Software Sensible al Contexto: definiciones y desarrollo de un estudio de caso en *Google Glass*

Galico, Diego*
Natanzon, Kevin*
Vega, Carlos*
Matalonga, Santiago*
Solari, Martín*
**Centro de Investigación e Innovación en Ingeniería de Software – CI3S*

Facultad de Ingeniería, Universidad ORT Uruguay

Mayo de 2015

Abstract

La sensibilidad al contexto es una propiedad emergente del software que indica su capacidad de adaptación al usuario, la tarea y su contexto. La sensibilidad al contexto es una tendencia de creciente importancia, sin embargo, es aún poco comprendida por los desarrolladores. En el escenario actual, donde el uso de software en dispositivos móviles se incrementa por sobre el uso de las computadoras de escritorio, se vuelve cada vez más urgente estudiar cómo los métodos de ingeniería de software responden a esta tendencia. Este trabajo tiene por objetivo analizar las definiciones relacionadas a la computación ubicua y al software sensible al contexto. Se identificaron herramientas disponibles para el desarrollo de software sensibles al contexto. Finalmente, se desarrolló un estudio de caso sobre la plataforma *Google Glass*, con el propósito de identificar buenas prácticas y desafíos para los desarrolladores. El estudio confirma la importancia creciente del software sensible al contexto. En el desarrollo del estudio de caso sobre *Google Glass* se observaron varias dificultades, debidas principalmente a la inmadurez relativa de la plataforma.

Palabras clave — computación ubicua, computación pervasiva, software sensible al contexto, *wearable technology*

Documento de Investigación, Nro. 13, mayo 2015. Universidad ORT Uruguay. Facultad de Ingeniería. ISSN 1688-6372

Software Sensible al Contexto: definiciones y desarrollo de un estudio de caso en Google Glass

Diego Galico, Kevin Natanzon, Carlos Vega, Santiago Matalonga y Martín Solari

Centro de Investigación e Innovación en Ingeniería de Software - CI3S
Universidad ORT Uruguay

***Abstract* — La sensibilidad al contexto es una propiedad emergente del software que indica su capacidad de adaptación al usuario, la tarea y su contexto. La sensibilidad al contexto es una tendencia de creciente importancia, sin embargo, es aun poco comprendida por los desarrolladores. En el escenario actual, donde el uso de software en dispositivos móviles se incrementa por sobre el uso de las computadoras de escritorio, se vuelve cada vez más urgente estudiar cómo los métodos de ingeniería de software responden a esta tendencia. Este trabajo tiene por objetivo analizar las definiciones relacionadas a la computación ubicua y al software sensible al contexto. Se identificaron herramientas disponibles para el desarrollo de software sensibles al contexto. Finalmente, se desarrolló un estudio de caso sobre la plataforma Google Glass, con el propósito de identificar buenas prácticas y desafíos para los desarrolladores. El estudio confirma la importancia creciente del software sensible al contexto. En el desarrollo del estudio de caso sobre Google Glass se observaron varias dificultades, debidas principalmente a la inmadurez relativa de la plataforma.**

Palabras clave — computación ubicua, computación pervasiva, software sensible al contexto, *wearable technology*.

I. Introducción

El desarrollo de software ubicuo comprende la construcción de aplicaciones que se ejecutan en entornos dinámicos, con la capacidad de ejecutarse en cualquier momento, en cualquier lugar, y en dispositivos capaces de requerir la mínima atención posible por parte del usuario [1]. Una característica fundamental del software que soporta dispositivos ubicuos es la sensibilidad al contexto, donde se requiere la capacidad de obtener información relevante para adaptar el comportamiento de la aplicación dinámicamente en función de su contexto. Los conceptos de *context-aware* y computación ubicua han adquirido diferentes definiciones desde sus inicios en la década de los ‘80, progresivamente abarcando nuevos conceptos como lo es el más reciente de *wearable technology*.

La construcción del software sensible al contexto es compleja, dado que además de la adquisición de la información relevante requiere el adecuado procesamiento de dicha información. Con el fin de facilitar dicho proceso, se han desarrollado distintas herramientas como *frameworks*, SDKs (*Software Development Kits*), y APIs (*Application Programming Interfaces*), tanto para soportar tecnologías específicas como para el desarrollo de software *context-aware* genérico, independiente de la tecnología utilizada. Teniendo en cuenta los distintos esfuerzos analizados y estudiados en el campo, se observa que la ubicuidad es un aspecto que los desarrolladores de software están comenzando a tener en cuenta, aunque aún la industria carece de estándares adoptados.

El objetivo de este estudio es analizar las definiciones de conceptos relacionadas con la computación ubicua y el software sensible al contexto. Con el objetivo de facilitar el proceso a desarrolladores, se analizaron distintas herramientas útiles con procedencia tanto de la industria como de la academia. Finalmente se desarolló un estudio de caso sobre la plataforma Google Glass que es un caso específico de *wearable technology*. Este desarrollo fue realizado con el propósito de identificar dificultades y desafíos prácticos para el desarrollador de software, así como evaluar el nivel de madurez de la plataforma.

