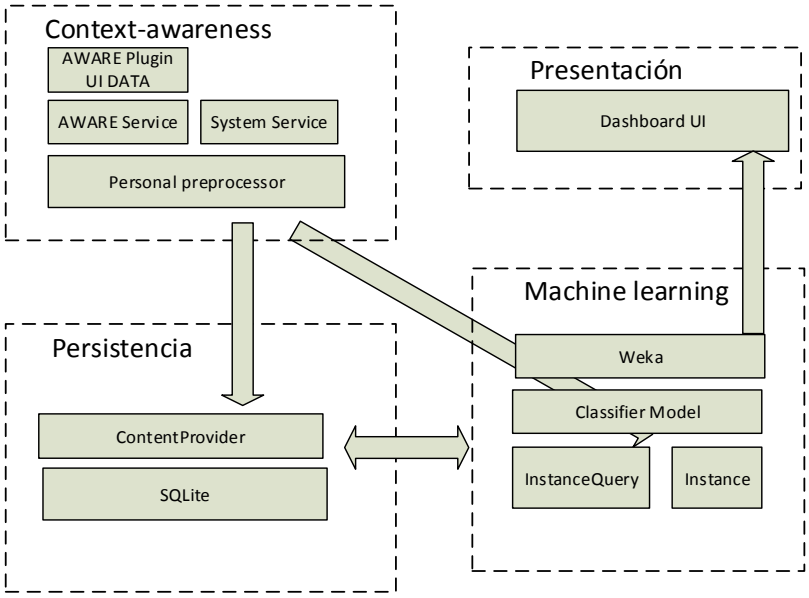


Figura 19 - Arquitectura del servicio

En esta arquitectura se aprecian las siguientes capas:

1) Context-awareness (sensibilidad al contexto): Es donde se implementa la funcionalidad del sistema en términos de sensibilidad al contexto. Incluye un consumidor de los sensores móviles e interacciones de usuario, un módulo de pre-procesado y tratamiento de datos.

2) Machine Learning: Es donde se procesa los datos de contexto. Incluye un módulo de muestreo, la generación y uso de modelos clasificadores para inferir datos del usuario, y por ultimo su evaluación.

3) Persistencia: Es donde se almacenan los datos y proporciona una serie de servicios que permite compartir los datos, facilitando las operaciones de inserción, consulta, borrado y modificación.

4) Presentación: Donde se proporcionará a los usuarios la posibilidad de interaccionar con el sistema a través de un interfaz de usuario. Esta da la posibilidad al usuario de visualizar un resumen de la información de las evaluaciones que los modelos clasificadores que se han generado.

Para lograr la implementación, en la Figura 20 - Distribución de paquetes se puede observar la distribución de paquetes dentro del desarrollo, conservando la separación de las capas

planteadas en la arquitectura. Se puede evidenciar los diferentes componentes del servicio que permiten dar solución al problema planteado:

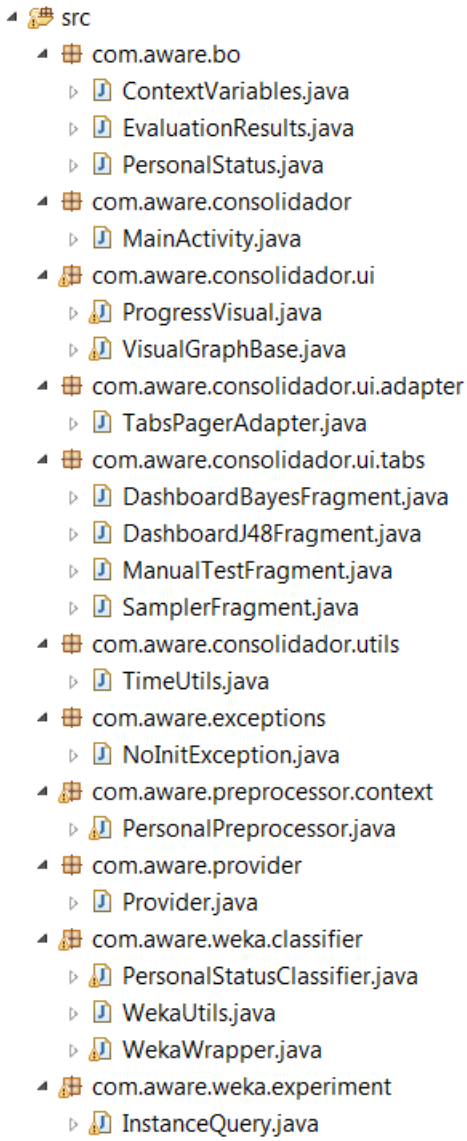


Figura 20 - Distribución de paquetes

A continuación entraremos en detalle de cada una se las capas y se expondrán las diferentes consideraciones que se tuvieron en el momento de hacer la implementación. En algunos de los apartados se hablará un poco sobre las implementaciones Ad-hoc que se desarrollaron como pruebas de laboratorio o simplemente para evidenciar el uso y aplicabilidad de las mismas.

5.3.1 Capa de persistencia

Persistencia se refiere a la propiedad de los datos para seguir existiendo en el tiempo. En nuestro caso, en este apartado es donde se almacenan los datos de contexto del usuario. Android provee múltiples formas de hacerlo dependiendo de las necesidades[9]. A continuación se lista algunas de estas y se detalla cual es utilizada dentro del proyecto:

- Shared Preferences: Almacenamiento privado de datos primitivos (Se llama tipo primitivo o tipo elemental a los tipos de datos originales de un lenguaje de programación), en pares de datos lave valor (key-value pairs).
- Almacenamiento local en ficheros: interno (privado) o externo (publico)
- Bases de datos SQLite: Almacenamiento estructurado en bases de datos.
  Este sistema de gestión de base de datos es usado en Android por su portabilidad, simplicidad y eficiencia en el manejo de recursos en dispositivos limitados.
- Content Provider o Proveedores de contenidos: mecanismo para exponer datos de una aplicación a otra (archivos normales, bases de datos SQLite o incluso archivos de red). Es un componente opcional que expone el acceso de lectura / escritura a los datos de aplicación, sin perjuicio de las restricciones que quiere imponer.

Figura 21 - Persistencia en Java

Al evaluar si los datos deben ser privados a la aplicación o accederse desde otras aplicaciones, la manera de consultarlos y la cantidad de espacio que los datos requieren, se puede concluir que el uso del Content Provider en combinación con una base de datos SQlite, es la mejor opción. Requerimos que estos datos puedan accederse desde diferentes aplicaciones para generar conocimiento a partir de ellos, además hay que considerar que los volúmenes de datos pueden ser grandes, ya que la información del contexto está cambiando constantemente.

[9] http://developer.android.com/intl/es/guide/topics/data/data-storage.html

Como ya hemos visto, un ContentProvider en Android es el encargado de gestionar un conjunto de datos de aplicación los cuales se han de compartir. Estos datos pueden ser almacenados en un sistema de archivos, una base de datos SQLite, en la Web o en cualquier otro lugar persistente a la que la aplicación pueda acceder. Es por esto que cuando pensamos usar un ContentProvider debemos tener las siguientes consideraciones:

- Qué tipo de archivos vamos a compartir con el resto de aplicaciones.
- Deberemos crear una clase que extienda de Content Provider con los métodos necesarios y esta será la encargada de compartir los datos con el resto de aplicaciones.
- Definir las URIs que sean necesarias, simplemente son enlaces simbólicos hacia los datos que queremos compartir, similar a los enlaces web. Para crearla se debe seguir la siguiente estructura:

  <prefijo>://<autoridad>/<datos>/<id>

Al implementar el `ContentProvider` en el proyecto, se define que el tipo de recurso a publicar será una base de datos SQLite, que estará almacenada en la carpeta definida en la variable `DATABASE_NAME` y cuenta con una URI referenciada en `CONTENT_URI`. Los nombres de las columnas que componen la tabla en donde se almacenan los registros, se definen en `PersonalContext_Data` que implementa la interfaz `android.provider.BaseColumns`. Para aumentar o disminuir el número de columnas es cuestión de modificar dicha clase y se puede hacer fácilmente:

```
public static final class PersonalContext_Data implements BaseColumns {
    private PersonalContext_Data() {};

    public static final Uri CONTENT_URI = Uri.parse("content://"+Provider.AUTHORITY+"/personalcontext");
    public static final String CONTENT_TYPE = "vnd.android.cursor.dir/vnd.aware.plugin.personalcontext";
    public static final String CONTENT_ITEM_TYPE = "vnd.android.cursor.item/vnd.aware.plugin.personalcontext";

    public static final String _ID = "_id";
    public static final String TIMESTAMP = "timestamp";
    public static final String DEVICE_ID = "device_id";
    public static final String DAY_OF_WEEK = "day_week";
    public static final String DAY_PERIOD = "day_period";
    public static final String HOUR_OF_DAY = "hour_day";
    public static final String SEASON = "season";
    public static final String EVENT_ON_CALENDAR = "event_on_calendar";
    public static final String LIGHT_LEVEL = "light_level";
    public static final String NOISE = "noise";
    public static final String DEVICE_PROFILE = "device_profile";//Sound
    public static final String WIFI_STATUS = "wifi_status";
    public static final String SCREEN_STATUS = "screen_status";
    public static final String ON_MOVEMENT = "on_movement";
    public static final String SCREEN_BRIGHNESS = "screen_brightness";
    public static final String BATTERY = "battery";
    public static final String SCREEN_ORIENTATION = "screen_orientation";
    public static final String PERSON_STATUS = "person_status";
}
```

