UNIVERSIDAD DEL PAÍS VASCO
EUSKAL HERRIKO UNIBERTSITATEA

Programa: Ingeniería Informática
Área: Arquitectura y Tecnología de Computadores
Línea: Sistemas Informáticos Distribuidos

Un análisis sobre aplicaciones distribuidas dependientes del contexto

Doctorando:

David Sainz González

Tutor del trabajo:

Alberto Lafuente

San Sebastián, enero de 2009

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

Este documento es un análisis de los elementos necesarios que deben contener las aplicaciones distribuidas context-aware. Su tipo de arquitectura y principales características. También contiene un análisis de diferentes frameworks útiles para realizar este tipo de aplicaciones. En general, la estructura del documento viene reflejada a continuación:

- En una primera parte se dan definiciones y descripciones de los conceptos básicos, tales como contexto, computación ubicua, context-aware...

- Seguidamente se hace un análisis de diferentes arquitecturas que se utilizan para este tipo de aplicaciones, junto con diferentes frameworks, aplicaciones y sistemas que las utilizan.

- Tras el análisis, se propone un modelo de arquitectura ideado especialmente para dispositivos móviles.

- De este análisis y definición de arquitectura surgen importantes conceptos que pasan a analizarse en la siguiente sección. La medición de la Calidad de Contexto (QoC), las técnicas de selección de contexto relevante y las formas de modelado de contexto son aspectos que se analizan más en detalle en esta sección.

- En la siguiente sección se habla de aplicaciones context-aware para dispositivos móviles, dirigidas a realizar Reality Mining. Este concepto se basa en el uso el análisis del contexto móvil para averiguar personalidad y datos interesantes sobre usuarios. Se explica más en profundidad en la sección.

2. COMPUTACIÓN UBICUA Y CONTEXT-AWARENESS

Para entender de una forma más clara los elementos de estudio de este trabajo, es necesario detallar previamente unos conceptos para una correcta puesta en situación.

2.1. COMPUTACIÓN UBICUA

El primer concepto básico que debe ser entendido es el de computación ubicua. Se trata de un modelo de computación que varios autores han descrito. Uno de los primeros en visualizarla fue Weiser [1], quien vio los elementos de computación como entes fundidos con el entorno, de forma que cada objeto de la vida diaria (tanto una lavadora como una cuchara) tuviera capacidad de proceso y supiera adaptarse a las necesidades del usuario que se encuentra en el medio. Objetos normales, ropa e incluso el propio cuerpo adquieren capacidades de proceso, lo que cambia completamente el paradigma que hasta ahora se ha tenido en computación. Ahora los elementos serían inapreciables, indistinguibles del medio en el que vivimos, pero no por ello dejarían de colaborar entre ellos para realizar tareas aún más complejas. Con esa visión se acuñó el término, y numerosos autores siguieron después definiendo y dando forma al concepto, aunque realmente aún no se ha llegado a crear la visión que Weiser creó. Los elementos de proceso están todavía lejos de fundirse con el entorno hasta el punto de pasar desapercibidos. Este paradigma también implica que los elementos aumentados se comuniquen unos con otros y por tanto lleguen a una forma de computación distribuida, aunque como parte del entorno. Abowd [2] describe Computación Ubicua como computación distribuida “fuera del escritorio”, donde los elementos comunes son aumentados con capacidades de proceso y realizan tareas tanto por separado como de forma colaborativa.

Abowd también menciona que para la evolución de la computación ubicua es necesario avanzar en ciertos aspectos:

- **Factor de forma**: Para que los objetos posean esas características aumentadas, es necesario que la tecnología hardware evolucione hacia elementos más miniaturizados, además de evolucionar en la forma física. Grandes avances se han conseguido en este campo, aunque tampoco se ha llegado aún a un estado suficiente para este paradigma. Este amplio concepto queda fuera del ámbito de este trabajo.

- **Interfaces naturales**: Muchos de los objetos deben ser manejados por la gente que se encuentra en el espacio, por lo tanto se hacen necesarias nuevas formas de interacción con nuevos objetos, fuera de los habituales medios de escritorio a los que estamos acostumbrados. Elementos como reconocimiento del habla, reconocimiento de ondas cerebrales y en general todos los avances en HCI, serán necesarios para llegar a interactuar con objetos aumentados de forma natural.

- **Capacidad de obtener información del entorno**: Los objetos aumentados se funden con en el entorno y conviven con él, por tanto, además de interactuar con personas, han de estar provistos de medios para interactuar con el medio en el que viven. Ello implica capacidad para captar toda la información que necesiten del medio. Esta información recibe el nombre de contexto, y este será e tema principal en el que se centrará este trabajo.

- **Recoger experiencias a lo largo del tiempo**: En este paradigma se recoge la interacción con el usuario a través de las interfaces naturales anteriormente mencionadas. Como consecuencia de ello, se deduce que es necesaria alguna forma automatizada y estándar de recoger las experiencias, como las interacciones hombre-máquina a lo largo del tiempo. Sea cual sea la forma de experiencia y su contexto, se hace vital una forma de adquirirlo y almacenarlo, teniendo en cuenta también que el tiempo es un factor importante. Los objetos aumentados en su mayoría deben de recoger experiencias de una forma continua en el tiempo, incluso hay experiencias que no pueden ser definidas en un instante concreto. Una experiencia de interacción con un objeto aumentado que defina un acto de comunicación entre dos personas, ha de durar mientras el acto de comunicación perdure, tendiendo en cuenta también que no importa la hora del día, el objeto ha de estar preparado para recoger la experiencia en todo momento. Una vez recogida la experiencia, también será necesario proveer de una forma uniforme de permitir el acceso a ella en el futuro.

Milner [3] completa la definición de Computación Ubicua mencionando capacidades que los objetos aumentados deben poseer:

- Tomarán decisiones hasta el momento hechas por nosotros, lo que implica avances también en el campo de inteligencia artificial.
- Por su naturaleza, podrán estar en cualquier objeto, con lo que su número será muy superior a los sistemas actuales.
- Continuamente se adaptarán a nuevos requisitos, a lo largo del tiempo.
- Las unidades de proceso interactuarán unas con otras, lo que convertirá la Computación Ubicua en un inmenso sistema distribuido.

El número de objetos aumentados será órdenes de magnitud mayor al número de sistemas al que estamos acostumbrados actualmente.

El masivo crecimiento de estos elementos de proceso invierte la proporción de personas y computadores. En los inicios un gran computador trabajaba para muchas personas, más tarde se evolucionó hacia un computador menos potente para cada persona. El siguiente paso en la evolución será, según Xerox [4] la tercera oleada (Third Wave) de computación, en la que una persona tenga enormes cantidades de computadores de minúsculo tamaño a su disposición.

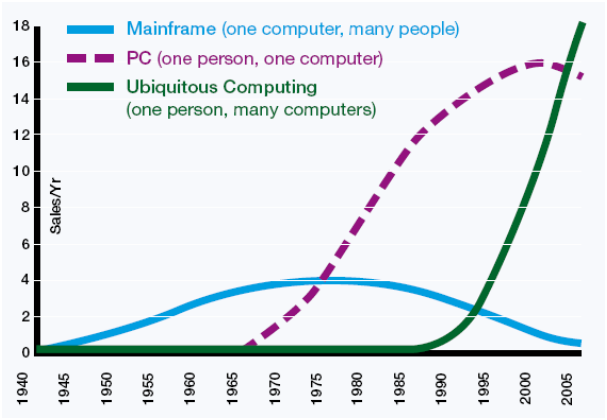

Figura 1. Las tres oleadas de computación a lo largo del tiempo [4]

Ley [5] afirma que dentro de este paradigma hay nociones que ya están desarrolladas pero necesitan serlo específicamente para este modelo de computación:

- Conectividad: Definir interconexiones entre elementos de forma Ubicua
- Interoperabilidad: Estándares de redes y dispositivos, lo que conlleva técnicas de descubrimiento, identificación, auto configuración, etc.
- Seguridad y confiabilidad: Privacidad, seguridad y creación de sistemas fiables.
- Sistemas inteligentes: Técnicas de inteligencia Artificial para que los objetos aumentados se adapten y razonen para los usuarios.

Con ayuda de los nuevos factores de forma, en la universidad de Berkeley iniciaron un proyecto que muestra de una forma clara el cambio de paradigma: Smart Dust [6], que crea una red de sensores de un tamaño razonablemente pequeño capaces de interactuar entre ellos. El concepto de Smart Dust muestra este cambio en el que el “polvo” se difumina en el entorno pasando desapercibido, pero adquiriendo capacidades de proceso y comunicación.

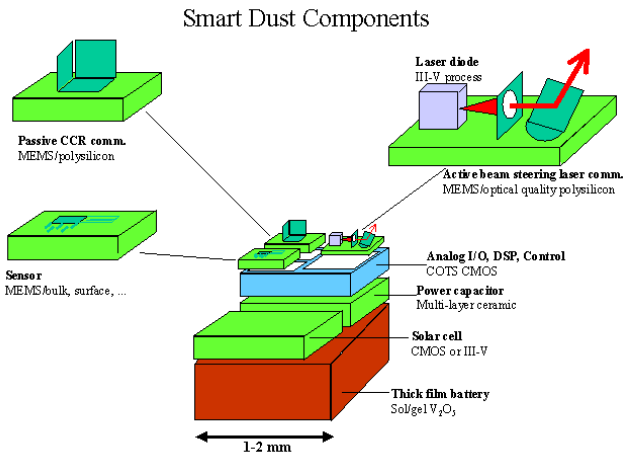

Figura 2. Arquitectura Smart Dust de Berkeley (Fuente http://robotics.eecs.berkeley.edu/~pister/SmartDust)

La tecnología y la investigación científica sigue adelante para forjar este concepto, por ello, es uno de los 6 retos científicos que han sido planteados por el UK Computing Research Comitee [7]. El reto, llamado UbicompGC [8] propone abarcar 3 perspectivas:

- El problema de la **experiencia**: Los seres humanos no estamos acostumbrados a este paradigma. Esto implica también nuevas interfaces.
- El problema del **diseño**: Creación de principios de diseño e ingeniería del software
- Aproximación **científica**: Crear principios y teorías científicas para abarcar correctamente los dos problemas anteriores.