II. Definiciones

La formalización de los conceptos de tecnología ubicua, sensibilidad al contexto en software (*context-awareness* de aquí en más), y *wearable technology* se remonta a los inicios de la década de los 90 [2][3]. Esto hace que la literatura académica de estos conceptos abunde y sea una fuente de información sustancial a la hora de definir correctamente estos conceptos y relacionarlos entre sí. Sin embargo, para un estudio comprensivo a la fecha el mismo necesita ser complementado con literatura técnica actualizada para una mejor comprensión de las tendencias actuales.

A. *Computación ubicua*

El concepto de computación ubicua refiere a la integración de la computación en el contexto de la persona, de forma omnipresente y no percibida como productos u objetos informáticos diferenciados. Se caracteriza por conjunto de productos y dispositivos conectados, integrados en las tareas de la vida cotidiana de las personas de forma transparente. La interacción con la computación ubicua se produce a través de diversos dispositivos y sistemas computacionales, en un

paradigma distinto al del uso de las computadoras, en el cual la interacción se encuentra dada como una tarea diferenciada y dedicada en la vida ordinaria.

El término original de computación ubicua fue acuñado por Weiser en 1988, como director del laboratorio de ciencias en computación del centro de R&D en Xerox PARC. En su definición, se encontraba consolidada su visión de un futuro en el cual la tecnología estuviera embebida en los artefactos del día a día y sea utilizada para ayudar en estas actividades. Una versión extensa del significado de computación ubicua para Weiser se puede encontrar en su ensayo *The Computer for the Twenty-First Century* de 1991 donde dice "las tecnologías de más alto impacto son aquellas que desaparecen. Se mimetizan en la vida cotidiana hasta que las mismas no se distinguen de ella" (traducción de los autores) [3]. La esencia de la visión de Weiser es que tanto las tecnologías móviles como cualquier procesador embebido, puedan comunicarse entre sí y con la infraestructura que los rodea de forma transparente.

De acuerdo a Krumm [4] computación ubicua es el término dado a la tercera era de la computación moderna. La primera era fue definida por los *mainframes*, un solo gran equipo de recursos compartidos utilizado por muchas personas al mismo tiempo en una organización. En segundo lugar, llegó la era del *Personal Computer*; una computadora personal utilizada por una sola persona. La tercera era, la de computación ubicua, se caracteriza por la explosión de pequeños productos informáticos portátiles conectados en red en forma de *smartphones*, asistentes digitales personales, y ordenadores integrados incorporados en dispositivos cotidianos no informáticos por naturaleza.

B. Sensibilidad al contexto

Por otro lado - y de forma cronológicamente posterior a la tecnología ubicua -, surge el concepto más específico de *context-awareness*. El mismo hace referencia a que las computadoras (en el sentido más amplio de la palabra) pueden sentir y reaccionar en base al entorno que las rodea. Gracias a avances en el desarrollo de sensores, los dispositivos cuentan con información acerca de las circunstancias en que están siendo utilizados y, basados en un determinado conjunto de reglas o estímulos, pueden reaccionar de forma acorde. También se puede definir como la propiedad de los dispositivos móviles que complementa la sensibilidad a la ubicación, en la cual un dispositivo móvil puede responder a cambios en su entorno en función de no sólo la ubicación del mismo, sino una buena idea del proceso o tarea que el usuario se encuentra llevando a cabo.

Según Schneiderman y Plaisant la trascendencia del modelo típico de computación personal hacia una computación *context-aware* en el diseño de los productos surge a partir de la necesidad de incorporar los entornos físicos y sociales en dicho diseño, ya que los mismos están directamente ligados a la forma en que los usuarios consumen información y hacen uso de las tecnologías de la computación [5].

Los mismos continúan diciendo que la contextualización de la computación es especialmente relevante a los dispositivos móviles y a las innovaciones en computación ubicua. Este tipo de dispositivos son portátiles y se ven inmersos en un espacio físico del cual pueden (y suelen) extraer información específica a los mismos. Esto se puede ejemplificar en la siguiente lista de cinco pares de acciones:

- **Monitorizar** presión arterial, valores en bolsa, calidad del aire, etc. y **dar alertas** cuando éstos sobrepasan ciertos umbrales.
- **Recolectar** información de, por ejemplo, reuniones programadas con otras personas y **propagar** el estado personal a cada uno de ellos.
- **Participar** en un grupo grande de actividad por medio del voto, y **relacionarse** con sus integrantes con el uso de mensajes privados.
- **Localizar** el restaurante más cercano e **identificar** los detalles de la ubicación actual.
- **Capturar** información como fotos contribuidas por otros, a la vez **compartiendo** las mías con futuros visitantes.

Suchman nota que las acciones que los usuarios llevan a cabo en sus dispositivos se ven altamente influenciadas por las contingencias del entorno, así como también la gente que los rodea. Por ejemplo, de encontrarse atascados en el uso de una determinada interfaz, los usuarios quizás recurran a otras personas que los rodean para pedir ayuda (por encontrarse cortos de tiempo y con el recurso intelectual de otras personas a su disponibilidad); otros podrían consultar un manual si lo tuvieran disponible. Las acciones de los usuarios son llevadas a cabo bajo un marco de tiempo y espacio, y éste afecta enormemente cómo ellos deciden utilizar las interfaces, en definitiva cambiando sus planes en respuesta a las circunstancias constantemente. El argumento detrás de este concepto conocido como "cognición distribuida" se basa en que el conocimiento no sólo se encuentra en las cabezas de los usuarios, sino que también se encuentra distribuido en sus entornos [6].