Luego se crea la clase `Provider` que extiende de "`ContentProvider`", la cual será la encargada de gestionar el acceso, creación, eliminación y modificación de datos, por ende debemos sobrescribir los siguientes métodos:

- onCreate: se llama al iniciar el Content Provider. En este método básicamente se instancia el `DatabaseHelper` (es una clase provista por el framework AWARE que hereda de `SQLiteOpenHelper` y hace genérica el acceso a datos) y `SQLiteDatabase` que son usados para conectarnos a la base de datos SQLite.

```
@Override
public boolean onCreate() {
    if( databaseHelper == null ) databaseHelper = new DatabaseHelper(getContext(), DATABASE_NAME,
            null, DATABASE_VERSION, DATABASE_TABLES, TABLES_FIELDS);
    database = databaseHelper.getWritableDatabase();
    return (databaseHelper != null);
}
```

Para instancia la clase `DatabaseHelper,` es necesario pasar el nombre de la base de datos, los nombres de las tablas, los nombres y tipos de datos de los campos y el número de versión. Esta información se define al inicio de la clase `Provider`.

```
public static final int DATABASE_VERSION = 1;
public static String DATABASE_NAME = Environment.getExternalStorageDirectory()+"/AWARE/personalcontext.db";
public static final String[] DATABASE_TABLES = {
    "personalcontext"
};
public static final String[] TABLES_FIELDS = {
            PersonalContext_Data._ID + " INTEGER primary key autoincrement," +
            PersonalContext_Data.TIMESTAMP + " REAL default 0," +
            PersonalContext_Data.DEVICE_ID + " TEXT default ''," +
            PersonalContext_Data.DAY_OF_WEEK + " TEXT default ''," +
            PersonalContext_Data.DAY_PERIOD + " TEXT default ''," +
            PersonalContext_Data.HOUR_OF_DAY + " TEXT default ''," +
            PersonalContext_Data.SEASON + " TEXT default ''," +
            PersonalContext_Data.EVENT_ON_CALENDAR + " TEXT default ''," +
            PersonalContext_Data.LIGHT_LEVEL + " TEXT default ''," +
            PersonalContext_Data.NOISE + " TEXT default ''," +
            PersonalContext_Data.DEVICE_PROFILE + " TEXT default ''," +
            PersonalContext_Data.WIFI_STATUS + " TEXT default ''," +
            PersonalContext_Data.SCREEN_STATUS + " TEXT default ''," +
            PersonalContext_Data.ON_MOVEMENT + " TEXT default ''," +
            PersonalContext_Data.SCREEN_BRIGHNESS + " TEXT default ''," +
            PersonalContext_Data.SCREEN_ORIENTATION + " TEXT default ''," +
            PersonalContext_Data.BATTERY + " TEXT default ''," +
            PersonalContext_Data.PERSON_STATUS + " TEXT default ''," +
    "UNIQUE (" + PersonalContext_Data.TIMESTAMP+","+ PersonalContext_Data.DEVICE_ID+")"
};
```

- getType: devuelve el tipo MIME de los datos del Content Provider (MIME se podría definir como una serie de convenciones o especificaciones dirigidas al intercambio de todo tipo de archivos (texto, audio, vídeo, etc.) Internet de una forma transparente para el usuario).
- insert: inserta nuevos datos en el Content Provider

- update: Actualiza datos existentes del Content Provider
- delete: Elimina datos del Content Provider
- query: devuelve los datos de consulta generados sobre la clase.

También es necesario incluir la siguiente línea en el manifest.xml para activar el ContentProvider:

```
<provider
    android:name="com.aware.provider.Provider"
    android:exported="true"
    android:authorities="com.euge.provider.consolidador"
    android:writePermission="com.aware.WRITE_CONTEXT_DATA" android:readPermission="com.aware.READ_CONTEXT_DATA"
    >
</provider>
```

Para hacer uso de este proveedor de contenido es necesario utilizar la Uri dispuesta para ellos: PersonalContext_Data.CONTENT_URI.

Para concluir con este apartado, al instanciar esta clase se puede acceder desde el sistema de ficheros a la base de datos, según se definió:

day_week	day_period	hour_day	season	event_on_calendar	light_level	noise	device_profile	wifi_status	screen_status	on_movement	screen_brightness	screen_orientation	battery	person_status
7	Tarde	13	Summer	0	Claro	0	Silent mode	OFF	locked	?	?	reverse landscape	Cargando	Estudiando
7	Tarde	13	Summer	0	Incandecente	0	Silent mode	OFF	locked	?	?	reverse landscape	Cargando	Estudiando
7	Tarde	13	Summer	0	Incandecente	0	Silent mode	OFF	on	?	?	reverse landscape	Cargando	Estudiando

Cabe resaltar que este componente del desarrollo se presenta de una manera aislada por que el proveedor de contenido puede estar publicado, ya sea interno a la aplicación o puede ser expuesto en un servicio aparte, debido a las propiedades antes mencionadas.

5.3.2 Capa de context-awareness

La captura de datos de usuario referentes al contexto y las interacciones que tiene este con el móvil, son el eje principal de este apartado. Como ya se ha descrito previamente, al utilizar estos datos es posible caracterizar la situación del usuario, para posteriormente generar conocimiento o información más relevante del usuario que permita la auto-adaptación de un sistema o mejore la interacción con el mismo. Este proceso de captura puede generar demasiados volúmenes de datos los cuales deben ser almacenados (apartado de persistencia).

Como se ha visto en el apartado anterior referente a trabajos relacionados, las fuentes de estos datos son amplias. En esta implementación haremos uso de los datos que son capturados por medio del framework AWARE (Ferreira, D., 2013) y otros servicios que brinda el sistema operativo Android.

La manera como se llegó a la decisión de usar el framework AWARE, fue a partir de la consulta mejores prácticas y de realizar algunas implementaciones Ad-hoc para capturar la información de sensores e información contextual.

Dentro de estas implantaciones Ad-hoc, se creó un servicio (hereda de la clase android.app.Service) el cual se ejecuta en backgroud, he inicia el proceso de captura de información de contexto al instanciar la clase `android.hardware.SensorManager` por medio de un servicio del sistema (SensorManager) `context.getSystemService( Context.SENSOR_SERVICE)` y a su vez es necesario registrar cada uno de los escuchadores `(registerListener)` correspondientes a los sensores del dispositivo. Con esta implementación se logró capturar datos tales como:

Tabla 3 Sensores implementación Ad-hoc

Nombre	SensorManager Sensor
Aceleración	getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
Aceleración Lineal	getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION)
Gravedad	getDefaultSensor(Sensor.TYPE_GRAVITY)
Campo magnético	getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD)
Orientación	getDefaultSensor(Sensor.TYPE_ORIENTATION)
Temperatura	getDefaultSensor(Sensor.TYPE_TEMPERATURE)
Luz	getDefaultSensor(Sensor.TYPE_LIGHT)
Presión	getDefaultSensor(Sensor.TYPE_PRESSURE)
Datos de código de barras	getDefaultSensor(Sensor.TYPE_BARCODE_READER)
Rotación vectorial	getDefaultSensor(Sensor.TYPE_ROTATION_VECTOR) LocationManager

En esta implementación Ad-hoc se evidenció la complejidad que conlleva la captura de contexto usando las herramientas y servicios que brinda nativamente el sistema operativo, y se identificaron las siguientes problemáticas:

1. Es necesario realizar manualmente el control de los cambios de datos en cada uno de los sensores.
2. Para implementar muestreo por tiempos, es necesario hacerlo de manera individual por cada sensor.
3. La persistencia de datos y el mecanismo para compartirlos es necesario desarrollarlo desde cero

Con base en el punto anterior, y revisando los múltiples beneficios que ofrece AWARE, como su madurez, facilidad de implementación, portabilidad, soporte académico (Ferreira, D., 2013), licencia de código abierto y vigencia (actualmente está en constante evolución, última actualización: 6 de agosto, 2014[10]), se tomó la decisión de utilizar y aprovechar las ventajas que AWARE ofrece, para integrarlo dentro de la implementación de este proyecto.