Tanto el reto como Weiser proponen áreas de desarrollo de proyectos:

- Computación Ubicua en el Entorno Urbano
- Computación Ubicua en el entorno de la salud
- Análisis del movimiento en un entorno
- Automatización de carreteras y autopistas
- Técnicas de análisis de los modelos de Computación Ubicua
- Diseño riguroso de protocolos

2.2. CONTEXTO

Fuera del ámbito de la Computación Ubicua, cualquier representación o entidad está ligada a un contexto. Una representación (como una sentencia en el ámbito del habla) no puede ser totalmente representada únicamente enumerando cada una de sus partes, porque siempre habrá información implícita [9]. Un ejemplo de representación puede ser una sentencia como “está lloviendo”, que tiene la información implícita relevante de tiempo y lugar. Esta dependencia del contexto está formalmente definida en la “metáfora de la caja” [10], en la que la información se representa por las sentencias e informaciones que la componen, dentro de una caja, y los parámetros de los que depende (Pi) fuera de la caja. El valor de los parámetros determina, al menos parcialmente, el significado de las sentencias.

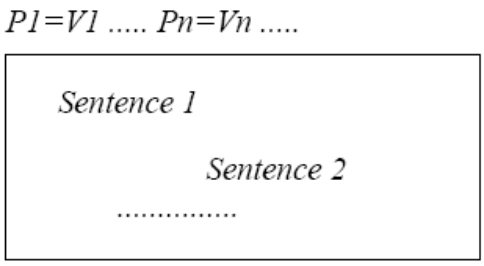

Figura 3. Metáfora de la caja [9]

Según esta metáfora, se puede encajar un ejemplo de frase sacada de contexto: “Coge uno”. La frase y el hecho de que el objeto a coger es de género masculino son sentencias que se extraen de la propia frase y que están dentro de la caja. Los parámetros fuera de la caja simbolizan el contexto que rodea a los hechos de la frase: situación, entorno etc. Si uno de los parámetros fuera “Contenido de la caja=Bombones” la frase queda definida de una forma muy diferente a si uno de los parámetros fuera “Contenido de la caja=cuchillos de combate”. Los parámetros forman parte del significado pero no se explicitan en el contenido de la frase.

Varios autores han definido la palabra contexto dentro del ámbito de la Computación Ubicua, y con el paso del tiempo esta definición ha ido evolucionando hasta formas más generales. Schilt [11] lo limitaba básicamente a la información de localización, ya que es una de las principales fuentes de información. Aún así, no es la única, siendo este concepto mucho más amplio. Ryan [12] y Brown [13] añaden a esta información de localización, los elementos de usuario, tiempo y ciertas nociones de temperatura, estación del año etc. Dey [14] añade además de lo anterior, estados de animo del usuario, orientación y grado de atención, además de otros objetos y gente alrededor del usuario.

Estas definiciones son primarias y muy concretas, cada una va un paso más allá que la anterior y completa de alguna manera la definición, pero realmente quedan limitadas a ámbitos concretos de aplicación. Frameworks como los stick-e notes [15] generalizan hablando del entorno completo de un usuario y lo que hay alrededor de él.

Según se avanza en la Computación Ubicua, el concepto de contexto se generaliza y autores como Schilit et al. [16] dan pasos definitivos a descripciones más generales y ambiciosas. Para Schilit et al. el contexto puede resumirse respondiendo a las preguntas ¿Dónde estoy? ¿Con quién estoy? Y ¿Qué objetos tengo a mi alrededor?

Otras definiciones [17] apuntan que el contexto es una serie de objetos complejos que contienen información relevante al problema o dominio que se examina.

Una buena definición de contexto la dan Abowd et al. [18], quienes argumentan que el contexto es cualquier información que pueda ser utilizada para caracterizar la situación de una entidad. Una entidad es una persona, lugar u objeto que es relevante para la interacción entre el usuario y la aplicación, incluyendo al usuario y a la aplicación propiamente dichos.

Esta definición comprende cualquier tipo de información que los autores anteriores describen (tiempo, factores climatológicos, localización, estado de ánimo, estado de la máquina etc.) y da pie a nuevas informaciones de contexto relevantes.

Otra definición de contexto que podría englobar los aspectos tan generales y el amplio abanico de ámbitos podría ser:

“El conjunto de información estructurada que describe las entidades que pertenecen a un universo concreto y el estado las mismas, siendo estas entidades personas, lugares u objetos considerados relevantes para el universo.”

Una vez definido el contexto de manera general, se han diferenciado diversas categorías de contexto para ayudarlo a estructurarse de una manera organizada. Ryan et al [12] clasifica la información de contexto en los elementos de:

- Localización
- Entorno
- Identidad
- Tiempo

Esta clasificación denota elementos muy importantes en información, sin embargo no abarca todo el contexto al que se refieren las definiciones más amplias. Por ejemplo, el estado de ánimo de un usuario es una información propia de contexto pero es difícil de encajar en esta clasificación.

Además, el elemento de "entorno" es demasiado ambiguo y bien podría incluso representar un sinónimo del mismo concepto de contexto que se intenta categorizar. Abowd et al. [19] recogen la categorización anterior, pero sustituyen el concepto de "entorno" por el de "actividad", que da una información más definida de qué está pasando dentro del entorno.

La información referente a un contexto que cae dentro de alguna de las categorías, forma un sub nivel dentro de esta clasificación. Es decir, con la descripción categórica anterior se obtiene el nivel **primario** de contexto. El nivel **secundario** de contexto será el tipo de información obtenida y que cae dentro de una de las categorías. Así, información de DNI, nombre y correo electrónico de una persona encajan dentro del contexto primario "identidad". Identidad sería el nivel primario mientras que la información "DNI" pertenecería al nivel secundario. Esta categorización, pese a no ser completa, ayuda a organizar el contexto obtenido en diferentes niveles jerárquicos, de forma que el acceso y la representación sean más fáciles y organizados.

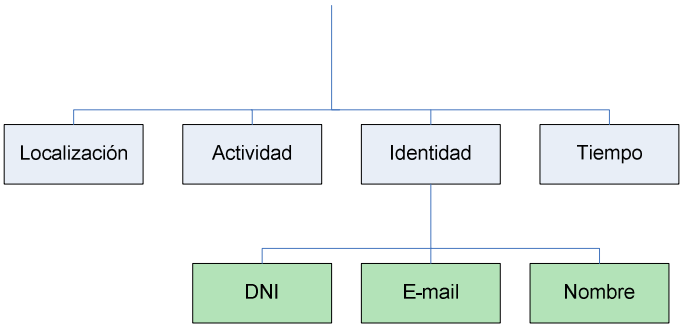

Figura 4. Contexto primario y secundario de Abowd et al. [18]

Para Schilit et al. [16] el contexto se agrupa en diferentes tipos, dependiendo de sus características:

- Contexto de usuario: Tanto su localización, como estado de ánimo, situación social etc.
- Contexto de computación: Recursos disponibles en las máquinas, procesos ejecutándose, espacio libre, memoria, dispositivos disponibles, medios de interacción con ellos etc.
- Contexto del entorno: temperatura, nivel de ruido, etc.

Esta clasificación es más amplia y puede abarcar mucha más información de contexto. Tanto esta versión más extendida como las anteriores, se refieren al origen del contexto, es decir, la información está clasificada dependiendo de a qué entidad pertenece la información. Por ello bien podrían ser clasificaciones *morfológicas*, en cuanto a que dependen de la forma.

Tras las clasificaciones morfológicas, existen otros tipos de clasificaciones que agrupan informaciones de contexto según el grado de complejidad del mismo. Algunos autores [17, 20] proponen clasificar de contexto en lo que llaman **interno** y **externo**.

- El contexto externo se refiere al contexto directo fácilmente adquirible por sensores, en la forma deseada. Ejemplos son temperatura, localización, luz, sonido, movimiento, etc. Está muy relacionado al contexto relativo al entorno exterior al usuario.

- El contexto interno se refiere al nivel interno del usuario, que describe su status. Toda información, pertenezca o no al propio usuario, que describa su estado, entra dentro de esta catalogación interna. Ejemplos son estado de ánimo, objetivos e intenciones, procesos de negocio etc.

Puede verse fácilmente que el contexto interno es inherentemente más difícil de obtener y en muchas ocasiones viene dado por un proceso previo del contexto externo, que es preprocesado para concluir un valor. Esta clasificación no está reñida con las clasificaciones morfológicas, ya que un elemento de contexto puede clasificarse bajo ambas visiones. Por ejemplo una información puede ser externa y a su vez secundaria de localización.

Esta clasificación es matizada [21] en contexto **lógico** y **físico**:

- El contexto físico es similar al contexto externo anteriormente citado, se refiere asimismo a la información del ambiente en general, en valores directamente captados por sensores y que normalmente es refrescada frecuentemente para tener siempre un valor adecuado de la situación actual.

- El contexto lógico sería un contexto derivado del contexto físico, un valor más abstracto que se obtiene de razonar y procesar el contexto físico. Más difícil también de obtener pero mucho más complejo y con mucho más significado. Estados de ánimo de un usuario o grado de adecuación de un invernadero para un tipo de platas son ejemplos de contexto lógico.

Guan et al. [22] proponen otra clasificación más general que las anteriores, en la que no se hace diferencia entre interior, exterior o perteneciente al usuario. Ellos diferencian entre contexto de **alto nivel** y contexto de **bajo nivel**:

- El contexto de bajo nivel es toda aquella información adquirible y almacenable sin necesidad de procesado o combinación. Los valores que se leen u obtienen vienen directamente en la forma deseada y por tanto el nivel de complejidad es bajo. A diferencia del contexto físico, no tiene por qué estar relacionado con valores físicos del entorno.

- El contexto de alto nivel es el resultado de razonar sobre el contexto de bajo nivel y obtener conclusiones relevantes. Por sus características, contiene en sí mismo mucha más información y resulta más relevante y valioso para las aplicaciones. Está muy relacionado con el concepto de contexto lógico.

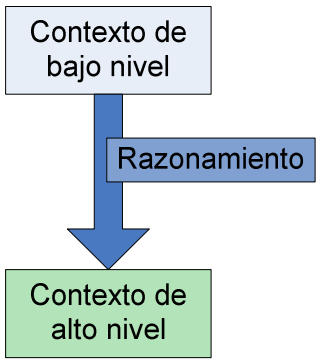

Figura 5. Categorización del contexto según Guan et al. [22]

El contexto recogido por Schilit et al. es una buena aproximación de clasificación morfológica, mientas que la categorización de Guan et al. es una forma general a la vez que muy informativa de categorizar el contexto por su nivel de complejidad.