Krumm expresa que la sensibilidad al contexto permite a las aplicaciones comprender el entorno en el cual están siendo utilizadas, y adaptar su operación para proveer la mejor experiencia de usuario posible [4]. El contexto del usuario o dispositivo es difícil de modelar, gracias a sus muchas dimensiones (ubicación, dispositivos cercanos, personas cercanas, tiempo, variables del entorno como sonido, movimiento, temperatura, orientación, etc.). También tiene un impacto importante sobre la arquitectura del sistema de software. Esto es, cuánta operativa deberá estar implementada en el dispositivo mismo (donde tenemos la capacidad de aprovechar los sensores *on-board*), cuánta en los servicios web *back-end*, entre otras decisiones de arquitectura.

La sensibilidad al contexto por tanto se puede definir como

aquel tipo de computación que "toma conciencia" de las características estáticas y dinámicas de su entorno, y adapta su forma de operar en función de ellos con el interés de brindar una mejor experiencia al usuario (en comparación a la computación tradicional orientada a las tareas). Atiende la necesidad de incorporar aquellos factores presentes en el contexto del usuario (y por consiguiente, del dispositivo en sí mismo) que alteran o influencian la manera en que el usuario utilizará dicho dispositivo, incluyendo en definitiva aquellos fragmentos de conocimiento que se encuentran presentes en los elementos que componen al entorno mismo.

C. Tecnología wearable

El término *wearable computing* o computadora corporal refiere a la evolución de las tecnologías ubicuas hacia tecnologías electrónicas o computadoras que son incorporadas en indumentaria o accesorios existentes que se pueden portar cómodamente en la persona del usuario. Sus posibles implementaciones van desde simples podómetros en pulseras, hasta la funcionalidad de computadores personales enteros en lentes como Google Glass. Este tipo de artefactos son capaces de proveer muchas de las funcionalidades tanto de teléfonos móviles como de computadoras personales, aunque las tecnologías que soportan a esta última forma de computación tienden a ser más sofisticadas que las tecnologías portátiles. Esto se debe a que éstas necesitan proveer sensores y funciones de análisis que no son de uso masivo en dispositivos móviles y portátiles como *smartphones* o *tablets*. Ejemplos de esto son la biorretroalimentación y el seguimiento de la función fisiológica de sus usuarios. Además proveen información en tiempo real, lo que los hace particularmente útiles en ciertos contextos.

Una de las ventajas principales a la computación *wearable* es la consistencia en la experiencia de usuario que brinda por naturaleza. Esto significa que existe una constante interacción entre la computadora y el usuario (no hay necesidad de encender y apagar el dispositivo), mientras que otras tras características incluyen la habilidad de realizar varias tareas a la vez; el usuario no necesita dejar de hacer lo que se encuentra haciendo para interactuar con el dispositivo (esto es, no requiere de atención dedicada), sino que el mismo se encuentra "aumentado" en su vida cotidiana. Esto genera la nueva sensación de que los dispositivos actúan como prótesis: son extensiones del usuario (ya sea en cuerpo o en mente).

La naturaleza de los dispositivos *wearable* son un muy buen ejemplo del concepto de computación o tecnología calma (*calm technology*). En base al conocimiento proveniente del área de la neurología, la tecnología calma supone dos elementos principales en la interacción de las personas con su entorno: el centro de atención y su periferia. Surge luego el concepto de tecnología calma como aquella que es capaz de atravesar la frontera entre estos dos elementos de forma transparente para su usuario, y de forma tal que ello no exija esfuerzo alguno de éste [7].

El desarrollo de dispositivos *wearable* hereda además muchas de las problemáticas inherentes a la computación móvil, la inteligencia contextual, la gestión de la energía y disipación de calor, arquitecturas avanzadas de software, y conectividad móvil.

D. Relaciones de conceptos y tendencias

Evaluando los tres conceptos vistos, se puede notar que la computación ubicua engloba los conceptos de computación *wearable* y de la computación *context-aware*, a distintos niveles.

Por un lado se puede apreciar cómo la sensibilidad de contexto supone un subconjunto de las facultades o atributos de la computación ubicua. Tomando la definición de ubicuidad basada en la descripción de la "computación integrada al contexto de la persona", la sensibilidad al contexto se vuelve el conjunto de artefactos y recursos de hardware, software, y de conocimiento que permiten darle a la computación un componente de contextualización. Ejemplos claros de esta relación se ven clarificados por la existencia de dispositivos catalogados como ubicuos, pero que no necesariamente explotan enteramente el componente contextual del mismo.

Por otro lado, se vuelve trivial notar que la computación *wearable* compone el subconjunto de los posibles dispositivos de computación ubicua orientados a ser utilizados como prendas o indumentaria de moda. Si bien su llegada al mercado y a la escena tecnológica se inició por medio de la incorporación de computadores en prendas e indumentaria ya existente (lentes, relojes, calzados deportivos, etc.), la definición no limita el concepto a esto. Se puede por tanto considerar *wearable* a cualquier dispositivo en cualquier *form factor* bajo la condición de que éste pueda ser portado en la persona del usuario en todo momento.