Datos de contexto

Existen múltiples datos que pueden ser obtenidos de los dispositivos móviles relacionados con el contexto del usuario, en la siguiente tabla se relacionan cada uno de estos datos, los cuales fueron expresados en variables y su respectiva fuente de origen:

Tabla 4 Datos de contexto y sus fuentes

Nombre variable raw	Fuente
Nivel luz	Sensor de luz (AWARE)
Ruido de ambiente	Micrófono (AWARE)
Perfil de sonido de dispositivo	SO Android
Eventos de calendario	SO Android
Estado de wifi	SO Android
Estado de pantalla	Screen (AWARE)
Movimiento	Location GPS (AWARE)
Brillo de pantalla	SO Android
Orientación de pantalla	SO Android
Estado batería	SO Android
Día de la semana	Reloj
Periodo del día	Reloj
Hora del día	Reloj
Estación del año	Reloj

[10] http://www.awareframework.com/

Adicionalmente se pueden utilizar otros datos de sensores o interacciones de usuario. Estos datos, los cuales no fueron usados directamente en esta implementación, también están disponibles, para futuros trabajos. La Tabla 5 Fuentes de datos no usadas se encuentra un listado de datos con su respectiva fuente:

Tabla 5 Fuentes de datos no usadas

Nombre variable raw	Fuente
Cambios de orientación	Giroscopio (AWARE)
Aceleración lineal	Acelerómetro (AWARE)
Info aplicaciones (aplicaciones en backgroud, aplicaciones instaladas,	AWARE
Cambios de presión	Barómetro
Bluetooth	AWARE
Datos de comunicación (llamadas, mensajes)	AWARE
Gravedad	Sensor de gravedad (AWARE)
Campos magnéticos	Magnetómetros
Temperatura	Termómetro (AWARE)
Datos de operador móvil	Telefonía (AWARE)
Proximidad	Sensor de proximidad (AWARE)
Llamada actual	AWARE

Pre-procesado de información

Con el pre-procesado se busca aplicar una fase proveniente del datamining (minería de datos), llamada "Transformación del conjunto de datos de entrada", la cual tiene como objetivo, preparar los datos para aplicar la técnica de minería de datos que mejor se adapte a los datos y al problema. En este caso se hace una transformación de los datos a variables nominales.

Para el caso el pre-procesado de datos se realiza por medio de la clase "`PersonalPreprocessor`". Para implementarla se utilizó el patrón de diseño Singleton que permite restringir la creación de múltiples instancias de esta clase y proporciona un punto de acceso global a ella. Esta clase puede ser extendida al aumentar las fuentes de datos.

Contiene los siguientes métodos:

- getCalendarStatus

- getDarknessStatus
- getScreenStatus
- getDeviceProfileStatus
- getNoiseStatus
- getWifiStatus
- getBatteryStatus
- getBrightnessStatus
- getMovementStatus
- getScreenOrientation

En los métodos donde se envía como parámetro una variable tipo calendar, se busca condicionar la vigencia de los datos, debido que la capa de persistencia de los mismos, está separada para permitir mayor desacoplamiento de los otros componentes del sistema. Esto quiere decir que si un registro de datos de contexto proporcionados por AWARE, no está vigente, no se tendrán en cuenta para la descripción del contexto actual. Esta comparación se hace con la variable de timestamp que genera cada uno de los registros. Como ejemplo se puede observar la línea de código que nos permite realizar está comparación (se hace en todos los métodos que utilizan información proveniente de los ContentProviders que publican datos de contexto):

```
int diff=Math.abs(TimeUtils.minDifference(cal.getTime(),lightLevelDate));
if(diff>=0 && diff<MAX_MIN_DIFF){
```

Adicional a la recolección de datos dependiendo de su vigencia, se realizó el pre-procesado definiendo las siguientes variables nominales con sus respectivos valores para mejorar el proceso de clasificación:

Tabla 6 Variables de contexto

Fuente	Nombre variable raw	Nombre de variables nominales	Valores definidos
Sensor de luz (AWARE)	Nivel luz	Nivel luz	Incandescente, brillante, claro, semi oscuro, oscuro

Micrófono (AWARE)	Ruido de ambiente	Existe ruido	Si, No
SO Android	Perfil de sonido de dispositivo	Perfil dispositivo	Silent mode, Vibrate mode, Normal mode
SO Android	Eventos de calendario	Existe un evento agendado	Si, No
SO Android	Estado de wifi	Estado de wifi	On, Off
Screen (AWARE)	Estado de pantalla	Estado de pantalla	on, off, locked, unlocked
Location GPS (AWARE)	Movimiento	Hay movimiento	Si, No
SO Android	Brillo de pantalla	Brillo de pantalla	LOW, MEDIUM, HIGHT
SO Android	Orientación de pantalla	Orientación de pantalla	portrait, landscape, reverse portrait, reverse landscape
SO Android	Estado batería	Estado batería	Cargando, Full, Descargando, Desconocido
Reloj	Día de la semana	Día de la semana	Lunes, Martes, Miércoles, Jueves, Viernes, Sábado, Domingo
Reloj	Periodo del día	Periodo del día	Madrugada, Mañana, Mediodía, Tarde, Noche, Medianoche

Reloj	Hora del día	Hora del día	0,1,…,23
Reloj	Estación del año	Estación del año	Spring, Winter, Summer, Autumn

ContenProviders y SystemService

Los datos de contexto e interacciones listadas en el punto anterior, tienen como fuente el framework AWARE y el sistema operativo, que son expuestos como servicios del sistema, presentes en el contexto (`Context.getSystemService()`. Para el caso de AWARE, los datos son capturados y expuestos en cada uno de los ContentProviders que se disponen para ello. Esto facilita el acceso a los datos y permite su uso en cualquier momento, ya sea desde una aplicación, servicios locales, o de manera remota.

A continuación se listan los diferentes servicios, tanto del sistema como del framework, que fueron usados para la implementación:

- Nivel luz

```
Cursor sensorData = context.getContentResolver().query(Light_Data.CONTENT_URI,
   new String[] {  Light_Data.LIGHT_LUX, Light_Data.TIMESTAMP  },
   //KEY_ID + "=?",
   null,
   null,
   Light_Data.TIMESTAMP+" DESC");
```

- Existe ruido

```
Cursor sensorData = context.getContentResolver().query(AmbientNoise_Data.CONTENT_URI,
           new String[] {  AmbientNoise_Data.IS_SILENT, AmbientNoise_Data.TIMESTAMP  },
           //KEY_ID + "=?",
           null,
           null,
           AmbientNoise_Data.TIMESTAMP+" DESC");
```

- Perfil dispositivo

```
AudioManager am = (AudioManager)context.getSystemService(Context.AUDIO_SERVICE);
String mode="?";
switch (am.getRingerMode()) {
    case AudioManager.RINGER_MODE_SILENT:
        mode="Silent mode";
        Log.i(LOG_TAG,"Silent mode");
        break;
    case AudioManager.RINGER_MODE_VIBRATE:
        mode="Vibrate mode";
        Log.i(LOG_TAG,"Vibrate mode");
        break;
    case AudioManager.RINGER_MODE_NORMAL:
        mode="Normal mode";
        Log.i(LOG_TAG,"Normal mode");
        break;
}
```

- Existe un evento agendado

```
String[] EVENT_PROJECTION = new String[] {
    Calendars._ID,                           // 0
    Calendars.ACCOUNT_NAME,                  // 1
    Calendars.CALENDAR_DISPLAY_NAME,         // 2
    Calendars.OWNER_ACCOUNT                  // 3
};

// The indices for the projection array above.
int PROJECTION_ID_INDEX = 0;
int PROJECTION_ACCOUNT_NAME_INDEX = 1;
int PROJECTION_DISPLAY_NAME_INDEX = 2;
int PROJECTION_OWNER_ACCOUNT_INDEX = 3;

Time t = new Time();
t.setToNow();
long timeNow = t.toMillis(false);
//t.set(59, 59, 23, t.monthDay, t.month, t.year);
//long endOfToday = t.toMillis(false);
t.set(timeNow+3600000L);//suma una hora
long endOfToday = t.toMillis(false);

Uri.Builder eventsUriBuilder = CalendarContract.Instances.CONTENT_URI.buildUpon();
ContentUris.appendId(eventsUriBuilder, timeNow);
ContentUris.appendId(eventsUriBuilder, endOfToday);
Uri eventsUri = eventsUriBuilder.build();
Cursor cursor = null;
cursor =context.getContentResolver().query(eventsUri, EVENT_PROJECTION, null, null, CalendarContract.Instances.DTSTART + " ASC");
```

- Estado de wifi

```
WifiManager mainWifiObj = (WifiManager) context.getSystemService(Context.WIFI_SERVICE);
int state=mainWifiObj.getWifiState();
switch (state) {
case WifiManager.WIFI_STATE_DISABLED:
    stateString="OFF";
    break;
case WifiManager.WIFI_STATE_DISABLING:
    stateString="OFF";
    break;
case WifiManager.WIFI_STATE_ENABLED:
    stateString="ON";
    break;
case WifiManager.WIFI_STATE_ENABLING:
    stateString="ON";
    break;
case WifiManager.WIFI_STATE_UNKNOWN:
    stateString="";
    break;

}
```

- Estado de pantalla

```
Cursor sensorData = context.getContentResolver().query(Screen_Data.CONTENT_URI,
    new String[] {  Screen_Data.SCREEN_STATUS, Screen_Data.TIMESTAMP  },
    null,
    null,
    Screen_Data.TIMESTAMP+" DESC");
```

- Hay movimiento

```
Cursor sensorData = context.getContentResolver().query(Locations_Data.CONTENT_URI,
    new String[] {  Locations_Data.SPEED, Locations_Data.LATITUDE, Locations_Data.LONGITUDE, Locations_Data.TIMESTAMP  },
    //KEY_ID + "=?",
    null,
    null,
    Locations_Data.TIMESTAMP+" DESC");
```

- Brillo de pantalla

```
int screenBrightness = android.provider.Settings.System.getInt(context.getContentResolver(),
        Settings.System.SCREEN_BRIGHTNESS);
```