Gwizdka enumera tras su análisis del contexto [23] los 4 principales usos que las aplicaciones hacen de él:

- Contexto como una entrada adicional de datos: El contexto se utiliza con entrada de datos que la aplicación utilizará en su beneficio. Por ejemplo, se puede utilizar el contexto del nivel de ruido de una habitación para ajustar las preferencias de sonido al recoger el audio de un usuario.

- Contexto como elemento que modifica la entrada de datos: El contexto es información que sirve para interpretar la entrada de datos de usuario de forma correcta. Un ejemplo podría ser averiguar la

localización del usuario para poder interpretar si se trata de una entrada de datos relacionada con el trabajo o con el ocio, y en consecuencia guardarlas en carpetas diferentes.

- Contexto como información que vuelve al usuario: Se crea un bucle en el que el contexto es proporcionado por el sistema de vuelta al usuario para que este lo interprete y pueda o no modificar su entrada de datos manual. Un caso práctico sería un sistema de monitorización en el que se deduce un contexto actual de anomalía en una máquina. Esa información se devuelve al usuario quien la interpreta y decide en base a ello qué acción tomar de vuelta en el sistema.

- Contexto como un disparador de eventos: El sistema recibe la información de contexto, la analiza y si alguna condición se vuelve verdadera se dispara una serie de eventos.

Figura 6. Contexto como entrada adicional de datos [23]

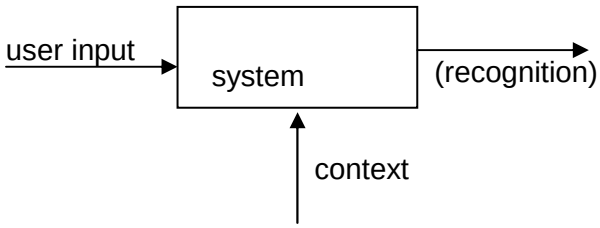

Figura 7. Contexto como elemento que modifica la entrada de datos [23]

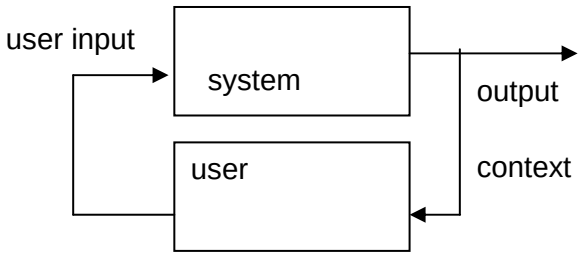

Figura 8. Contexto como información que vuelve al usuario [23]

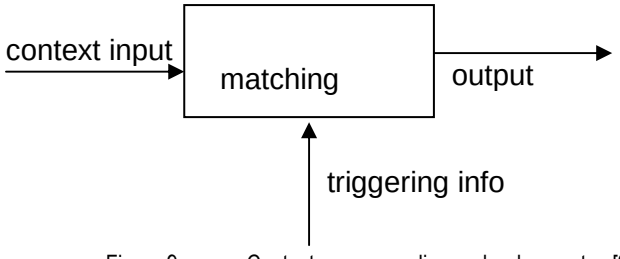

Figura 9. Contexto como un disparador de eventos [23]

2.3. CONTEXT-AWARE

Tras esta definición y puesta en situación acerca del contexto, es necesaria una definición de aquellas aplicaciones que son context-aware, es decir, traducidas como conscientes del contexto. Así como la definición de contexto en sí misma ha ido evolucionando de una forma más concreta a una más general, según la experiencia demostraba la gran amplitud de tipos de información posibles, la definición de context-aware ha ido de forma sincronizada evolucionando. Schilit et al. [11] lo definen como un software que se adapta dependiendo de la localización, objetos y gente cercana, así como los cambios de estado de esos objetos en el tiempo.

Al haber visto la evolución del concepto contexto, es fácil ver que esta definición de context-aware es asimismo muy limitada, teniendo en cuenta pocos aspectos de información, además de decir que únicamente de “adapta” a él.

Una de las primeras aplicaciones context-aware que pudieron verse fue Active-Badge [24] en 1992, un sistema de localización en entorno de oficina. Con la puesta en práctica de aplicaciones el concepto fue evolucionando. Aparecieron sinónimos como Situated Computing [25] que se define como la capacidad del software para detectar, interpretar y responder a los aspectos del entorno del usuario. Un paso más adelante: aunque aún ligado al concepto de usuario y ambiente alrededor de él, ya se habla no sólo de detectar el contexto, sino de interpretarlo y dar una salida respecto a él. Otro sinónimo utilizado por algunos autores [26] indistintamente a context-aware es context-sensitive.

Autores como Flickas [27] definen una aplicación context-aware dando un sentido más completo y eliminando por fin la limitación de que el contexto fluye siempre alrededor de un usuario. Usando el sinónimo de environment-directed, definen una aplicación context-aware como software que monitoriza el estado del entorno y adaptan su funcionamiento con respecto a él, siguiendo unas guías y preferencias definidas por el usuario. Schilit y Theimer [11] tienen una definición de estas aplicaciones como sistemas que descubren cambios en el entorno en el que están situados y reaccionan a ellos. Mientras esta descripción puede acerarse al ámbito general, realmente un sistema no tiene por qué reaccionar únicamente a los cambios del entorno. Una aplicación puede ofrecer información de contexto que no cambia y ser context-aware.

Brown et al. [13] ven las aplicaciones context-aware como sistemas que cambian su comportamiento dependiendo del contexto que capten en cada momento. Así, estos sistemas están principalmente dirigidos por información de contexto. Abowd et al. [18] definen un sistema context-aware como un sistema que utiliza el contexto para proporcionar información relevante y/o servicios al usuario, donde la relevancia depende de la tarea del propio usuario.

Estas dos últimas descripciones son suficientemente generales para poder abarcar los sistemas context-aware dentro de la ya amplia definición del propio contexto.

Tras la definición de sistemas de este tipo, podemos ver que es un concepto muy relacionado con la Computación Ubicua. Si los elementos de computación han de mezclarse y fundirse con el entorno, tenerlo presente y recoger su información, realmente las aplicaciones que se ejecuten dentro de este paradigma, de alguna u otra forma, en mayor o menor medida, serán aplicaciones context-aware.

Para entender aún mejor este concepto, es posible categorizar las aplicaciones context-aware y crear una taxonomía de las mismas. Schilit et al. [16] definen también una taxonomía que tiene dos principales dimensiones. La primera dimensión se refiere a si las aplicaciones tienen como objetivo recoger o presentar información, o es ejecutar un comando. La segunda dimensión se refiere a si las tareas son ejecutadas automática o manualmente. Dependiendo de esto, los autores definen cuatro tipos principales de aplicaciones:

- **Selección por cercanía**: este tipo es más bien un medio de interfaz con el usuario, en el que los objetos que están alrededor de él son seleccionados o de alguna forma se enfatizan o se hacen más fáciles de elegir que el resto. Se da muy comúnmente en entornos de oficina, para objetos que dependen de la cercanía del usuario (impresoras, altavoces, pantallas). Así, un ejemplo puede ser una aplicación que se representa en la pantalla de una PDA y mantiene información de la localización de usuario, de forma que cuando este cambia de lugar, la aplicación siempre procura reflejar en su interfaz los elementos más cercanos. En un mapa, se enfatizan los servicios que el usuario pide y que además están cerca de donde se encuentra. En este tipo se muestra información y su selección es de forma manual.

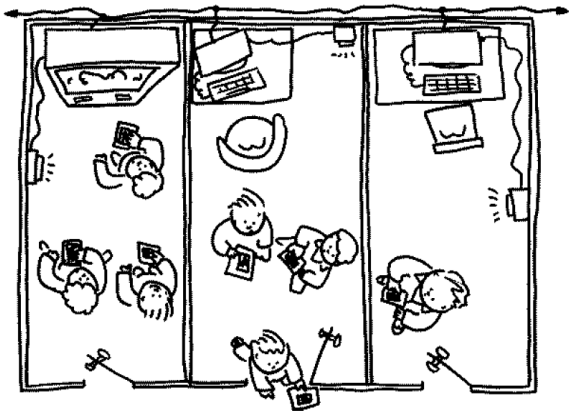

Figura 10. Ejemplo de "selección por cercanía" de Schilit et al. [16] ejemplificado en sonido

- **Reconfiguración contextual automática**: Los componentes que forman el espacio se reconfiguran dependiendo del contexto. Con reconfiguración se puede entender añadir o retirar componentes, y eliminar o crear conexiones entre ellos. Un ejemplo es cómo se configura una sala de reuniones dependiendo de la gente que esté en ella. Si entran personas se crea componentes que las representen y se crean conexiones entre ellos. Además, se crea una conexión entre cada persona y la pizarra. Después la pizarra pasa a estado activo. En este tipo se recoge información de contexto y las tareas se hacen de forma automática.

- **Comandos contextuales**: Son aplicaciones que ejecutan comandos de forma manual por el usuario, pero dependiendo del contexto. De esta forma, un comando imprimir puede ser seleccionado manualmente y por defecto, basándose en el contexto, imprimirse en la impresora más cercana.

- **Acciones originadas por el contexto**: En este tipo de aplicaciones, se ejecutan comandos automáticamente por el sistema basándose en el contexto. Normalmente implican reglas o algún tipo de base de conocimiento, unido con preferencias de usuario. Se da en sistemas donde, por ejemplo, la luz se enciende automáticamente al llegar una persona a la habitación.

	manual	automatic
information	proximate selection & contextual information	automatic contextual reconfiguration
command	contextual commands	context-triggered actions

Tabla 1. Taxonomía de aplicaciones context-aware para Schilit et al. [16]

Para Pascoe [28] las aplicaciones context-aware se clasifican por las características inherentes del contexto más que por el propio uso o servicios de la aplicación. Este tipo de clasificación es a su vez compatible con la clasificación dada anteriormente por Schillit et al. [16]. Su clasificación se divide en:

- **Captación** de contexto: La forma más simple de aplicación context-aware. El contexto se recoge de los sensores y se presenta de forma adecuada en la salida. No hay procesado previo, a lo sumo se le da un formato a la salida.

- **Adaptación** contextual: El sistema se adapta a sí mismo dependiendo del contexto. Más que utilizar la información para presentársela al usuario, se utiliza para concluir qué acciones de adaptación deben tomarse.

- Descubrimiento contextual de **recursos**: Los sistemas que caen en esta categoría van un paso más allá de la adaptación contextual. Estas aplicaciones son conscientes de otros recursos a su disposición. Realizan búsquedas en el entorno y encuentran objetos que pueden ser usados por ellos o con los que pueden colaborar. Un claro ejemplo es una PDA que busca en una habitación pantallas disponibles para presentar la salida de datos.