La computación ubicua comprende un conjunto de distintas áreas en la rama de computación además de las vistas previamente, incluyendo computación distribuida, computación *mobile*, y computación en base a la ubicación e inteligencia artificial. Los conceptos expuestos por Schneiderman y Plaisant, en los cuales se detallan las diferentes escuelas o corrientes de pensamiento frente al avance de la interacción humano-computadora ayudan a distinguir estos conceptos desde diferentes perspectivas [4]:

- La corriente más orientada al desarrollo tecnológico se centra en que los avances se dan mediante el desarrollo de nuevos dispositivos, los cuales poseen la característica de ser ubicuos, penetrantes, y cada vez más omnipresentes en la vida cotidiana.
- Otra escuela sugiere dispositivos *wearables*, móviles, y personales; los cuales pueden ser portados por los usuarios en todo momento.

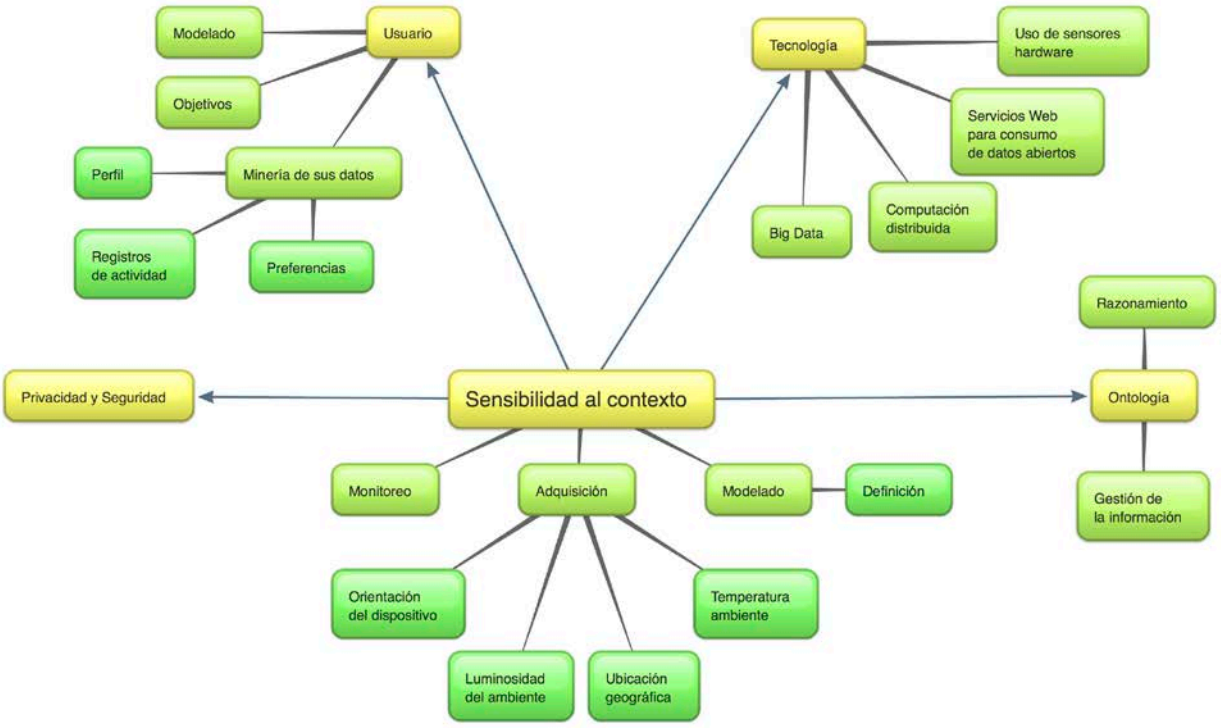

Figura 1: Mapa de conceptos de software sensible al contexto

- Una tercera escuela sugiere dispositivos embebidos, sensibles al contexto, y ambientes. Esto supone la incorporación de los mismos en elementos de la vida cotidiana de manera transparente o invisible al usuario, pero los cuales están a disposición del mismo cuando éste los necesita.
- Por último, algunos dispositivos se ven etiquetados como perceptivos del usuario, sugiriendo que los mismos perciben el estado personal del usuario y sus necesidades, incorporando la habilidad de interactuar con él de manera visual, auditiva, táctil, mediante gestos, u otros estímulos.

En la Figura 1 se puede ver un diagrama que vincula los principales conceptos relacionados a la sensibilidad al contexto: Usuario, Tecnología, Privacidad y Seguridad, Ontologías. El objetivo de este mapa de conceptos es explorar las principales líneas de investigación posibles en el área de software sensible al contexto.

III. Frameworks de Desarrollo Contextual

Como parte del presente estudio, se realizó un resumen de las APIs disponibles en donde se ven aplicados los conceptos de *context-awareness*, ubicuidad, y tecnología wearable. Se consideraron los principales frameworks comerciales con una implementación utilizable y dos propuestas académicas que aún no tienen implementación pero se consideraron relevantes. Las APIs estudiadas se detallan en orden cronológico de aparición, de las más recientes a las más antiguas.