- Orientación de pantalla

```
Display mDisplay = ((WindowManager) context.getSystemService(Context.WINDOW_SERVICE)).getDefaultDisplay();
int rotation=mDisplay.getRotation();
switch (rotation) {
    case Surface.ROTATION_0:
        status= "portrait";
    case Surface.ROTATION_90:
        status= "landscape";
    case Surface.ROTATION_180:
        status= "reverse portrait";
    default:
        status= "reverse landscape";
}
```

- Estado batería

```
String stateString="?";
IntentFilter ifilter = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);
Intent batteryStatus = context.registerReceiver(null, ifilter);

// Are we charging / charged?
int status = batteryStatus.getIntExtra(BatteryManager.EXTRA_STATUS, -1);
switch (status) {
case BatteryManager.BATTERY_STATUS_CHARGING:
    stateString="Cargando";
    break;
case BatteryManager.BATTERY_STATUS_FULL:
    stateString="Full";
    break;
case BatteryManager.BATTERY_STATUS_DISCHARGING:
    stateString="Descargando";
    break;
default:
    stateString="Desconocido";
    break;
}
```

5.3.3 Capa de machine learning (aprendizaje automático).

En primer lugar, para aplicar el reto de machine learning, se debe entender la minería de datos o exploración de datos (KDD), el cual es un campo de las ciencias de la computación referido al proceso que intenta descubrir patrones en grandes volúmenes de conjuntos de datos. Utiliza los métodos de la inteligencia artificial, aprendizaje automático, estadística y sistemas de bases de datos. El objetivo general del proceso de minería de datos consiste en extraer información de un conjunto de datos y transformarla en una estructura comprensible para su uso posterior. Además de la etapa de análisis en bruto, que involucra aspectos de bases de datos y de gestión de datos, de procesamiento de datos, del modelo y de las consideraciones de inferencia, de métricas de intereses, de consideraciones de la Teoría de la complejidad computacional, de post-procesamiento de las estructuras descubiertas, de la visualización y de la actualización en línea. En la Figura 22 - Ciclo data mining y/o machine learning se puede observar ciclo de trabajo compuesto por las diferentes etapas que lo componen:

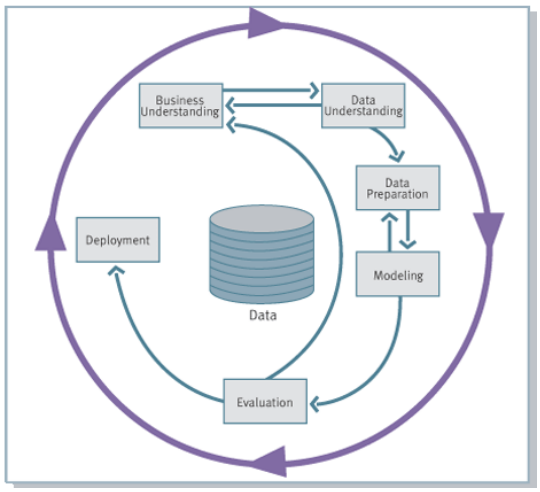


Figura 22 - Ciclo data mining y/o machine learning

Para implementar el aprendizaje automático que se hace a partir de la experiencia, se utiliza el aprendizaje supervisado, el cual depende de ejemplos. Estos ejemplos se deben generar a partir de preguntas que se hacen al usuario para retroalimentar el sistema y así aprender de la experiencia.

En el desarrollo de este apartado, además del pre-procesado, el cual se describe en el punto anterior, se utiliza como técnica de modelado la Clasificación para predecir o inferir el estado/perfil del usuario. Estos perfiles describen una situación o estado general del usuario

y según trabajos previos, pueden estar asociados a perfiles de configuración de la aplicación, modos de trabajo o alguna clasificación personalizada. Los perfiles son los siguientes (y pueden ser extendidos si es necesario):

- Ocio: usuario realizando actividades de ocio como ver televisión, estar en el ordenar, escuchar música, etc.
- Comiendo: usuario se encuentra comiendo
- Presentación: reunión donde se usa un video beam realizar alguna presentación.
- Transporte: usuario se encuentra en algún medio de transporte como un vehículo, bus, metro, etc.
- Caminando: usuario se encuentra andando en algún lugar abierto.
- Durmiendo: usuario se encuentra durmiendo
- Estudiando: usuario se encuentra estudiando, en algún salón de clases o lugar relacionado.
- Trabajando: usuario se encuentra en su despacho realizando labores de oficina.

Muestreo

El aprendizaje que se realiza partir de los datos que rodea al usuario, tiene lugar en el módulo para hacer el muestreo. Este muestreo se ejecuta como una tarea en background que está programada para hacer repeticiones de muestreo cada t tiempo. Este tiempo se define en la variable TIME_TO_SAMPLE donde el valor por defecto son 30 segundos. La clase involucrada es `SamplerFragment,` usando un `TimerTask` extendido en la clase `ContextSampler` para optimizar su ejecución. Los perfiles de selección están enmarcados en un `Spinner` llamado `spPersonalStatus` el cual es global a la clase `SamplerFragment.` Para detallar esto se puede observar las siguientes líneas de código:

```
private class ContextSampler extends TimerTask {
    @Override
    public void run() {
        if (spPersonalStatus != null) {
            //ignorar opcion de espacio en blanco
            //No se inserta si no hay datos del profile de la persona
            if(spPersonalStatus.getSelectedItemPosition()>0){
                String personStatus=PersonalStatus.VARS[spPersonalStatus.getSelectedItemPosition()-1];
                if(personStatus!=null && !personStatus.equals(VACIO)){

                    //Timestamp del sample que se captura
                    Calendar cal=Calendar.getInstance();
                    cal.set(Calendar.SECOND, 0);
                    cal.set(Calendar.MILLISECOND, 0);

                    ContentValues rowData = new ContentValues();
                    rowData.put(PersonalContext_Data.DEVICE_ID, Aware.getSetting(context.getContentResolver(), Aware_Preferences.DEVICE_ID));
                    rowData.put(PersonalContext_Data.TIMESTAMP, System.currentTimeMillis());
                    rowData.put(PersonalContext_Data.LIGHT_LEVEL, PersonalPreprocessor.getInstance().getDarknessStatus(cal));
                    rowData.put(PersonalContext_Data.NOISE, PersonalPreprocessor.getInstance().getNoiseStatus(cal));
                    rowData.put(PersonalContext_Data.DEVICE_PROFILE, PersonalPreprocessor.getInstance().getDeviceProfileStatus());
                    rowData.put(PersonalContext_Data.EVENT_ON_CALENDAR, PersonalPreprocessor.getInstance().getCalendarStatus());
                    rowData.put(PersonalContext_Data.WIFI_STATUS, PersonalPreprocessor.getInstance().getWifiStatus());
                    rowData.put(PersonalContext_Data.DAY_OF_WEEK, cal.get(Calendar.DAY_OF_WEEK));
                    rowData.put(PersonalContext_Data.DAY_PERIOD, TimeUtils.getDayPeriod(cal));
                    rowData.put(PersonalContext_Data.HOUR_OF_DAY, cal.get(Calendar.HOUR_OF_DAY));
                    rowData.put(PersonalContext_Data.SEASON, TimeUtils.getSeasonName(cal));
                    rowData.put(PersonalContext_Data.SCREEN_STATUS, PersonalPreprocessor.getInstance().getScreenStatus(cal));
                    rowData.put(PersonalContext_Data.ON_MOVEMENT, PersonalPreprocessor.getInstance().getMovementStatus(cal));
                    rowData.put(PersonalContext_Data.SCREEN_BRIGHNESS, PersonalPreprocessor.getInstance().getBrightnessStatus());
                    rowData.put(PersonalContext_Data.SCREEN_ORIENTATION, PersonalPreprocessor.getInstance().getScreenOrientation());
                    rowData.put(PersonalContext_Data.BATTERY, PersonalPreprocessor.getInstance().getBatteryStatus());
                    rowData.put(PersonalContext_Data.PERSON_STATUS, personStatus );
                    try{
                        context.getContentResolver().insert(PersonalContext_Data.CONTENT_URI, rowData);
                        PersonalStatusClassifier.getInstance().trainNaiveBayesClassifier(
                            (String)rowData.get(PersonalContext_Data.DAY_OF_WEEK),
                            (String)rowData.get(PersonalContext_Data.DAY_PERIOD),
                            (String)rowData.get(PersonalContext_Data.SEASON),
                            (String)rowData.get(PersonalContext_Data.EVENT_ON_CALENDAR),
                            (String)rowData.get(PersonalContext_Data.DEVICE_PROFILE),
                            (String)rowData.get(PersonalContext_Data.WIFI_STATUS),
                            (String)rowData.get(PersonalContext_Data.PERSON_STATUS)
                                );
```

Cabe resaltar que después de capturar los datos, se hace uso del ContentProvider creado para persistir los datos consolidados y pre-procesados relacionados con el usuario.