- **Aumento** contextual: Estos sistemas son más complejos y además de captar, procesar y reaccionar al contexto, aumentan sus datos con información de la realidad (de contexto) o incluso bidireccionalmente aumentan la representación de la realidad con los datos de contexto. En una aplicación para turistas, la información del sistema varía según el contexto cuando se muestran en un mapa los puntos de interés que rodean al usuario. Por otra parte, cuando una pantalla en una sala de reuniones se enciende a consecuencia de la llegada de gente, el contexto es aumentado gracias al sistema.

Esta última categorización aporta elementos nuevos al concepto de context-awareness que la primera clasificación no contempla: el aumento del contexto. Hacer que la información del sistema aumente en relación al contexto y que incluso el propio entorno aumente su información a causa del sistema.

Brown e al. [13] proponen otra clasificación basada en cuán rápido cambia el estado del contexto. Para Brown et al. las aplicaciones se dividen en dos tipos:

- **Continuas**: El contexto de estas aplicaciones cambia continuamente y es necesario consultarlo en cortos intervalos de tiempo. Aplicaciones de localización son típicamente continuas.

- **Discretas**: El contexto apenas varía o no lo hace en absoluto. Ejemplos son aplicaciones que manejan el estado de los muebles de una habitación.

Las aplicaciones continuas son por naturaleza más difíciles de implementar y pueden incluso requerir características de aplicaciones en tiempo real. Las que en cambio son discretas no tienen necesidades tan exigentes de tiempo.

Dey y Abowd [18] presentan otra clasificación basada en el uso que se le da al contexto. Tiene similitudes con la taxonomía de Pascoe, pero la matiza de la siguiente manera:

- **Presentación de servicios o información al usuario**: El usuario recibe la información el contexto, en las formas adecuadas, tanto de bajo como de alto nivel. Esta información de contexto puede representarse también como objetos que ofrecen a su vez servicios.

- **Ejecución automática de un servicio**: Los sistemas de esta categoría utilizan la información de contexto para lanzar servicios al cumplirse ciertas condiciones. Si es requerido, se hace respetando preferencias de usuario.

- **Uso de información de contexto para etiquetar datos**: Estos sistemas usan la información de contexto para poder añadirla a contenido y datos que el usuario y sistema genera o recaba.

Esta taxonomía no es excluyente en cuanto a que una misma aplicación puede ser clasificada bajo varias categorías al mismo tiempo, ya que puede tener un uso mixto de todo el contexto que le rodea. Las categorías presentadas aportan un factor muy importante, que es el etiquetado de datos con la información de contexto. En otras palabras, lo que define como aumento contextual de la información de la aplicación.

Sin embargo, esta última taxonomía adolece de una carencia que otras clasificaciones abarcan: la adaptación de la aplicación y su configuración con respecto al contexto. Aplicaciones que se limitan a adaptar sus características dependiendo de factores externos no encajan en esta clasificación y sin embargo sí son context-aware.

Una clasificación más completa quedaría de la siguiente forma:

- Presentación de servicios o información al usuario basados en el contexto

- Ejecución automática servicios basada en el contexto

- Adaptación al contexto

- Uso de información de contexto para etiquetar contenidos

2.4. FRAMEWORK

Los sistemas Context-Aware están en plena evolución, tienen diversos usos y un amplio abanico de posibilidades. Para evitar la situación en que cada autor comience el desarrollo de su aplicación Context-Aware desde el principio, mucho trabajo se centra en el desarrollo de frameworks que ayuden a estas aplicaciones. Según Jonson [29] un framework es el esqueleto de una aplicación, que puede ser personalizado por el desarrollador de una aplicación que lo utilice. Es un diseño reusable de todo o parte de un sistema. El principal cometido de un framework es identificar el factor común de un grupo y tipo concreto de aplicaciones (en el caso de este trabajo aplicaciones Context-Aware) y proporcionar un diseño reusable del mismo, de forma que sea eficiente y fácil de utilizar.

Pree [30] considera que un framework es un conjunto de bloques que están listos para usarse por aplicaciones y constituyen un sistema casi terminado, pendiente tan solo de la concretización por parte de los autores del sistema al dominio concreto de la aplicación. Producir aplicaciones específicas quiere decir bajo esta visión, ajustar los bloques hacia el dominio concreto.

Un framework se compone por tanto de dos tipos diferentes de bloques. Por una parte los bloques estáticos, que componen el sistema por dentro y conocen sus peculiaridades, conforman el problema general que es el mismo en todos los dominios de aplicación. Por otra parte están los bloques adaptables, que son personalizables a los dominios concretos y la forma en que un framework puede adaptarse a las distintas necesidades de cada aplicación.

La dificultad estriba en definir con precisión los componentes adaptables de un framework [31] ya que una falta de flexibilidad hará que numerosas aplicaciones no puedan utilizarlo. Por otra parte una flexibilidad excesiva conlleva mucha dificultad de aprendizaje y poca resolución de problemas, ya que muchos de los mismos se pasarían a los autores de aplicaciones.

Desde el punto de vista de la orientación a objetos, los componentes estáticos son las clases que vienen escritas y cuya funcionalidad se usa, y los componentes adaptables son las clases abstractas que hay que derivar para personalizarlas al dominio.

Un framework bien diseñado sólo puede salir desde la experiencia con numerosos dominios distintos [30], ya que para dar solución al problema de un conjunto de aplicaciones en sus dominios, se ha de analizar primero cuáles son las principales dificultades y trabajo común que realizan por separado.

La diferencia principal entre un framework y un componente [29] es que el último provee de reutilización de código, mientras que el primero provee principalmente de reutilización de diseño y puede venir con componentes que reutilicen el código. Otros componentes pueden también utilizar el framework y aprovechar sus ventajas, haciendo que el desarrollo de nuevos componentes sea más sencillo.

Un componente además ofrece y actúa de librería para las aplicaciones, solucionando problemas conocidos y reutilizando código. Aunque esta es una parte importante de un framework, sus límites van más allá, proporcionando reglas y patrones de diseño, ofreciendo arquitecturas e incluso pudiendo llegar a cambiar el rol de invocación. Una librería se invoca por la aplicación. En cambio, una aplicación usando un framework puede llegar a ser invocada por éste.

Un framework es por tanto, mucho más completo que una librería o API, ya que esta última es invocada para realizar tareas de las cuales la aplicación se puede abstraer. Sin embargo un framework además de todo ello puede proveer de toda una arquitectura de componentes, diseñados de una forma concreta. Además, este diseño se promueve dentro de la aplicación (en alguno casos forzándolo) de forma que ésta lo adopte para mejorar el rendimiento, ya que es un diseño que se ha probado eficiente tras analizar muchos problemas en aplicaciones similares. El framework ofrecerá también varios patrones de diseño que las aplicaciones deben adoptar para utilizarlo. De esta forma no suele ser posible utilizar un conjunto de bloques de forma aislada, y normalmente el uso de uno de ellos implica aprender el uso de varios relacionados. De hecho, el nivel de definición de los frameworks es tal que muchas veces van ligados a un lenguaje de programación concreto, ya que se promueven elementos que son solo posibles bajo ciertos lenguajes de programación. La implementación del framework para su uso suele en muchas ocasiones estar ligada a un único lenguaje de programación [29].

Los frameworks son más que código: son ideas que se promueven para llegar a una solución eficiente del problema, van más allá de la propuesta de un diseño y son más generales que componentes o librerías. Por ello utilizan numerosos patrones de software [32]. Como se ha dicho anteriormente no solamente los utilizan, sino que son la forma de contacto entre la aplicación y el framework en sí, haciendo que la aplicación se adhiera al uso de los mismos patrones para poder utilizar el framework.

Los problemas más comunes que se dan en los frameworks son primero su dificultad de desarrollo, mucho más alta que la de una aplicación ya que implica tener en cuenta muchos tipos de aplicaciones, muchos tipos de problemas y un equilibrio entre bloques adaptables y bloques rígidos.

Por otra parte, aunque implica abstracción a las aplicaciones de problemas concretos, requiere de un tiempo de aprendizaje que puede llegar a ser muy alto.

Por otra parte, los frameworks comparten con las librerías el problema de requerir un lenguaje concreto. Normalmente una aplicación está escrita en el mismo lenguaje del framework en el que se basa. Esto último puede abarcarse traduciendo frameworks a diferentes lenguajes de programación, o a su vez con frameworks como CORBA [33] o COM [34], ideados para solventar el problema de convivencia entre diferentes lenguajes.

Otro de los grandes problemas que se dan en los frameworks es la necesidad que se crea debido a la dependencia que la aplicación llega a tener con respecto al propio framework. Basar mucha de la funcionalidad en él tiene la gran ventaja de la abstracción, pero la gran desventaja de la dependencia. Los problemas que surgen al manejar el framework se han de solventar con los expertos del mismo. También se requiere un estudio detallado de la documentación que provee, hasta el punto de que mucha autores, vaticinando este problema, descartan un framework que no venga provisto de extensa documentación. El grado de uso del framework por otras aplicaciones también marcará hasta qué punto hay información suficiente de otras experiencias provenientes de diferentes aplicaciones.

Este grado de dependencia también viene dado por el soporte del framework. Una aplicación depende de los creadores del framework en el momento en que quiere aumentar las capacidades del mismo. Cuando se detectan elementos demasiado rígidos dentro de la arquitectura, o surgen nuevas necesidades que se deben cubrir, se ha de esperar a que un equipo externo provea de otra versión más completa. Debido a esto, el abandono del soporte al framework por parte de los autores o equipo de desarrollo ocasiona grandes desventajas, posiblemente dando lugar a la utilización de otros frameworks cuya migración puede resultar traumática.

En siguientes apartados se describen algunos frameworks centrados en aplicaciones Context-Aware.

3. FRAMEWORKS Y APLICACIONES CONTEXT-AWARE

Desde la creación de la visión de Computación Ubicua muchos han sido los intentos tanto de definir aplicaciones context-aware, como de organizarlas y por supuesto; crearlas. Tras los primeros intentos pronto surge la primera cuestión a abarcar: la necesidad de marcos de aplicaciones (frameworks) para poder avanzar en el desarrollo de este tipo de sistemas. Es necesaria, como enuncia el manifiesto del Reto de la Computación Ubicua [8], la creación de modelos de aplicaciones context-aware y derivar por tanto en frameworks que ayuden a su construcción.

Tras detectar esta necesidad, muchos autores han pasado a enunciar su modelo, su propuesta de arquitectura y framework context-aware antes de desarrollar las propias aplicaciones. A continuación se enumeran las más relevantes y se analiza su arquitectura.