A. *Framework comerciales*

Android Wear[1] es la versión de Android como sistema operativo diseñado para *wearables*. Según Google, Android Wear extiende a la plataforma Android a una nueva generación dentro de los dispositivos *wearable*. Android Wear SDK permite la adaptación de las aplicaciones, brindando vez interfaces de usuario personalizadas, control de sensores y acciones de voz asociadas. La API provee la posibilidad de diseñar interfaces personalizadas y *activities* que se ejecutan en el dispositivo, sin necesidad de ser únicamente una notificación externa. Además, la API permite la interacción con los distintos sensores del dispositivo, pudiendo ejecutar acciones de voz como por ejemplo "Ok Google, toma una nota", siendo esto a su vez un nuevo modelo de interacción.

Google Glass[2] es el nombre que llevan los lentes inteligentes creados por Google. Estos lentes (los cuales utilizan una versión adaptada de Android como sistema operativo), ofrecen una experiencia de realidad aumentada para los usuarios mediante una conexión visual que ofrece muchas de las características de un smartphone Android y conecta a los usuarios con las facilidades que ofrece Google en la nube como mapas, calendario, Gmail. Los Google Glass permiten comunicación con teléfonos smartphone y acceso a internet usando únicamente comandos de voz. Google

[1] http://developer.android.com/wear
[2] http://www.google.com/glass/start/what-it-does/

provee a los desarrolladores de un SDK para el desarrollo de software para Glass que comparte mucha de su arquitectura con el sistema operativo Android. Esto hace que el desarrollo de software para Glass sea un proceso más de adaptación para aquellos acostumbrados a programar para Android y no uno de re-aprendizaje. A su vez también ofrece una variedad de librerías que facilitan la implementación de diferentes aplicaciones, así como también un emulador de Android, con el fin de testear el producto sin necesidad de tener los lentes presentes. Los sensores que se pueden utilizar en el desarrollo de Glass son los mismos que se encuentran típicamente en un dispositivo Android, siendo estos los sensores de movimiento, en particular, acelerómetro, giroscopio, rotación vectorial y los sensores de gravedad; accediendo a ellos a través de la misma API. La única diferencia es que la referencia del sensor se fija con respecto al movimiento de la cabeza del usuario.

Apple iBeacon es un nuevo sistema de posicionamiento *indoors* basado en BLE (*Bluetooth Low Energy*). A través de objetos iBeacon, es posible determinar la ubicación de un dispositivo iOS con alta precisión, relativa a los iBeacon más cercanos. A raíz de esto, diversas *startups* y grandes compañías comenzaron a desarrollar herramientas para el desarrollo de aplicaciones que interactúan con iBeacons, generando así distintas soluciones para el desarrollo de aplicaciones *context-aware* basadas en sensores de proximidad BLE. Existen distintas implementaciones de iBeacons para otras plataformas, convirtiéndose así en uno de los estándares más importantes a tener en cuenta a la hora de desarrollar aplicaciones *context-aware*.

Hay diferentes *frameworks* que utilizan iBeacons, dentro de los que se encuentra Gimbal, Estimote y Swirl, entre otros. Qualcomm Gimbal ofrece dispositivos iBeacons muy pequeños, capaces de ser colocados fácilmente dentro de cualquier espacio, de forma tal de poder determinar la ubicación exacta de un dispositivo relativa a los dispositivos Gimbal. Qualcomm provee un SDK para desarrolladores, capaz de integrar los dispositivos a las aplicaciones sin esfuerzos. Además, provee una plataforma web, donde una vez integrado el SDK con la aplicación, permite gestionar el manejo de disparadores y acciones relacionadas a los dispositivos Gimbal. Dentro de las posibles acciones al detectar sensores Gimbal, la plataforma ofrece enviar mensajes push teniendo en cuenta la ubicación del dispositivo respecto al sensor, o incluso sólo por *geofencing* (delimitación de espacios por coordenadas geográficas).

Estimote provee dispositivos iBeacon más potentes (capaz de alcanzar 70 m) y con más prestaciones a la hora de determinar la ubicación de dispositivos, dado que los iBeacon Estimote cuentan con chip ARM y acelerómetro. A diferencia de Gimbal, Estimote sólo provee un SDK para integrar con aplicaciones, ya que no provee una plataforma web para gestionar la interacción.

Similar a Gimbal, Swirl provee sus propios iBeacons, un SDK y una plataforma de gestión con un *dashboard* que permite medir las interacciones con la aplicación. Se encuentra orientado a aplicaciones de retail, donde la plataforma es gestionada por un equipo no experto en IT.

En la Tabla 1 puede verse un resumen de los frameworks encontrados para el desarrollo de software sensible al contexto. En el campo observaciones se indica si el framework está ligado a una tecnología o platagorma específica.

Una de las industrias más ligadas al concepto de ubicuidad, es la industria de domótica o *smart home*. Muchas de estas nuevas iniciativas se basan en la integración y desarrollo de aplicaciones dentro de las plataformas para casas y espacios inteligentes. Los frameworks específicos de domótica han sido dejados fuera del alcance de este estudio, con la excepción del caso de Smart Things que también es mencionado como aplicable en casos generales de software sensible al contexto.