Aprendizaje automático: contruccion, entrenamiento y pruebas de los modelos

Para realizar este fase del aprendizaje automático, se utiliza la librería provista por el proyecto Weka. La librería que proporciona Weka (weka.jar) está inicialmente diseñada para correr en entornos de escritorio o servidores, pero debido a la portabilidad que brinda Java y la alta cohesión de la librería, fue posible con algunos cambios, ejecutar y hacer uso de ella en el proyecto. Esto quiere decir que es posible ejecutar la mayoría de funcionalidades que tiene expuesta weka sobre el sistema operativo Android. A continuación se describen los ajustes que se hicieron a la librería para mejorar el desempeño y ejecución de la misma en el ambiente de desarrollo:

1. Eliminar las funcionalidades experiment, provistas por weka para facilitar experimentos con datos. Esto se realizó con el fin de reducir el tamaño del jar y permitir mejorar el uso de memoria en tiempo de compilación.

2. Implementación de la clase IntanceQuery la cual proporciona un mecanismo para generar instancias a partir de una Base de datos. La clase actual permite a través de jdbc conectarse a diferentes bases de datos como MySql, Sql server, Oracle, sqlite, etc. En nuestro caso para Android, el uso de jdbc para las conexiones a base de datos no es la forma natural de hacerlo, estas conexiones se pueden hacer a través de DatabaseHelpers o ContentProvider que simplifican su conectividad. Es por ello que se modificó esta clase para que haga uso de un cursor expuesto por el ContentProvider y así recorrer el data set para generar las instancias.

Adicional a estos ajustes, fue necesario configurar el entorno de desarrollo Eclipse con más recursos del sistema. Esto quiere decir que se aumentó la memoria heap en el archivo de configuración eclipse.ini, donde se modificaron los parámetros –Xms y –Xmx, con valores mínimos 512m y 1024m respectivamente. Esta configuración fue necesaria debido que en el momento de ejecución del proyecto, el entorno de desarrollo no respondía y mostraba un error "java.lang.OutOfMemoryError: Java heap space". Este problema se presenta debido a pre-validación que hace el entorno sobre la librería de Weka en el momento de ejecución.

Retomando los cambios realizados en IntanceQuery, se puede mencionar que su método más relevante es **retriveInstances**, el cual ajusta los datos al formato de instancias y basa en Weka toda su operación. En las siguientes lineas de codigo podemos observar como está desarrollado este metodo. Es importante mencionar que este metodo clasifica los datos en 2 tipos de variables: númericas o nominales. En nuestro caso para entrenar modelos de clasificación se hace uso de variables nominales.

```
public Instances retrieveInstances( Uri uri_q, String[] columns_q,  String[] args_q, String tablename_q, String orderby_q) throws Exception {
  Cursor instancesCursor = context.getContentResolver().query(uri_q,
              columns_q ,
              null,
              //KEY_ID + "=?",
              args_q,
              orderby_q);
  if(instancesCursor!=null){
      //md.moveToFirst();
      if(!instancesCursor.moveToFirst())
          return null;

      // Determine structure of the instances
      //int numAttributes = md.getColumnCount();
      int numAttributes =instancesCursor.getColumnCount();
      int [] attributeTypes = new int [numAttributes];
      Hashtable [] nominalIndexes = new Hashtable [numAttributes];
      FastVector [] nominalStrings = new FastVector [numAttributes];
      for (int j = 0; j < numAttributes; j++) {
        switch (instancesCursor.getType(j)) {
          case Cursor.FIELD_TYPE_STRING  :
              attributeTypes[j] = Attribute.NOMINAL;
              nominalIndexes[j] = new Hashtable();
              nominalStrings[j] = new FastVector();
              break;
          case Cursor.FIELD_TYPE_INTEGER:
              attributeTypes[j] = Attribute.NUMERIC;
              break;
          case Cursor.FIELD_TYPE_FLOAT:
              attributeTypes[j] = Attribute.NUMERIC;
          break;
          default:
              attributeTypes[j] = Attribute.STRING;
              break;
        }
      }

      // For sqlite
      // cache column names because the last while(rs.next()) { iteration for
      // the tuples below will close the md object:
      Vector<String> columnNames = new Vector<String>();
      for (int i = 0; i < numAttributes; i++) {
        columnNames.add(instancesCursor.getColumnName(i));
      }
```

Para realizar todo el proceso de machine learning se ha implementado la clase `PersonalStatusClassifier`. En ella se utiliza el patron Sigleton para restringir la creación de múltiples instancias y proporcionar un punto de acceso global a ella, generando una carga de los modelos al inicializar la clase y evitar estár leyendolos desde los ficheros. Esta clase es la encargada de realizar las diferentes operaciones con los modelos de clasificación, que van desde la carga, creación, entrenamiento, guardado hasta las pruebas y evaluación. Contiene los siguientes métodos:

- Init: Inicializa los modelos clasificadores cargando los modelos desde archivo. Este metodo es necesario llamarlo una unica vez, para comenzar a trabajar con la clase, sino lanza un NoInitException. Por ultimo, es necesario pasar a este metodo el contexto de la aplicación o Activity.

```
public void init(Context c) throws Exception{
    context=c;
        Resources resources = context.getResources();
        assetManager = resources.getAssets();
        InputStream inputStream = assetManager.open("ModelV1.model");
        j48M=(J48) (new ObjectInputStream(inputStream)).readObject();

        try{
            File file = new File(URI.create(Environment.getExternalStorageDirectory()+"/AWARE/bayesPerson.model"));
            inputStream = new FileInputStream(file);
        }catch(Exception e){
            inputStream = assetManager.open("bayesPerson.model");
        }
        nbuM=(NaiveBayesUpdateable) (new ObjectInputStream(inputStream)).readObject();
```

- trainJ48Classifier: Entrenar el modelo J48
- trainNaiveBayesClassifier: Entrenar el modelo NaiveBayesUpdatable

```
public void trainNaiveBayesClassifier(String ...par) throws Exception{

    Instances data = new Instances("TestInstances",attributeList,1);
    data.setClassIndex(attributeList.size()-1);

    Instance inst =initInstance(data,true,par);
    nbuM.updateClassifier(inst);

}
```

- testJ48Classifier: Probar el modelo J48
- testNaiveBayesClassifier: Probar el modelo NaiveBayesUpdatable
- saveBayesClassifier: Salvar el modelo NaiveBayesUpdatable entrenado a partir de datos frescos.

```
public void saveBayesClassifier() throws Exception{

    weka.core.SerializationHelper.write(Environment.getExternalStorageDirectory()+"/AWARE/bayesPerson.model", nbuM);
}
```

- saveClassifier: Salvar un fichero .model clasificador de forma genérica para un uso posterior del modelo.
- loadClassifier: Cargar un fichero .model correspondiente a un clasificador previamente guardado (en weka ui o a través de la aplicación móvil) de forma genérica.
- initInstance: inicializar una instancia con datos de prueba o entrenamiento, especificos para esta implementación.
- getPrediction: Se obtiene una predicción de clasificación del perfil de usuario dependiendo de las condiciones de contexto.

Clasificación con J48

En este punto se utilizó el algoritmo J48 de WEKA es una implementación del algoritmo C4.5, uno de los algoritmos de minería de datos más utilizado. Se trata de un refinamiento del modelo generado con OneR (una regla) el cual es un clasificador más sencillo.

Para el uso y entrenamiento de este tipo de modelos tenemos 2 opciones:

- Creación a apartir de datos existentes.

```
Classifier cModel = (Classifier)new J48();
cModel.buildClassifier(isTrainingSet);
```

  Donde is TrainingSet es un conjunto de instancias para entrenar el modelo.
- Reutilización de modelo previamente generado (carga desde fichero)

```
Resources resources = context.getResources();
assetManager = resources.getAssets();
InputStream inputStream = assetManager.open("ModelV1.model");
j48M=(J48) (new ObjectInputStream(inputStream)).readObject();
```

Clasificación con Naive bayes actualizable

En teoría de la probabilidad y minería de datos, un clasificador Naive bayes (Bayesiano ingenuo) es un clasificador probabilístico fundamentado en el teorema de Bayes y algunas hipótesis simplificadoras adicionales. Es a causa de estas simplificaciones, que se suelen resumir en la hipótesis de independencia entre las variables predictoras, que recibe el apelativo de ingenuo. Por su origen probabilistico se han desarrollado algunas variantes que permiten actualizar el modelo clasificador a partir de datos fresco. En esta implementación se utilizó esta variante para mostrar la posibilidad de tener modelos actualizables.