3.1. STICK-E DOCUMENTS

Uno de los primeros pasos que se dieron hacia la creación de frameworks para aplicaciones Context-Aware fueron los Stick-e Documents de Brown [35]. Se trata de una metáfora creada para aplicaciones Context-Aware en forma de notas que muestran mensajes dadas unas ciertas condiciones de localización de usuarios. En los inicios de estos sistemas, el objetivo de la mayoría de aplicaciones era presentar información cuando se dieran ciertas condiciones relacionadas con la posición de usuarios u algún otro tipo de contexto dentro de un entorno.

Cada Stick-e Document consta del propio contenido o mensaje, y del contexto relacionado. En un ejemplo, cuando un usuario lleva consigo una PDA que usa algún medio de localización, se consigue tener una localización constante. Así, el propio usuario utilizando su PDA puede dejar un mensaje en la localización donde se encuentra en ese momento o en cualquier otro punto. Cuando pase una vez más por ese emplazamiento el Stick-e note es mostrado con el mensaje. Es una forma electrónica de dejar Post-its a lo largo de determinados lugares.

Además, estos mensajes pueden dispararse no sólo bajo condiciones de localización, sino cuando se cumpla alguna otra condición de contexto. Las condiciones posibles según los Stick-e Documents [35] pueden basarse en:

- Localización, que puede ser el emplazamiento de una nota en un lugar.
- Proximidad a otros objetos o personas. Cuando se detecte la presencia próxima de un elemento señalado, puede dispararse el Stick-e Document.
- Estados críticos. En el momento en que ciertos sensores detecten un valor que se sale del rango aceptado, puede mostrarse un mensaje.
- Estados del computador o dispositivo. Condiciones como entrar en una carpeta o abrir cierto archivo pueden ser condiciones para mostrar mensajes.
- Compañeros imaginarios. Brown los llama Spirits [35] y representan objetos o gente que no existe realmente o que no lleva ningún dispositivo de localización que el sistema pueda percibir. Al no poder ser detectados, sería el usuario el que manualmente indicase que está próximo a un Spirit. Los mensajes pueden dejarse ligados a un Spirit para que aparezcan todos juntos al indicar que el compañero imaginario está próximo.
- Tiempo. Los mensajes pueden mostrarse a una determinada hora. Este es el tipo de mensaje más comúnmente extendido.

Estas condiciones incluso pueden combinarse para crear condiciones más completas. Por ejemplo podrían combinarse condiciones de proximidad a una persona y localización.

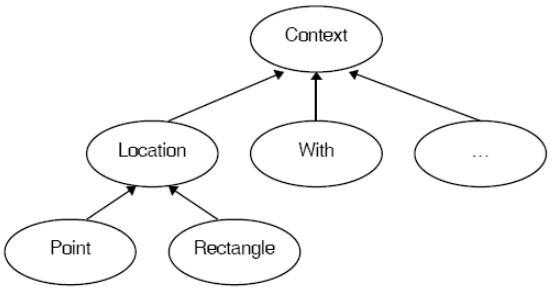

Figura 11. Modelo de condiciones relacionadas con el contexto en los Stick-e Documents [36]

Brown hace referencia a un Stick-e Document como un conjunto de Stick-e Notes [36] y define una Note como mensaje con su contexto y condiciones. Un Stick-e Document sería entonces un conjunto de Notes. De esa forma, todo el conjunto de Stick-e notes que se dejan en un Spirit, compone un Stick-e Document.

Brown también ha dado una representación en SGML [37] para una Stick-e Note. Como puede verse, las condiciones pueden marcarse como “required” para decir que son obligatorias. El contexto puede ser tanto un menaje de texto, como una página HTML o un programa que ha de ser ejecutado:

```
<note some attributes>
<required>
<with> J.D. Bovey <or> X. Chen
<at> co-ordinates of location
<content>
This is the content.
</note>
```

Código 1. Descripción general de una Stick-e Note [35]

Cuando en una nota se refiere a contexto, en realidad se hace referencia a las condiciones de activación de la Stick-e Note.

Este framework ayuda a realizar aplicaciones Context-Aware, principalmente definiendo una metáfora de triggers que se accionan bajo condiciones. Sin embargo no provee de componentes explícitos para solucionar problemas como la heterogeneidad del contexto, la necesidad de abstracción del medio físico de los sensores, etc. Las reglas que se especifican como condiciones representan los primeros pasos de una reacción hacia el contexto, aunque quedan insuficientes para aplicaciones con una necesidad más compleja y contexto de más alto nivel.

Este marco de aplicaciones tampoco proporciona ningún componente que se encargue de la creación de contexto de alto nivel ni el razonamiento sobre él. Además, el ámbito del contexto es muy limitado.

Todos los inconvenientes relatados anteriormente se deben principalmente a que se trata de una herramienta que apareció en los orígenes de las aplicaciones Context-Aware, cuando las definiciones de contexto y Context-Aware tenían unos límites muy cerrados. A pesar de estas limitaciones, los Stick-e Documents consiguen hacer una generalización eficiente en un framework y lo más importante: el propio concepto de Post-it electrónico aún hoy sigue siendo referencia para aplicaciones modernas.

3.2. CONTEXT TOOLKIT

Este framework ha sido creado por Dey y Abowd [38] en la Universidad de Berkeley. Su objetivo es ayudar a la ardua tarea de la captación de contexto. El contexto tiene extensiones muy vastas y una naturaleza muy heterogénea. Por tanto, una de las principales problemáticas a la hora de realizar estos sistemas es la tarea de captación del contexto de una forma homogénea dadas sus diversas naturalezas. Esta tarea de captación ha de ser no solo homogénea, sino reusable por otras aplicaciones.

La primera aproximación al pensar en contexto es la de tratarlo como una entrada más de datos, con lo que su tratamiento podría ser parecido a la entrada de datos a través de interfaces de usuario. Este planteamiento es erróneo, por varias cuestiones que Dey y Abowd [39] enumeran:

- La información de interfaz de usuario viene de una misma máquina, la información de contexto viene de varias entidades distintas de forma distribuida. Se hace necesario un mecanismo que abstraiga de la naturaleza distribuida del contexto.

- El contexto habitualmente no está en el mismo formato en que una aplicación lo necesita. El contexto de localización puede dar coordenadas GPS mientras que la aplicación que lo pide necesita una dirección postal. Se hacen necesarios mecanismos de conversión de formatos.

- Los elementos de interfaz son homogéneos, los sensores que proveen de información de contexto son de naturaleza muy homogénea y no convencional.

- Los elementos de interfaz de usuario dependen de la aplicación que los ejecuta, tienen el mismo ámbito y duran lo que dura la ejecución de la aplicación. Los elementos que proveen información de contexto no pertenecen a ninguna aplicación en concreto y no deben hacerlo, ya que el contexto es independiente de una simple instancia de aplicación. Puede incluso ser utilizado por varios sistemas context-aware distintos.

- El contexto es dinámico, cambia y debe ser chequeado de la forma adecuada para informarse de los cambios. La información de interfaz de usuario viene siempre dada desde el usuario hacia la aplicación.

Basándose en todas estas particularidades se ha creado el framework Context Toolkit. De forma general, los servicios que el framework provee son:

- Encapsulamiento de sensores

- Acceso a datos de sensores a través de una API de red

- Abstracción de los datos del contexto y su formato a través de intérpretes

- Compartición de la información de contexto a través de una infraestructura distribuida

- Almacenamiento de información de contexto den Base de Datos, incluyendo un histórico

- Control de acceso básico por razones de privacidad y seguridad

Con el objetivo de ofreces estos servicios, Context Framework dispone de la siguiente arquitectura:

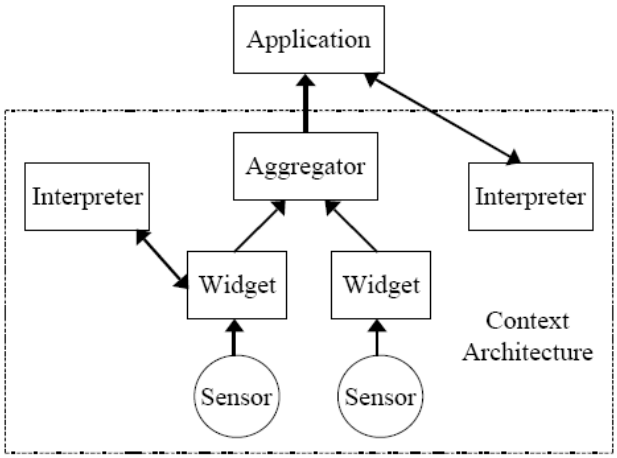

Figura 12. Arquitectura de Context Toolkit [39]

- **Widgets**: Estos elementos reciben el mismo nombre que los elementos gráficos en forma de pequeños programas que proveen de información. En cierto sentido el concepto es el mismo. Un Widget de contexto es un componente software que encapsula información acerca de un único elemento de contexto. Además, provee de una interfaz que abstrae de la naturaleza inherente del proveedor de información. Soportan mecanismos de polling y de suscripción.

- **Agregadores**: Los agregadores pueden ser vistos como meta-widgets, que pueden agregar y combinar información de varios widgets para ofrecer una única pieza de información. A su vez, estos componentes abstraen a la aplicación de la forma en a que diferentes informaciones de contexto deben ser combinadas para obtener información de un cierto mayor nivel.

- **Intérpretes**: Los intérpretes son los encargados de recoger cierta información en conjunto e interpretarla, para crear contexto de más alto nivel. Estos intérpretes pueden recibir la información de varios widgets o agregadores y combinarla para crear eventos de mayor abstracción del tipo “se ha iniciado una reunión”.

Estos elementos están dispuestos de forma distribuida, de forma que no pertenecen al ciclo de vida de una aplicación, sino que pueden estar en cualquier otra máquina. De todos estos detalles se abstrae la aplicación que usa el framework, que como en cualquier aplicación de componentes distribuidos, instancia un componente sin saber si realmente en la misma máquina o no. Estos elementos están además basados en estándares de comunicación (XML [40] sobre HTTP [41]) y de ello deriva que sean interoperables, para ser usados desde máquinas con distintas arquitecturas. Todas estas características palian el problema enunciado al principio apuntando que el contexto no está en la misma máquina.

Asimismo, se solventan problemas como que el contexto no está frecuentemente por naturaleza en el formato en que la aplicación lo necesita y que la naturaleza de su información es heterogénea. Los widgets abstraen de la naturaleza real de los proveedores de información, y ellos junto con los agregadores e intérpretes adecuan el formato a las necesidades de la aplicación. La posibilidad de suscripción a notificaciones solventa el problema de de contexto cambiante.