TABLA 1: FRAMEWORKS PARA SOFTWARE SENSIBLE AL CONTEXTO

Framework	Marco	Observaciones
GDK	Comercial	Solamente válido para el desarrollo para hardware Google Glass.
Android Wear	Comercial	SDK público; válido para el desarrollo sobre hardware corriendo Android Wear.
Apple iBeacon	Comercial	Define un standard basado en BLE cerrado para dispositivos iOS.
Qualcom Gimbal	Comercial	Basado en iBeacon; requiere de la integración de los Beacons Gimbal para proveer información de contexto.
Estimote	Comercial	Basado en iBeacon; requiere de la integración de los Beacons Estimote para proveer información de contexto.
Smart Things	Comercial	Limitado al contexto provisto por la dómotica.
Hermes	Académico	Propuesta de un framework para el desarrollo de aplicaciones mobile context-aware, pero sin aplicación real.
SOCAM	Académico	Propuesta de un middleware orientado a servicios para el desarrollo de aplicaciones context-aware.

B. Frameworks Académicos

Si bien un estudio basado en las herramientas comerciales vistas anteriormente ayuda a entender el estado del arte en el desarrollo de dispositivos ubicuos y software sensible al contexto, un buen complemento a éste viene dado por un estudio de las numerosas

propuestas existentes en el ámbito académico. Ciertamente, muchas de las herramientas comerciales consideradas exitosas en la industria se basan en avances y descubrimientos motivados por la investigación. El objetivo de este sub-estudio radica en encontrar y analizar trabajos provenientes de la escuela de la investigación que hayan posibilitado los desarrollos en la industria.

Uno de los primeros resultados concluye que numerosos trabajos de investigación tuvieron por objetivo facilitar el proceso de ingeniería de software para aquellos escenarios en que la sensibilidad al contexto del mismo era clave. Trabajos de este tipo proponían *frameworks* orientados a generalizar el concepto de contexto y abstraerlo a uno que fuera compatible entre aplicaciones e independiente de la plataforma donde éstas se desarrollaran. Ejemplos de estos trabajos incluyen el *middleware* orientado a servicios para la construcción de software *context-aware* propuesto por Tao Gu et al. En el artículo que soporta tal *framework* [8], el equipo describe una infraestructura que apunta a resolver la complejidad y a mitigar el alto costo de tiempo incurrido en el desarrollo de sistemas sensibles al contexto, argumentando que el mismo se ha vuelto una tarea difícil debido a una falta de soporte de infraestructura adecuado. Continúan diciendo que tal infraestructura necesitaría proveer:

i. Un modelo de contexto común que pueda ser compartido y entendido por todos los dispositivos y servicios.
ii. Un conjunto de servicios que se encargan de la adquisición, descubrimiento, interpretación, y diseminación del contexto.

Si bien este segundo punto no llegó a ser desarrollado a la medida de tener trascendencia observable en la industria, el primero logra encontrar una clara trazabilidad con el estudio metafísico de las entidades y sus relaciones; materia de estudio de la ontología.

Bajo esta correspondencia, se puede notar una aplicación clara de los hallazgos académicos en la forma de estándares en la industria. En particular, y como ejemplo, nos podemos referir la familia de lenguajes de representación de conocimiento conocido como WOL que, en conjunto con otros estándares de representación como XML, logra proponer una herramienta genérica para la fácil representación de "la realidad". Partiendo de las distintas entidades que la componen el contexto de la realidad, así como también las relaciones que existen entre ellas, tanto componentes de software como personas pueden utilizar este lenguaje para la rápida presentación e introducción al modelo del contexto en cuestión.

IV. Desarrollo de Estudio de Caso en Google Glass

El desarrollo de este trabajo de investigación incluye un caso de estudio de implementación de un prototipo funcional de aplicación sensible al contexto sobre la plataforma de Google Glass. Este caso fue desarrollado con el fin de obtener una mejor comprensión de las características del desarrollo sensible al contexto, así como las herramientas provistas por el SDK de Google Glass como plataforma de desarrollo específica. El objetivo final es recoger dificultades prácticas y lecciones aprendidas en el proceso al introducir el concepto de sensibilidad al contexto en el software.

Los principales elementos a observar en este caso de estudio fueron:

- Implicancias particulares del desarrollo de software sensible al contexto, en el marco del proceso de ingeniería de software
- Estado de madurez de las herramientas y plataforma de desarrollo para Google Glass

En base a dicho criterio, se llevó a cabo un análisis cualitativo del proceso cuyo resultado se encuentra plasmado en las siguientes secciones.

A. Aplicación desarrollada

El prototipo funcional a desarrollar corresponde al de una aplicación para dispositivos *wearable* orientada a guitarristas *amateur* para el apoyo a la lectura de tablaturas de canciones. El objetivo de la misma es facilitar dicha lectura mediante el despliegue y visualización de los acordes a tocar con una presentación de realidad aumentada, en un formato acorde al del patrón de vista HUD (*head-up display*) y de forma sincronizada con la reproducción de la canción siendo reproducida.