Para utilizar este clasificador, al igual que el anterior se tienen las primeras opciones, la ultima solo es caracteristica de este tipo de modelos:

- Creación a apartir de datos existentes.

```
Classifier cModel = (Classifier)new NaiveBayesUpdateable();
cModel.buildClassifier(isTrainingSet);
```

- Reutilización de modelo previamente generado (carga desde fichero)

```
Resources resources = context.getResources();
assetManager = resources.getAssets();
InputStream inputStream = assetManager.open("ModelV1.model");
nbuM=(NaiveBayesUpdateable) (new ObjectInputStream(inputStream)).readObject();
```

- Aprendizaje con datos frescos

```
Instance inst =initInstance(data,true,par);
nbuM.updateClassifier(inst);
```

Según los resultados evaluados, este clasificador no arroja resultados comparablemente buenos como los generados por el clasificado obtenido de J48. Pero como previamente se mensiona, este tipo de modelos tiene la gran ventaja aprender con datos fresco, sin necesidad de volver a entrenar con datos historicos, a diferencia del J48 que si lo requiere. Esto permite un aprendisaje en tiempo real ya que el costo computacional de entrenamiento no es tan alto como lo fuera entrenando el modelo con datos historicos. Otros de los modelos que existen en Weka y tienen este mismo comportamiento son:

- HoeffdingTree,
- IBk,
- KStar,
- LWL,
- MultiClassClassifierUpdateable,
- NaiveBayesMultinomialText,
- NaiveBayesMultinomialUpdateable,
- SGD,
- SGDText

5.3.4 Capa de presentación

Como anteriormente se ha mencionado, la presentación es una capa donde se proporciona a los usuarios la posibilidad de visualizar datos, resultados o eventos del servicio, que son el resultado de evaluaciones, pruebas o predicciones que los modelos clasificadores han realizado, o bien para su entrenamiento. En el servicio, estas formas de visualización e interacción, se presentan de la siguiente forma:

- Toast
- Pantalla de muestreo
- Dashboard UI

Toast

Son notificaciones o mensajes que aparecen en pantalla y que sirven para informar al usuario sobre algún evento ocurrido en el sistema. Éste ocupa un espacio pequeño en la pantalla, y se realiza de forma no intrusiva. Las notificaciones aparecen y desaparecen automáticamente, y no aceptan eventos de interacción. Básicamente se utiliza para indicar al usuario el tipo de perfil/estado en el que se encuentra actualmente, según la información de contexto que ha capturado. Ver Figura 23 - Toast UI.

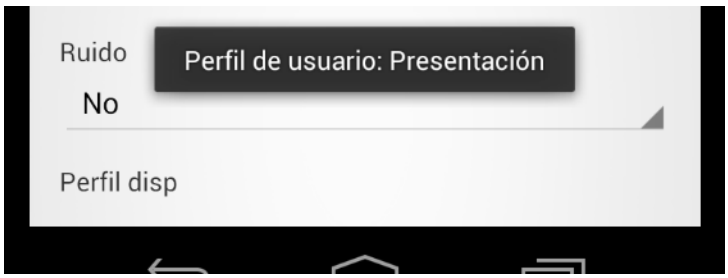


Figura 23 - Toast UI

Pantalla de muestreo (Sampler)

Esta interfaz de usuario que permite iniciar el muestreo de datos. Con esta captura de datos de contexto y respectiva clasificación se generan nuevos datos, para así entrenar los modelos clasificadores. Esta opción se encuentra en el tab “sampler” como se observa en la Figura 24 – Muestreo. Allí se selecciona el perfil/estado en el cual se encuentra el usuario. Cabe resaltar también que si se reutiliza este servicio dentro de otra aplicación es necesario desarrollar una nueva interfaz que permita reatrolimentar los modelos, pero tendría los mismos principios.

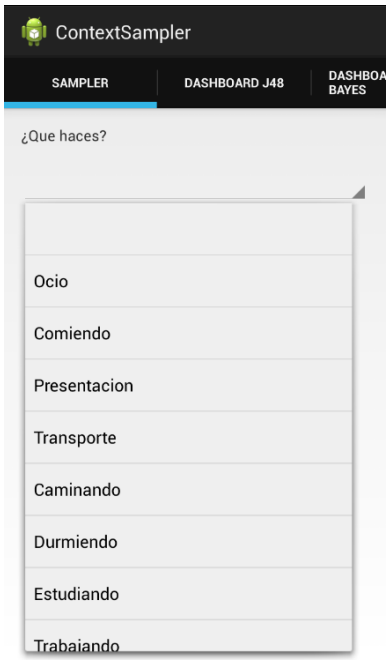


Figura 24 – Muestreo UI

Posterior a la selección del perfil se inicia el proceso de muestreo que anteriormente se ha explicado en el apartado anterior, y que básicamente se encarga de capturar la información de contexto, para luego persistirla en base de datos.

Dashboard UI

Adicionalmente a las funcionalidades anteriormente mesionadas, la aplicación cuenta con una interfaz gráfica para monitorear la evolución de los modelos. En la implementación solo se desarrolló para los dashboards relacionados con los algoritmos J48 (arboles de decisión) y Naive Bayes actualizable, como se puede ver en la Figura 25 - Dashboard UI.

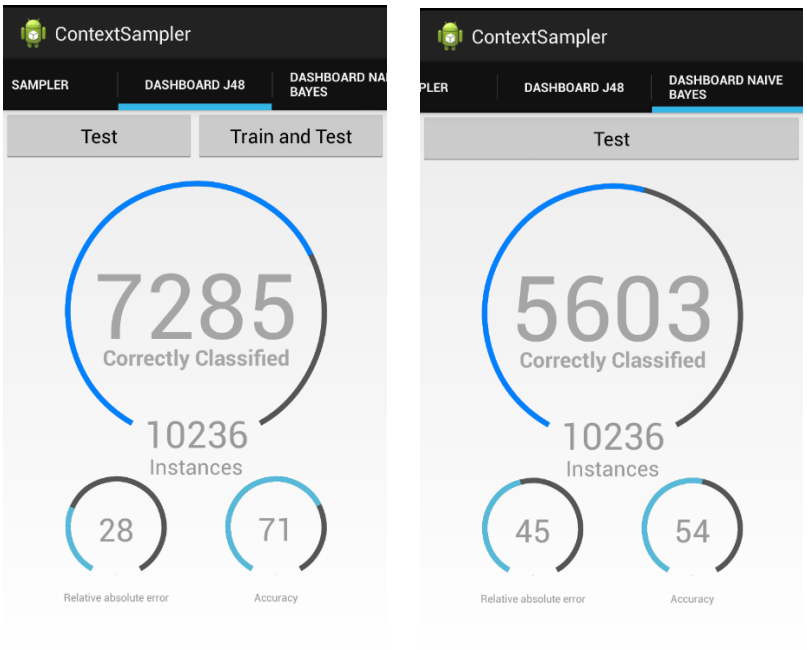


Figura 25 - Dashboard UI

Básicamente se puede hacer seguimiento de 3 variables que se usan en la minería de datos para el proceso de clasificación. Estas son:

- %Acierto
- %Error
- Precisión

5.4 Uso del servicio

Lo primero que debemos hacer es instalar el servicio en el dispositivo, por medio del archivo Consolidador.apk, el cual instancia la base de datos, los modelos y todos aquellos componentes que hacen parte del servicio. Este paquete de instalación tiene precargados unos modelos generados previamente con weka para escritorio, propios de unas pruebas realizadas. Lo cual permite generar clasificaciones desde un principio. Por otro lado si se requiere comenzar a entrenar el modelo, solo es necesario seleccionar la opción de sampler que previamente fue descrita.

Como ya hemos visto en apartados anteriores el servicio cuenta con componentes para el muestreo, entrenamiento y evaluación de los modelos, y básicamente lo que podemos obtener de este, es una clasificación del contexto actual. Estos modelos clasificadores tienen múltiples usos, entre ellos está la adaptación de interfaces de usuario que esta tesina no se implementó, pero que en combinación con trabajos previamente realizados se puede extender su uso.

Para hacer uso de los clasificadores, se debe invocar el método `getPrediction`, presente en la clase `PersonalStatusClassifier`. Este método, con una captura previa de datos de contexto, realiza una clasificación del perfil/estado del usuario, obteniendo inferencias del contexto que rodea al usuario. Este método recibe los siguientes parámetros:

1. Modelo a utilizar (int modelOp): opción del tipo de modelo a usar para hacer la clasificación de los datos. Para ello existen las opciones CLASSIF_J48 o CLASSIF_NAIVE_BAYES_UPDATE. Como se ha mencionado previamente se puede hacer uso de muchos otros clasificadores que no se implementaron en este trabajo, pero que es posible extender de una forma fácil.
2. Variables de contexto (String ...par): Todos los datos de contexto capturados del usuario. Es importante resaltar que el orden en que se pasan las variables de contexto debe corresponder al mismo con el que el método `initInstance` procesa los datos para la creación de las instancias. El orden es el siguiente:
    a. DAY_OF_WEEK
    b. DAY_PERIOD
    c. HOUR_OF_DAY
    d. SEASON
    e. EVENT_ON_CALENDAR
    f. LIGHT_LEVEL
    g. NOISE
    h. DEVICE_PROFILE
    i. WIFI_STATUS
    j. SCREEN_STATUS
    k. ON_MOVEMENT
    l. SCREEN_BRIGHNESS
    m. SCREEN_ORIENTATION
    n. BATTERY