Esta arquitectura distribuida es ejecutada en una instancia totalmente aparte de cualquier aplicación en ejecución, lo que la separa del ciclo de vida de cualquiera de ellas. La información se recoge y almacena aunque no haya aplicaciones ejecutándose en un instante concreto, lo que posibilita el acceso a un historial de información.Posteriores versiones incluyen mecanismos de búsqueda de widgets, lo que abarca un problema muy importante en la captación de contexto: el descubrimiento de nuevos proveedores de contexto adecuados.

Este framework es fue uno de los primeros en aparecer y por tanto adolece de la inexperiencia en aplicaciones context-aware. Se centra casi exclusivamente en el ámbito de captación de contexto, cuando este es sólo uno de los problemas que tienen estos sistemas. Es una buena aproximación, pero se antoja insuficiente según de van detectando nuevas necesidades. Este framework abarca también el concepto de obtención de contexto de alto nivel, pero según las necesidades de razonamiento de estas aplicaciones fueron en aumento, el concepto de agregación e interpretación de contexto no llegó a solventar el problema de una forma eficaz.

Sin embargo, sí que supuso un avance tanto en el desarrollo de frameworks como en planteamiento de soluciones a los problemas de adquisición de contexto. Cuestiones como los componentes distribuidos, el sistema de notificaciones y la abstracción de la naturaleza heterogénea del contexto han probado ser eficaces formas de abarcar los problemas.

3.3. AURA

El proyecto Aura [42] es una arquitectura diseñada para la movilidad del usuario cuando este se encuentra ejecutando aplicaciones en su dispositivo. Cuando un usuario se mueve a otro lugar, con diferentes recursos, Aura le asiste y se adecua al nuevo entorno. El objetivo es mover sus tareas de un entorno a otro sin que se vean afectadas. Para ello, este framework pretende hacer uso de los recursos del entorno al máximo posible sin que las tareas que el usuario realiza se vean afectadas en su rendimiento, avisándole si una correcta ejecución no es posible con los recursos disponibles. De hecho, además de un framework, Aura es una infraestructura que se detalla a continuación.

Aura no aplica una definición primitiva de contexto y se refiere a entorno como espacio físico en el que el usuario se encuentra en ese momento; no como sinónimo del contexto de la aplicación. Un cambio de entorno en Aura significa un traslado físico a otro lugar, mientras un cambio de contexto puede darse dentro de un mismo entorno.

La metáfora de Aura, el proyecto la lleva a la definición de Aura Personal de usuario, que le rodea y en todo momento sabe qué recursos tiene disponibles para poder ejecutar sus tareas y aplicaciones correctamente. Con tareas, los autores del proyecto se refieren a conceptos simples como escribir un documento, preparar una presentación, realizar una compra o navegar por Internet.

La arquitectura consta de unos componentes principales:

- Gestor de tareas, llamado *Prism*, que representa el mismo concepto de Aura Personal
- Observador del contexto, que monitoriza el contexto, sus recursos y sus cambios de estado. Da su información al resto de elementos.
- Gestor del entorno, que actúa de enlace con el entorno
- Suministradores, que representan los servicios básicos de los que se componen las tareas. Ejemplos son edición de video, escritura de texto
- Conectores, que gestionan las conexiones los diferentes componentes y sirven de abstracción del medio de conexión.

Cuantos más suministradores tenga un entorno, más rico y potente será. En cada entorno hay una serie de estos componentes ejecutándose, de forma que al cambiar de entorno haya nuevos suministradores e incluso sea posible que haya un componente observador del contexto diferente. Las conexiones de Prism con suministradores pueden variar con la llegada o marcha de suministradores en el entorno, o bien con el cambio a otro entorno diferente.

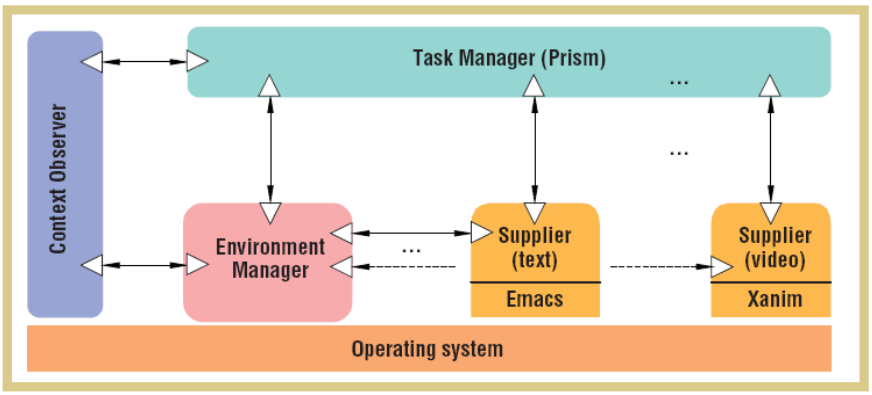

Figura 13. Arquitectura de Aura [42]

El **Gestor de Tareas** (Prism) entiende por tarea lo mismo que un usuario entiende por ella: todo un conjunto cohesivo de aplicaciones y archivos. De forma que para Aura manejar una tarea puede implicar manejar varios programas a la vez. Prism representa al Aura Personal de usuario y es el elemento central de la arquitectura. El gestor tiene en cuenta los siguientes sucesos:

- Un cambio a un **entorno diferente**: Prism coordina la migración de toda la información de la tarea al nuevo entorno y negocia la conexión con el nuevo gestor de entorno.
- El **contexto cambia**: si varían las condiciones relacionadas con la tarea en curso, Prism analiza los nuevos valores de Calidad del Servicio y si son incompatibles (bien porque los nuevos componentes no soportan ciertas características o porque simplemente componentes desaparecen) se negocia con el Gestor de Entorno una nueva configuración para dar soporte a la tarea.

- La propia **tarea** cambia: Cuando Prism recibe notificación de que se desea cambiar de tarea (el usuario lo desea, llega un evento en el calendario u ocurre algún cambio en el contexto que fuerza un cambio de tarea) gestiona el almacenamiento del estado de la tarea actual e instancia la nueva.

Los **suministradores** encapsulan todos los servicios de los que las tareas se componen. Puede haber varios suministradores en el mismo entorno con diferentes capacidades y condiciones. Estos suministradores han de ser capaces de representar el estado del servicio para obtener y dar información sobre él. Los autores han elegido lenguajes de marcado para la descripción, puesto que es una forma fácil de analizar la información que se conoce e ignorarla información que no se sabe manejar. Por ejemplo, una capacidad llamada "soporte de corrección ortográfica" puede ser parte de la descripción. Sin embargo, el propio suministrador no la conoce y no sabe cómo manejarla.

Una forma de obtener fácilmente suministradores es crear wrappers a aplicaciones existentes. Envolver a aplicaciones como Microsoft Word o Emacs, pueden crear suministradores de escritura de documentos.

El **Observador de Contexto** es el elemento que abstrae de la captación de toda la información contextual del entorno. Será más o menos sofisticado dependiendo de la cantidad de sensores que haya desplegados en cada entorno. Se encarga asimismo de la notificación de cambios de contexto tanto a Prism como al Gestor de Entorno. Podrá haber un Observador de Contexto en cada entrono diferente.

El **Gestor de Entorno** abstrae de la comunicación entre entorno y aplicación. Sabe qué componentes hay disponibles en cada momento y qué servicios pueden ser cubiertos. Cuando los suministradores se registran en un entorno o desaparecen de él, se actualiza el registro en el Gestor de Entorno. Actúa de pasarela para proveer del suministrador adecuado a las tareas.

Los **conectores** gestionan todos los aspectos relacionados con la comunicación y abstraen de ello a los componentes. En el cambio de contexto o de entorno, es necesaria una reconfiguración, añadir o quitar suministradores y quizá incluso conectarse con nuevos Observadores de Contexto y Gestores de Entorno. Estas conexiones implican diferentes medios físicos de transmisión de datos, diferentes tecnologías de comunicación distribuida e invocación remota. Los conectores existen para ocultar todos esos detalles a los componentes. Existen cuatro tipos diferentes de conectores: los que conectan Prism a los Suministradores, los que lo conectan al Gestor de Entorno, los que lo conectan al Observador de Contexto y los que conectan el Gestor de Entorno con el Observador de Contexto. Cada uno de estos tipos tiene su propio protocolo y puede ser implementado de varias formas diferentes dependiendo de los medios disponibles y la propia naturaleza de los componentes.

Para entender más claramente el funcionamiento de Aura, Garlan et al. [43] definen un caso de uso en el que existen 2 entornos diferentes: Una oficina y una casa. Cuando un usuario se mueve de su casa a la oficina, el Observador de Contexto anuncia ese hecho y Prism salva el estado de las tareas en ejecución, para después indicar al Gestor de Entorno que las detenga.

La información es entonces guardada en un sistema distribuido de archivos. Cuando el usuario entra en su oficina el Observador de Contexto de la oficina registra el evento y se comunica con Prism en la oficina. Éste vuelve a instanciar las tareas negociando con los Suministradores de la oficina. En el proceso de negociación dichos suministradores Informan de sus servicios a través de un lenguaje descriptivo de marcado [42].

```
<auraTask id="demo">
  <service type="editText">
    <duration unit="minutes" bad="10" good="30"/>
    <settings pane_height="360" pane_width="200">
      <spelling enabled="yes" ignoreAllCaps="yes"/>
      <editing overstrike="no" replaceSelection="yes"/>
    </settings>
  </service>
  <material origin="myTextFile" format="txt">
    <state cursor="104" scroll="28" zoom="100"/>
  </material>
</auraTask>
```

Código 2. Descripción de servicio en Aura [42]

En la reconfiguración se intenta mantener el mismo o superior nivel de servicio y utilizar al máximo posible los recursos de que se dispone.