Una primera aproximación de los requerimientos funcionales que la misma debería cumplir se ve simplificada en las siguientes funcionalidades principales:

- Visualizar nombre del acorde a tocar
- Visualizar diagrama del acorde a tocar en guitarra
- Visualizar letra de la canción con información de progreso instantáneo
- Control básico de reproducción (*play*, *stop*, pausa)
- Reproducir metrónomo
- Configurar *tempo* del metrónomo
- Avanzar a un momento determinado en la canción
- Retroceder pocos segundos en la reproducción
- Selección de canción a reproducir
- Pausa automática al detectar silencio (el usuario perdió o dejo de tocar)
- Tutorial de cómo tocar los distintos acordes (con

reconocimiento del acorde para la evaluación del progreso y *feedback* inmediato al usuario)
- Reconocer la correctitud del acorde tocado por el usuario en el momento de reproducción de canción con metrónomo
- Afinador

El desarrollo del prototipo se realizó en un período de dos meses, durante el cual se atacarían las funcionalidades en el orden de prioridad en el que aparecen en la lista anterior. Por esta restricción de tiempo, al final del período pudieron ser implementadas las funciones relacionadas a mostrar acorde, letra y metrónomo.

Limitaciones y atajos tomados durante el desarrollo de este prototipo incluyen la restricción del catálogo de canciones a uno de una sola canción de muestra, así como también la libertad de asumir deuda técnica asociada a malas prácticas de programación como la especificación de datos *hardcoded*. Esto se considera aceptable bajo el marco del desarrollo de un prototipo desechable como lo es el del presente trabajo.

B. Resultado y lecciones aprendidas

El equipo logró bajo el marco de tiempo estipulado cubrir los cuatro requerimientos prioritarios (detallar) en un prototipo funcional sobre la plataforma Google Glass. Además de servir de prueba de concepto para la aplicación propuesta, el equipo logró identificar tanto nuevos desafíos que surgen a partir del desarrollo de software sensible al contexto, como críticas a una herramienta con el nivel de madurez como el de Google Glass.

1) Adaptación del proceso de desarrollo

A nivel de proceso, el equipo observó que el proceso de desarrollo de software más afectado es el de prueba, tanto por factores asociados a la complejidad inherente en la consideración del contexto en software, como por ciertas debilidades del kit de desarrollo de Google Glass.

Principalmente se destaca la restricción de ejecución de la aplicación de forma exclusiva a hardware Google Glass, sin la alternativa de un simulador (como lo ofrecen la mayoría de las plataformas de desarrollo para *smartphones* como iOS, Android, y Windows Phone). Esto tiene como consecuencia una alta dependencia de un recurso caro como lo es el hardware Google Glass, lo que hace que un desarrollo para esta plataforma requiera una alta inversión de capital en la compra de hardware. Por otro lado, no contar con un simulador en el entorno de desarrollo enlentece considerablemente el proceso de debug y prueba unitaria. En el caso de desarrollo de este prototipo funcional, el equipo se vio afectado por la dificultad de paralelizar la codificación y de distribuir el proceso de desarrollo a lo largo de distintas estaciones de trabajo, dado que el mismo contaba con un único dispositivo Glass.

Otras dificultades asociadas al proceso de prueba incluyen la recreación del contexto de uso del software a la hora de llevar a cabo pruebas funcionales. Si bien las condiciones de uso de la aplicación propuesta no son extremas (esto es, un usuario tocando guitarra), la simulación de los escenarios de uso posible de la misma representa un proceso costoso. Durante la construcción de este prototipo, el equipo se limitó a la prueba del mismo en un contexto controlado (el mismo lugar de trabajo); quedan pendientes de probar situaciones de uso como las de un ambiente con poca iluminación, una situación con mucho ruido de ambiente que podría tapar la señal del metrónomo. La falta de un modelo metodológico integrado que permita al desarrollador representar y simular una ejecución bajo estos distintos escenarios en un alto nivel de abstracción se considera una de las mayores áreas a mejorar.

2) Documentación de la API

Por otro lado, otro de los mayores obstáculos que se manifestaron durante el proceso tanto de aprendizaje como de desarrollo fue la escasez de documentación disponible. El equipo no contaba con experiencia previa utilizando la plataforma de Google Glass para el desarrollo de aplicaciones por lo que el desarrollo del prototipo sirvió para evaluar la curva de aprendizaje de esta tecnología. El resultado del proceso muestra que algunos de los impedimentos técnicos encontrados no tienen solución en la documentación oficial de la API al momento de realizar el caso, aunque si están disponible soluciones en foros y comunidades de desarrollo como stackoverflow[3].

Si bien la documentación oficial de Google es buena para una primera aproximación a los patrones de diseño y a las características particulares del desarrollo en Glass, en el aspecto técnico la misma no se encuentra en el estado de madurez o nivel de detalle que se espera de una plataforma de este porte. Esto se debe al carácter de producto en desarrollo que tiene Google Glass. En la Figura 2 se puede ver un ejemplo de documentación de la API donde un parámetro no tiene efecto funcional y la explicación para el desarrollador para compatibilidad está poco elaborada. Google apoya gran parte de la documentación de la API en la intersección que existe entre Android SDK y GDK (Glass Development Kit), pero existen particularidades del desarrollo en Glass que no se encuentran contempladas en la anterior.

3) Limitaciones de implementación

Durante el desarrollo se dieron dos ejemplos de

[3] http://stackoverflow.com/questions/tagged/google-glass

obstáculos que no pudieron ser resueltos utilizando la documentación oficial.