En las siguientes líneas de código, se logra observar como al método se le envían los parámetros; este los transforma en una instancia, y luego dicha instancia es clasificada con alguno de los modelos de clasificación según el usuario del método lo haya solicitado.

```
public String getPrediction(int modelOp, String ...par) throws Exception{
    String info=null;
    if(context==null){
        throw new NoInitException();
    }else{

        try {

            Instances data = new Instances("TestInstances",attributeList,1);
            data.setClassIndex(attributeList.size()-1);

            // http://stackoverflow.com/questions/13029118/classifying-single-instance-in-weka
            //Classifying Single Instance in Weka
            Instance iExample =initInstance(data,false,par);

            Instances dataUnlabeled = new Instances("TestInstances", attributeList, 0);
            dataUnlabeled.add(iExample);
            dataUnlabeled.setClassIndex(attributeList.size()-1);

             // Test the model
            double result = -1;
            switch (modelOp) {
            case CLASSIF_J48:
                result = j48M.classifyInstance(dataUnlabeled.firstInstance());
                break;
            case CLASSIF_NAIVE_BAYES_UPDATE:
                result = nbuM.classifyInstance(dataUnlabeled.firstInstance());
                break;

            default:
                break;
            }

            Log.e(TAG, "Result: "+result);
            String label = dataUnlabeled.classAttribute().value((int) result);
            Toast.makeText(context, "Model "+modelOp+": "+label+" "+result+" - ", Toast.LENGTH_LONG).show();
```

También es importante adicionar que con el uso de los clasificadores, es posible implementar un mecanismo de retroalimentación. Esto quiere decir que si a un usuario de una aplicación que utiliza este mecanismo de clasificación, se le recomienda algún perfil/estado, el sistema sea capaz de evaluar la respuesta, preguntando al usuario si es correcta y generando nuevos datos de entrenamiento que posteriormente son persistidos en base de datos. Inclusive si el modelo es un Clasificador Actualizable se podría entrenar inmediatamente. Un ejemplo de ello es el método `trainNaiveBayesClassifier` que ilustra la manera de entrenar el modelo inmediatamente se reciben los datos.

```
public void trainNaiveBayesClassifier(String ...par) throws Exception{

    Instances data = new Instances("TestInstances",attributeList,1);
    data.setClassIndex(attributeList.size()-1);

    Instance inst =initInstance(data,true,par);
    nbuM.updateClassifier(inst);

}
```

Por último, se debe tener presente el uso del método `trainJ48Classifier()` el cual a partir de los datos persistidos en base de datos, entrena el modelo generando. Logrando el aprendizaje automático de los datos de contexto del usuario.

6. Caso de estudio

Después de implementar el servicio, utilizar las tecnologías relacionadas y aplicar algunos de los conceptos generales necesarios para el desarrollo del trabajo realizado, se procede a presentar la forma de aplicarlo. En general, puede ser aplicado en una gran diversidad casos de estudio en diferentes áreas. Algunas de estas son:

- Mejoras de usabilidad de interfaz o experiencia de usuario
- Mejoras de soluciones de la industria
- Mejoras de lugares de trabajo
- Mejoras de motores de búsqueda
- Mejoras de redes sociales

Para contextualizar la aplicación del servicio implementado se plantean dos escenarios relacionados con “Mejoras de usabilidad de interfaz o experiencia de usuario”, en los cuales, un usuario se puede encontrar en una situación común. Esto se realiza con el fin encaminar al lector, al entendimiento de este trabajo. Se asume que los usuarios que se nombraran a continuación tienen instalado en sus dispositivos móviles (Android), el framework AWARE y usan los componentes del servicio implementado. Además se debe asumir que dichos dispositivos disponen de los sensores requeridos en cada escenario y están correctamente configurados.

6.1 Escenario 1: Atendiendo a una presentación

Supongamos que un usuario llamado Juan, se encuentra en una reunión donde presentan los resultados de ventas de la compañía donde trabaja. Este requiere no ser interrumpido por ningún evento en su dispositivo móvil. A través del análisis de información de contexto recopilada por los dispositivos móviles, un sistema recomendador sensible al contexto puede identificar que el usuario está en una reunión (agenda/calendario) en un ambiente silencioso

(micrófono) en su oficina y no está en movimiento (GPS – localización) un martes de primavera en las horas de la tarde (reloj) y que la luz es tenue - semi oscuro (porque están proyectando información con el video vean en la sala de reuniones). Pero para este caso tiene una preferencia particular; solo debe ser interrumpido por mensajes que le envíe su esposa. Si no consideramos las preferencias de usuario, el sistema podría siempre recomendar las mismas opciones a la mayoría de usuarios con las mismas condiciones de contexto. Es por esto que también se debe hacer uso de este tipo de información.

Componentes de implementación involucrados:

Tabla 7 Escenario 1

Fuente	Nombre de variables nominales	Valores capturados
Sensor de luz (AWARE)	Nivel luz	semi oscuro
Micrófono (AWARE)	Existe ruido	No
SO Android	Perfil dispositivo	Vibrate mode
SO Android	Existe un evento agendado	Si
SO Android	Estado de wifi	On
Screen (AWARE)	Estado de pantalla	off
Location GPS (AWARE)	Hay movimiento	No
SO Android	Brillo de pantalla	?
SO Android	Orientación de pantalla	?
SO Android	Estado batería	?
Reloj	Día de la semana	Martes
Reloj	Periodo del día	Tarde
Reloj	Hora del día	2
Reloj	Estación del año	Spring

?: (no definido)

En el momento de recibir esta información de contexto, el servicio a través del método `getPrediction` clasifica dicha información contextual, generando un resultado según lo que ha aprendido de entrenamientos anteriores.

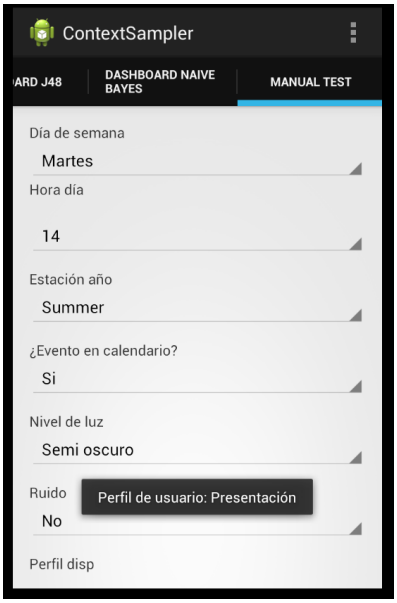


Figura 26 - Evaluación caso 1

Para complementar el caso, está información alimenta un servicio para la reconfiguración de la interacción en sistemas Android, adaptando las notificaciones al contexto (D. Sánchez Toledo, 2012). Esto logra que el usuario no sea interrumpido y permite encolar las notificaciones para una posterior lectura como se observa en la siguiente figura:

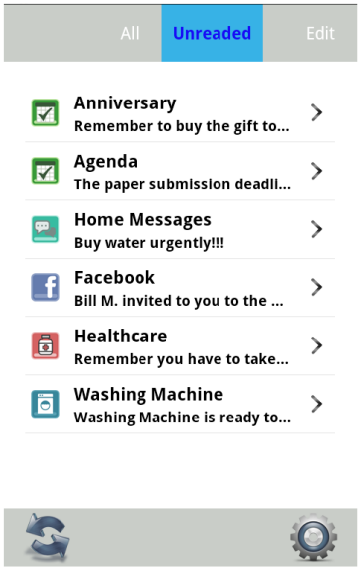


Figura 27 - Adaptación de notificaciones

En conclusión podemos decir, que el uso de este servicio en conjunto con la reconfiguración del sistema de alertas, proporcionan una solución aplicada que permite de una forma inteligente, gestionar las notificaciones de usuario y así minimizar el nivel de interrupción en los casos donde este no sea necesario aplicarlo de una forma proactiva (interrupción en circunstancias no apropiadas).

6.2 Escenario 2: Caminando donde un cliente

Pedro es un vendedor de una empresa que distribuye alimentos y bebidas; debe visitar sus clientes constantemente realizando pedidos desde su dispositivo móvil. A través del análisis de información de contexto recopilada por el dispositivo móvil, un sistema sensible al contexto puede identificar que Pedro se encuentra en la calle (GPS) a las 11:30 am, en un jueves soleado (sensor de luz) y está caminando (GPS o acelerómetro) hacia donde uno de sus clientes que lo esperan para ordenar el pedido (agenda). Gracias a las preferencias (app de ventas) y a las condiciones de contexto el sistema adapta la paleta de colores de la aplicación móvil a colores claros para que sean visibles en condiciones de mucha luz (minimizar el reflejo), podría llegar el caso a sugerir interacción vocal y sonora (si no hay mucho ruido – micrófono), para minimizar el tiempo que tiene que visualizar la pantalla del dispositivo y así evitar accidentes.