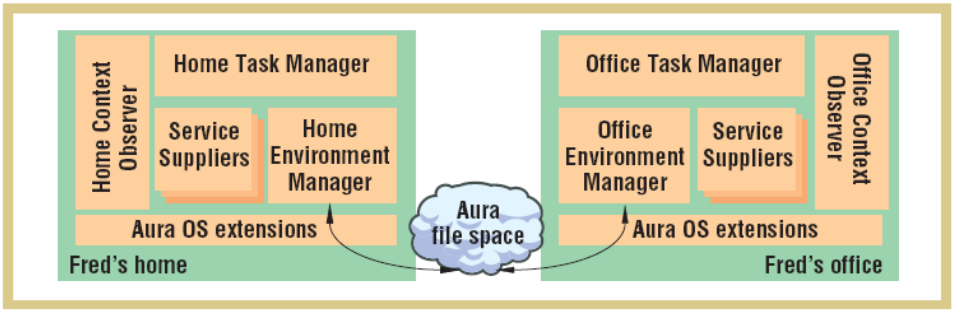

Figura 14. Diferentes entornos en Aura [43]

Este framework contempla muchos aspectos que ofrecen problemáticas en aplicaciones Context-Aware. Los componentes que crea están dedicados a un campo concreto que es la movilidad de usuario. No es un framework general pero es completo para el campo en el que está focalizado. Contempla aspectos como la abstracción de la captación del contexto y la notificación de cambios. Aunque los separa en diferentes componentes (Observador de Contexto y Gestor de Entorno). Además de la captación del contexto, este framework también ofrece posibilidades para reaccionar y adaptarse a él. Esta reacción y adaptación al contexto, como el propio framework, está muy fuertemente ligada al campo de migración y movilidad de tareas. Se preocupa también por aspectos de definición de lenguajes estándar para el contexto, en cuanto a que define un lenguaje para describir el estado de los servicios en los Suministradores.

Otro concepto muy importante que abarca es la abstracción del medio de transmisión de datos y mecanismos de comunicación, representada en el concepto de Conectores. Dada la naturaleza distribuida de aplicaciones Context-Aware, se hacen necesarias capas que abstraigan de todos los mecanismos de comunicación y medios de transmisión.

Para migrar tareas de un entorno a otro, la información necesaria (el contexto de la tarea) es guardada en un sistema de archivos distribuido. Esto significa que Aura es uno de los frameworks que comienza a ver la necesidad de un espacio para almacenar el contexto de forma distribuida.

Sin embargo, el framework no ofrece ninguna facilidad para el razonamiento sobre el contexto o algún medio similar como agregación o interpretación. A pesar de tener un ámbito concreto, sigue siendo necesario que sistemas obtengan contexto y lo interpreten consiguiendo más alto nivel. Su Observador de Contexto capta valores y envía notificaciones, pero no provee de componentes que ayuden al razonamiento o agregación.

3.4. THE CONTEXT MANAGING FRAMEWORK

Este framework ideado por Korpiää et al. [44] se centra básicamente en la captación, modelado y razonamiento del contexto. Su arquitectura está dispuesta en forma jerárquica teniendo un componente central al que la aplicación accede. De esta forma se crean dos capas: una con el componente central y otra con los componentes secundarios que este componente utiliza.

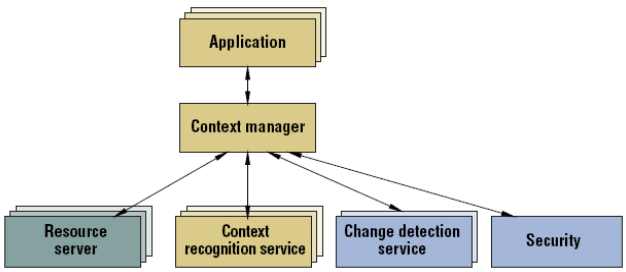

Figura 15. Arquitectura de Context Managing Framework [44]

Los componentes básicos son:

- Gestor de Contexto (Context Manager): Funciona como componente central y almacena el contexto obtenido para que las aplicaciones puedan acceder a él. Los clientes de este gestor hacen peticiones como si se tratase de una Base de Datos de contexto. También pueden suscribirse a cambios. El Gestor de Contexto maneja contexto tanto de bajo como de alto nivel de forma transparente para los clientes.

- Servidores de Recursos (Resource Servers): Se conectan a las fuentes de datos de contexto y recogen la información. Además, la preprocesan para eliminar inconsistencias o valores erróneos. También pueden hacer agregación o convertir los valores a categorías, haciendo así tanto una reducción de espacio en valores como una conversión al formato deseado. Por último, pueden añadir información semántica.

- Servicio de Reconocimiento de Contexto (Context Recognition Service): Recoge contexto de bajo nivel y devuelve contexto de alto nivel tras un proceso de computado.

El framework posibilita una forma de abstracción del marco físico del contexto y sus particularidades. Además de esta capa de abstracción situada en el Servidor de Recursos, después de obtener los datos se preprocesan para obtener el formato y corrección adecuados. Este framework introduce elementos avanzados de formateo de contexto como cuantización, extracción de categorías y etiquetado semántico.

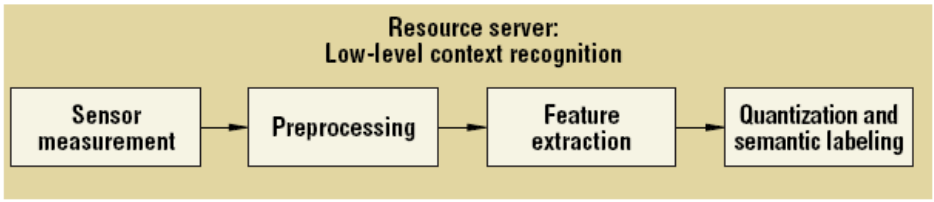

Figura 16. Proceso de la información del contexto de bajo nivel en el Servidor de Recursos [44]

Se hace asimismo diferenciación entre contexto de bajo nivel (en el Servidor de Recursos) y de alto nivel (en el Servicio de Reconocimiento de Contexto). El razonamiento se limita a reconocer contexto de alto nivel, no a modelar cuáles son las reacciones de la aplicación al contexto.

La arquitectura jerárquica es eficiente para entornos con poca capacidad de proceso donde una arquitectura totalmente distribuida puede llegar a ser demasiado costosa. Sin embargo, el Gestor de Contexto representa un cuello de botella y un punto potencial de fallo.

3.5. GAIA

El proyecto Gaia [45] es una infraestructura (además de un framework) que pretende servir de apoyo a aplicaciones Context-Aware. Gaia se define como un meta Sistema Operativo para espacios activos [46]. Con espacios activos se refieren a emplazamientos físicos que contienen objetos reales, dispositivos heterogéneos que pueden estar inter conectados y personas realizando actividades. Estos espacios activos están coordinados por un software que ayude a la realización de las actividades del usuario.

El proyecto Gaia tiene una arquitectura específica para este tipo de sistemas Context-Aware. Su infraestructura se divide en el propio núcleo y el framework que lo utiliza.

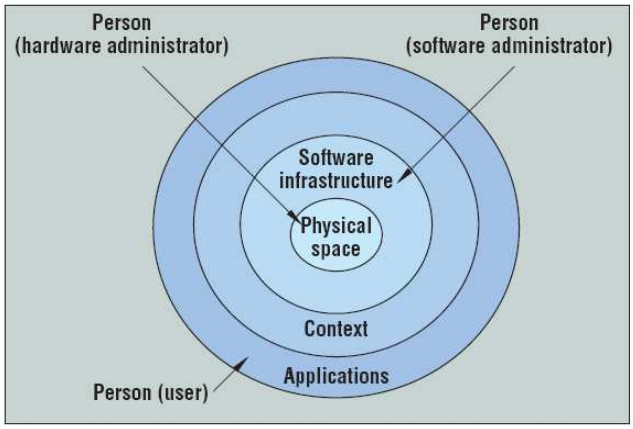

Figura 17. Un Espacio Activo en Gaia [46]

El núcleo de Gaia consta de los siguientes elementos:

- Gestor de Eventos
- Servicio de Contexto
- Servicio de Presencia
- Repositiorio
- Sistema de Archivos de Contexto

El **Gestor de Eventos** proporciona una estructura de eventos flexible para espacios donde los elementos son dinámicos y cambian constantemente (un componente comienza su ejecución, una persona entra en el recinto, una aplicación se mueve de un computador a otro). Este gestor de eventos distribuye los eventos en el Espacio Activo y lo hace de una forma desacoplada, utilizando suministradores de eventos, consumidores que se registran a ellos y canales para transportar la propagación de los eventos. Este gestor tiene un solo punto de entrada, con varias factorías de eventos que poder crear y a los que poder registrarse. Cada factoría crea el canal adecuado para propagar la información del evento. Gaia tiene una serie de canales por defecto, por los cuales propaga errores, nuevas aplicaciones o personas, etc. Aunque cada aplicación puede crear los canales que necesite. El desacoplamiento de los componentes del Gesto de Eventos proporciona fiabilidad, puesto que dado el fallo de un componente puede reemplazarse por otro sin perjudicar al sistema.

El **Servicio de Contexto** proporciona información del propio contexto del Espacio Activo. Actúa de punto de contacto entre el contexto y las aplicaciones del sistema, posibilitando a las mismas el poder informarse, reaccionar y adaptarse a él. Cambios en el contexto se propagan mediante la infraestructura de eventos. Se compone de proveedores de contexto que actúan de mediadores ente el servicio y los sensores, componentes que infieren contexto de alto nivel y un registro donde están registrados todos los proveedores. Este contexto se modela a través de lógica de primer nivel y álgebra de booleana, que además posibilita escribir reglas de descripción de contexto fácilmente.

La representación del contexto se hace en forma de predicado de cuatro elementos:

```
Context(<ContextType>, <Subject>, <Relater>, <Object>)
```

Código 3. Predicado de contexto en Gaia [46]

- El tipo de contexto (ContextType) se refiere al contexto que el predicado está describiendo
- El sujeto (Subject) es el objeto, persona o lugar a quien se refiere

- El objeto (Object) es el valor que adquiere ese contexto

- La relación (Relater) relaciona el sujeto con el objeto comúnmente mediante operadores relacionales (> = <), un verbo o una preposición.

Ejemplos de predicados son:

```
Context(temperature, room 3231, is, 98 F)
Context(printer status, srgalw1 printer queue, is, empty)
```

Código 4. Ejemplos de predicado de contexto en Gaia [46]

De la misma forma que se describe el contexto, se pueden describir las reglas necesarias para la inferencia. Los mismos predicados que sirven para constatar hechos, sirven también para enunciar reglas, que pueden combinarse utilizando operaciones lógicas como cuantificación, implicación, conjunción o disyunción:

```
Context(number of people, room 2401, >, 4)  AND Context(application, Powerpoint, is, running) =>
Context(social activity, room 2401, is, presentation)
```

Código 5. Ejemplo de regla en Gaia [46]

De esta forma, se crea un conjunto de reglas y una base de conocimiento que sirve para inferir contexto de más alto nivel a partir del contexto de bajo nivel recogido. Esto implica que el Gestor de Contexto lleva embebido un motor de reglas.