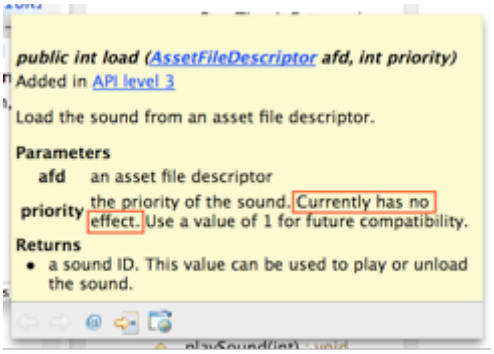

Figura 2: Ejemplo de documentación API Google Glass

Por un lado se resolvió con gran dificultad la implementación del comando de voz que activaría a la aplicación: *"OK Google, open Guitar Pro"*. Si bien la implementación del código necesario para asociar este comando a la ejecución de la aplicación era correcto, no se contaba con los permisos necesarios para utilizar comandos no registrados por Google.

Por otro lado, otro obstáculo técnico encontrado y resuelto sin la ayuda de la documentación oficial fue relativo a la reproducción de archivos de sonido. Además de las restricciones sobre los formatos de archivos de sonido soportados, al implementar la reproducción de música en el prototipo el equipo se encontró con que no se soportaba la reproducción de un archivo de 1.4 MB en tamaño por defecto, necesitando agrandar la asignación de memoria de forma explícita para soportar la reproducción de recursos de tal tamaño.

4) Selección de modelo de interfaz de usuario

Otras limitaciones encontradas durante el proceso de desarrollo surgen a partir de las restricciones impuestas sobre los distintos modelos de interfaz de usuario presentes en Google Glass. Si bien éstos se encuentran debidamente documentados (modelos de Static Card, Live Card, o Immersion), se debe prestar especial atención a los requerimientos del software a desarrollar para poder elegir el modelo más adecuado. Para este desarrollo en particular se escogió un modelo que no era adecuado para los requerimientos del prototipo en un principio (Live Card), haciendo que el dispositivo pasara a modo *stand-by* al tener éste un ángulo de inclinación inferior a 0°. La conclusión extraída es que el modelo de interfaz de usuario utilizado para el prototipo debería ser cambiado al de Immersion, ya que en el escenario de uso principal de la aplicación propuesta el ángulo de inclinación del dispositivo es generalmente negativo (el usuario se encuentra mirando el mástil de la guitarra).

V. Conclusión

En este artículo hemos presentado definiciones y una estudio de caso de desarrollo de software *context-aware*, analizando distintas herramientas y métodos de ingeniería de software relevantes a esa tendencia. En cuanto a las definiciones, se observa una convergencia de los conceptos teóricos de la computación ubicua, tecnología *wearable* y sensibilidad al contexto. Las ideas iniciales de computación calma o trasparente se vienen realizando en la práctica por la amplia distribución de dispositivos *smartphone* y software para esas plataformas. En paralelo, han aparecido dispositivos *wearables* (aunque normalmente dependientes de los *smartphones*) que permiten acceder al software desde nuevas interfaces de usuario y agregar otro tipo de sensores.

Basándonos en los esfuerzos comerciales y académicos analizados, se concluye que el desarrollo de software sensible al contexto se encuentra adquiriendo importancia de forma acelerada, y que el mismo presenta para los desarrolladores un desafío proporcional a ella. En respuesta a ello, se aprecia además surgimiento de herramientas tanto metodológicas como tecnológicas que apuntan a auxiliar las nuevas prácticas de desarrollo.

Un caso particular de dispositivos *wearable* y de software sensible al contexto es el de Google Glass. Se realizó un estudio de caso del tipo prueba de concepto, para identificar desafíos y recoger lecciones aprendidas desde un punto de vista del proceso de software. En opinión de los desarrolladores, el esfuerzo para desarrollar en esta API es significativo, debido a su relativa inestabilidad. Se concluye que esta plataforma presenta varios puntos inestables y una falta de guías concretas que pueden enlentecer el desarrollo. La aplicación desarrollada consiguió los objetivos adaptación al contexto desde el punto de vista del usuario. Sin embargo, todavía es necesario contacto con elementos de mayor nivel de abstracción y facilidades en la API para que el desarrollador pueda enfrentar con efectividad los nuevos desafíos técnicos y metodológicos que implica el desarrollo de software sensible al contexto.

Referencias

[1] Yang, K., Ou, S., Azmoodeh, M., & Georgalas, N. Policy-based model-driven engineering of pervasive services and the associated OSS. *BT Technology Journal*, vol. 23(3), pp.162-174, 2005.

[2] Schilit, B. N., & Theimer, M. M. Disseminating active map information to mobile hosts. *IEEE Network*, vol. 8 (5). 1994.

[3] Weiser, M. The Computer for the 21st Century. *Scientific American*, 265 (3). 1991.

[4] Krumm, J. *Ubiquitous Computing Fundamentals*. Chapman and Hall/CRC. 2009.

[5] Shneiderman, B. & Paisant, C. *Designing the User Interface*. Addison-Wesley 2009.

[6] Suchman, L. *Plans and situated actions: the problem of human-machine communication*. Cambridge University Press. 1987.

[7] Weiser, M., & Brown, J. S. The Coming Age of Calm Technology. *Beyond calculation*, pp.75-85. Springer. 1997.

[8] Gu, T., Pung, H. K., & Zhang, D. Q. A service-oriented middleware for building context-aware services. *Journal of Network and Computer Applications*, vol. 28 (1), pp.1-18, 2005.