Tabla 8 Escenario 2

Fuente	Nombre de variables nominales	Valores capturados
Sensor de luz (AWARE)	Nivel luz	Incandescente
Micrófono (AWARE)	Existe ruido	Si
SO Android	Perfil dispositivo	Normal mode
SO Android	Existe un evento agendado	Si
SO Android	Estado de wifi	On
Screen (AWARE)	Estado de pantalla	off
Location GPS (AWARE)	Hay movimiento	Si
SO Android	Brillo de pantalla	?
SO Android	Orientación de pantalla	?
SO Android	Estado batería	?
Reloj	Día de la semana	Jueves
Reloj	Periodo del día	Mañana
Reloj	Hora del día	11
Reloj	Estación del año	Spring

?: (no definido)

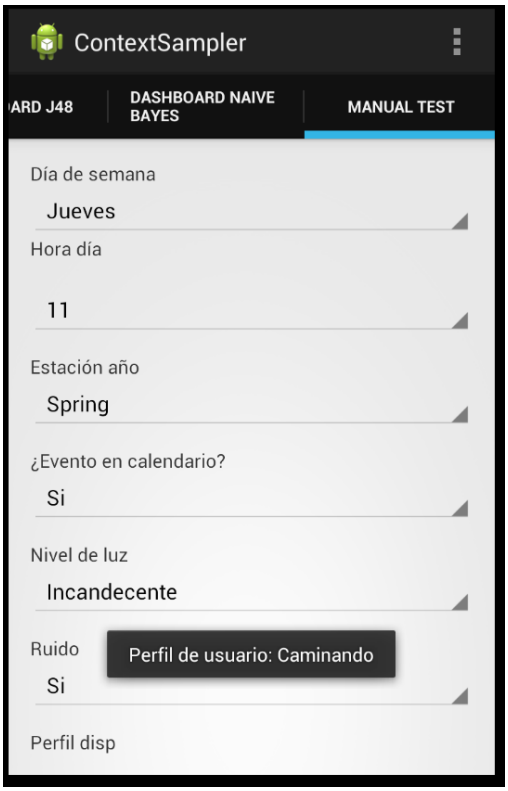
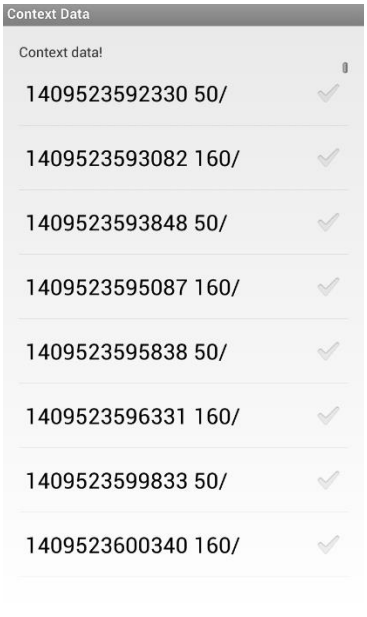


Figura 28 - Evaluación caso 2

Luego de identificar el perfil/estado del usuario (se identifica que está Caminando- ver Figura 28 - Evaluación caso 2), está información es aprovechada por la aplicación (app generar las ventas y ordenes por el móvil) que usa el vendedor, para adaptar la interfaz de usuario a las circunstancias, según se planteó en su diceño y aprovechar las bondades de la información de contexto. En más detalle, esta aplicación lo que hace es generar cambios de color del tema del background (ver Figura 29 - Adaptación de UI), dependiendo del perfil de contexto que se haya usado, todo esto de una forma automática e inteligente.

Perfil: Caminando



Perfil: Ocio

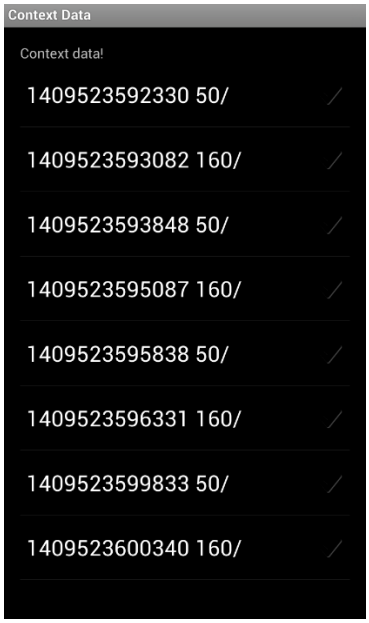


Figura 29 - Adaptación de UI

Este tipo de adaptaciones de UI, permiten aumentar la calidad de uso de las aplicaciones y en ciertos casos prevenir posibles accidentes que se puedan presentar. Cabe aclarar que este es un tema por tratar y se plantea para futuros trabajos, y hace parte de un área de investigación que actualmente se encuentra en desarrollo: HCI – interacción usuario máquina.

7. Conclusiones y trabajo futuro

Utilizando tecnologías y librerías que actualmente existen se ha podido diseñar e implementar, un servicio que logra capturar el contexto y la interacción del usuario en el dispositivo móvil, donde a partir de dichos datos recolectados se ha logrado por medio de técnicas de aprendizaje automático, la generación de conocimiento para permitir la adaptación de las aplicaciones en tiempo de ejecución.

Este tipo de acercamientos se encuentran aún en fase de investigación y existen muy pocas aplicaciones que lo implementen debido a que las tecnologías involucradas aún no han entrado en la etapa de generar herramientas o marcos contextuales que permitan ser usadas de forma global o masiva. A partir del estudio del contexto y las interacciones de usuario, se pueden identificar algunos vacíos y limitaciones, que se fundamentan en la complejidad de modelado de situaciones generales. Por el contrario, el estudio de casos puntuales logra resolver problemas específicos que genera soluciones puntuales, y es aquí donde está tesina muestra una solución.

El desarrollo de este trabajo ha supuesto un esfuerzo importante para entender y analizar la complejidad del usuario, el contexto que lo rodea y sus interacciones con los dispositivos móviles, concretamente para el sistema operativo Android. Adicionalmente, lograr una implementación en el tema de aprendizaje automático, permite aplicar muchos de los conceptos aprendidos en materias del Master como Análisis y Minería de datos. Con todo esto se logra cumplir los objetivos planteado. Es posible afirmar, que esta Tesina Final de Máster, ha sido un trabajo completo en muchas facetas del Máster en Ingeniería del Software, Métodos Formales y Sistemas de Información cursado en la Universidad Politécnica de Valencia.

Por último, la actual arquitectura y el desarrollo del servicio permiten reutilizarlo con pocas modificaciones, adicionando nuevas funcionalidades de una forma fácil.

Al hacer este tipo de implementaciones se identifican retos informáticos que pueden ser resueltos en posteriores trabajos, tales como:

- Tratamiento y análisis de grandes volúmenes de datos en los móviles.
- Generar modelos precargados, que estén optimizados y permitan, identificar mejor los perfiles o estados en los cuales se encuentra el usuario
- Análisis de las variables que tienen más peso dentro de los modelos generados
- Análisis de los clasificadores que más se ajustan a cada caso de estudio.
- Utilización de una ontología
- Adaptación mediante reconocimiento vocal
- Integración con servicio de adaptación de notificaciones
- Integración con servicio de adaptación de interfaces de usuario

Todos estos trabajos, pueden ser aplicados de manera combinada generando soluciones innovadoras que podrían comenzar a cambiar el uso de las aplicaciones, y mejorarían la interacción que los usuarios tienen con los sistemas o servicios.

Referencias

A. K. Dey, G. D. Abowd, and D. Salber (2001). "A conceptual framework and a toolkit for supporting the rapid prototyping of context- ware applications".

Alf Inge Wang1, Qadeer Khan Ahmad2 (2014). "CAMF – context-aware machine learning framework for android".

A. P. Angelbrecht (2007). "Computational Intelligence: An Introduction", 2 ed.: John Wiley & Sons.

Brahler, S. (2010). "Analysis of the Android Architecture".

CFP09ESD, ESDIS '09 (2009) "Workshop Internacional sobre Sistemas Auto-Adaptativos - IEEE Workshop on Evolving and Self-Developing Intelligent Systems".

D. Sánchez Toledo (2012) "Reconfiguración de la interacción en sistemas Android: adaptando las notificaciones al contexto".

Ferreira, D. (2013). "AWARE: A mobile context instrumentation middleware to collaboratively understand human behavior. PhD Thesis, Department of Computer Science & Engineering", University of Oulu, Acta Univ. Oul. C 458.

F. Gonzalez, C. Antonio (2010). "Context-aware mobile applications design: implications and challeges for a new industry".

G. Chen, D. Kotz, (2000). "A Survey of Context-Aware Mobile Computing Research".

G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, P. Steggles (1999) , "Towards a Better Understanding of Context and Context-Awareness", Springer-Verlag.

G. Pajares, J.M. de la Cruz (2010). "Aprendizaje automático, un enfoque práctico".

H. T. Lin, N. Koul (2011). "Learning Updatable Classifiers from Remote Data".

M. Baldauf, S. Dustdar, F. Rosenberg (2007). "A survey on context-aware systems". Int. J. Ad Hoc Ubiquitous Comput, vol. 2, pp. 263-277.

M. Miraoui, C. Tadj, C. b. Amar (2008), "Architectural survey of context-aware systems in pervasive computing environment". Ubiquitous Computing and Communication Journal, vol. 3.

M. Gil Pascual (2013). “Adapting Interaction Obtrusiveness: Making ubiquitous interactions less obnoxious”.

P. Liu, Y. Chen, W. Tang, Q. Yue3 (2012). “Mobile WEKA as Data Mining Tool on Android”.

Tom M. Mitchell (1997). “Machine Learning Hardcover”.

V. Genaro Motti, N. Mezhoudi, J. Vanderdonckt (2012). ”Machine Learning in the Support of Context-Aware Adaptation”.

Witten, I., Frank, E. (2005). “Data Mining: Practical Machine Learning Tools and Techniques.” Morgan Kaufmann Publishers. 2nd edition: 560.

Y. Lee, S.S. Iyengar, C. Min, Y. Ju, S. Kang, T. Park, J. Lee, Y. Rhee, J. Song “MobiCon: A Mobile Context-Monitoring Platform”.