El **Servicio de Presencia** es una particularización especial del contexto ideada para registrar el estado de los dispositivos, personas y componentes software. Al centrarse esta infraestructura en ser un meta Sistema Operativo, da especial importancia a este subconjunto de contexto y arquitectónicamente lo separa del Gestor de Contexto. Gaia discierne entre cuatro tipos de entidades: aplicaciones, servicios, dispositivos y personas. Este servicio está dividido en dos sub-sistemas: la presencia física (dispositivos y personas) y la presencia digital (aplicaciones y servicios).

En el sub-sistema digital, las entidades envían periódicamente señales para informar que siguen activas (heartbeats). Si se dejan de recibir se asume que bien han acabado o han sufrido algún fallo grave. Esto sirve para informar Gaia en todo momento acerca de las aplicaciones software. El servicio físico utiliza otros medios para detectar entidades físicas, como sistemas de localización.

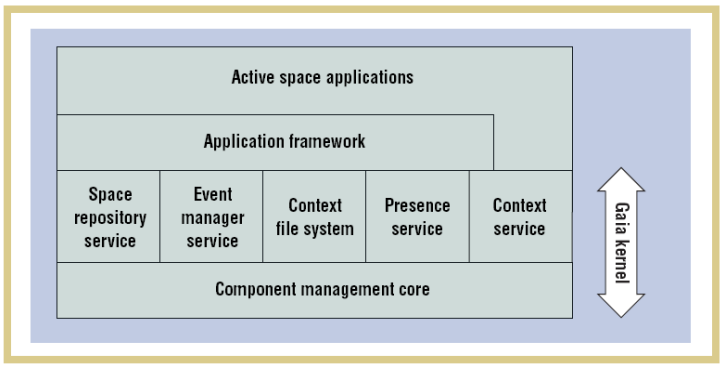

Figura 18. Arquitectura de Gaia [46]

El repositorio actúa de Base de Datos sobre lo elementos físicos y digitales que existen dentro del Espacio Activo. Es el lugar donde se guarda su información y de donde las aplicaciones los recogen y buscan. Cuando una aplicación comienza acude al repositorio para encontrar los recursos que necesita, como pantallas, altavoces y nodos de ejecución. Esto proporciona una forma muy desacoplada de definir aplicaciones, de forma que ellas puedan buscar recursos en Espacios Activos diferentes sin verse afectadas. Los recursos que se registran en el repositorio describen sus capacidades mediante un lenguaje XML.

El Sistema de Archivos de Contexto ofrece los recursos de almacenamiento a las aplicaciones y demás componentes de forma que incluye información de contexto, para que muchas tareas que se hacen manualmente puedan ser hechas de forma automática conociendo la información de contexto.

Ofrece buenas características de relación de sistema de archivos con el contexto. Concretamente es capaz de:

- Hacer accesible la información personal a las aplicaciones en el momento en que se entra en el Espacio Activo

- Organizar los datos para facilitar la búsqueda de material relevante tanto a aplicaciones como a usuarios. Por ejemplo, en lugar de navegar en un árbol de carpetas normal, este sistema de archivos puede navegar por un árbol de localizaciones físicas y encontrar, por ejemplo, documentos que se han creado en la “habitación 302”. De esta forma se navega por árboles jerárquicos de información de contexto, mediante jerarquías virtuales de archivos. De hecho, los valores de contexto pueden combinarse y presentar jerarquías de archivos donde el lugar sea “habitación 302” y la actividad en ese lugar sea “una reunión”.

- Ofrecer los datos en el formato más adecuado teniendo en cuenta las preferencias de usuario y las capacidades del dispositivo, mediante tipos de datos dinámicos. Una presentación en slides puede ser vista por el visor de presentaciones de un PC, o convertirse en un slideshow de imágenes cuando quiera verse por móvil.

Además de este núcleo, existe un API para aplicaciones que ofrece todas las posibilidades de la infraestructura a aquellos programas que se ejecuten en un Espacio Activo. El API consiste en:

- Un modelo de componentes distribuidos que encapsula las funcionalidades de Gaia y las ofrece a las aplicaciones

- Un mecanismo de mapeo que personaliza las aplicaciones para los Espacios Activos.

- Un conjunto de políticas que pueden definirse para configurar las preferencias de las aplicaciones, su forma de instanciación, su grado de movilidad etc.

Este framework describe soluciones eficientes para problemas de aplicaciones Context-Aware, aunque su foco es para aplicaciones que realmente comparten el meta Sistema Operativo. Su infraestructura modela el contexto, ofrece herramientas para razonar sobre él y soluciona a través de eventos y un repositorio el problema de descentralización del contexto. Quizá pudiera completarse con un módulo interno en el Gestor de Contexto que ofrezca agrupación o caracterización.

Su infraestructura es descentralizada y desacoplada, lo que la hace muy tolerante a fallos. Utiliza extensivamente modelos distribuidos de componentes. Su sistema de inferencia se basa en un motor de reglas de predicados lógicos de primer nivel, fáciles de representar, pero no ofrece mecanismos de representar incertidumbre ni lógica difusa.

3.6. HYDROGEN

Este framework creado por Hofer et al. [47] se basa en una arquitectura de tres capas que van desde la capa física de sensores hasta el uso de la información de contexto por las aplicaciones. La capa de más bajo nivel es llamada **Capa de Adaptación**, mientras que la capa que gestiona el contextote forma lógica es la **Capa de Gestión**. Las aplicaciones forman la tercera capa, llamada **Capa de Aplicación**.

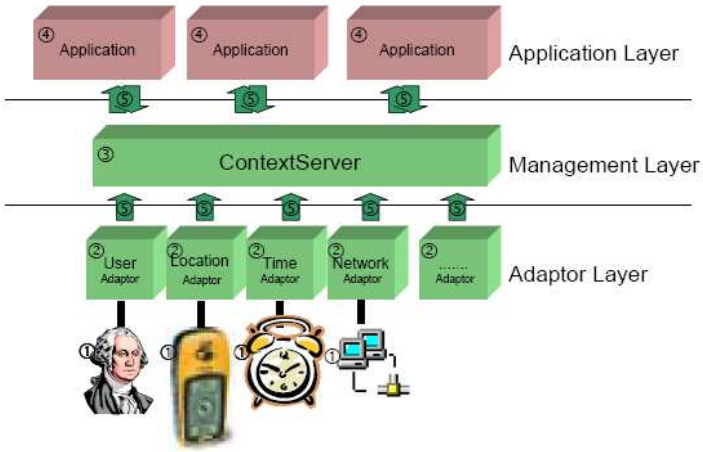

Figura 19. Arquitectura del framework Hydrogen de Hofer et al. [47]

Se trata de una arquitectura cuyo factor principal es la simplicidad y se encuentra centralizada en la Capa de Gestión. Estos componentes están ideados para situarse dentro de un dispositivo, en lugar de haber un grupo de componentes en un lugar al que todos los dispositivos puedan acceder, o disponer de un modelo distribuido. Cada capa contiene un tipo distinto de componente:

La Capa de Adaptación contiene los Adaptadores. Cada uno de ellos que recoge información de un sensor en concreto, está especializado y diseñado para él, con lo que conoce sus particularidades. Estos adaptadores pueden también tratar de alguna manera la información obtenida. Tras esto, envía la información a la Capa de Gestión.

La Capa de Gestión tiene un solo componente: el Servidor de Contexto (ContextServer), quien almacena toda la información e incluso puede compartirla con otros servidores situados en otros dispositivos, de forma que puedan acceder a información de contexto que ellos mismos no pueden obtener por carecer de sensores para ello. Esta distribución de información se hace en forma peer-to-peer. Debido a este hecho, el servidor distingue entre contexto local y contexto remoto. El servidor proporciona dos mecanismos para obtener la información: mecanismo de consulta síncrono y mecanismo de suscripción-notificación asíncrono.

La capa de Aplicación está formada por las aplicaciones que usan este Servidor de Contexto. Esta arquitectura está pensada para varias aplicaciones usando el framework a la vez, más que un modelo de componentes para una sola aplicación.

Toda la comunicación inter-capa se hace a través de lenguaje XML. Es posible obtener esta comunicación XML directamente, aunque el framework provee de un componente específico para ello llamado ContextClient (Cliente de Contexto). Él se encarga de la gestión del protocolo XML y la apertura y cierre de conexiones con el servidor. Este cliente puede utilizarse tanto para crear la aplicación como para crear nuevos Adaptadores de contexto.

El contexto que se modela en este framework se maneja en forma de objetos, que pertenecen a una jerarquía de clases. Esta jerarquía puede extenderse pero consta por defecto de los objetos de Tiempo, Dispositivo, Red, Usuario y Localización.

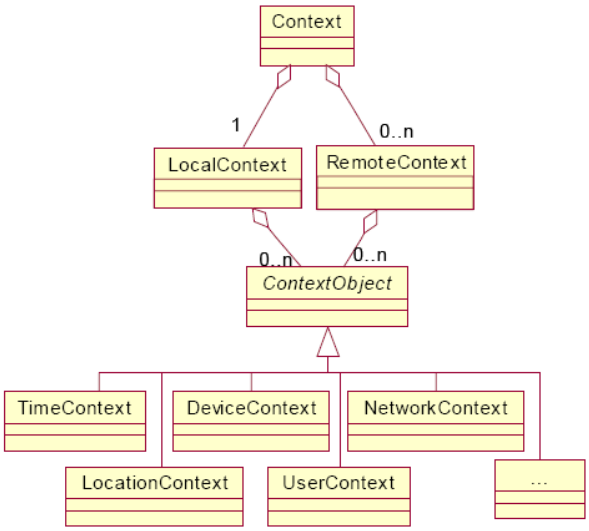

Figura 20. Jerarquía de objetos de contexto en Hydrogen [47]

Esta arquitectura es simple, robusta y tolerante a fallos de conexión. Si no hay conectividad no será posible la conexión con otros Servidores de Contexto, pero ya que habrá un servidor en cada dispositivo, las aplicaciones podrán seguir su funcionamiento, aunque con un conjunto más reducido de información. El hecho de ser centralizada la hace más adecuada para dispositivos con menores capacidades de cómputo, aunque un servidor central para muchas aplicaciones es un potencial punto de fallo.

Proporciona una buena abstracción del contexto, sin embargo no ofrece herramientas para un razonamiento o inferencia con el objetivo de obtener contexto de alto nivel. El modelado del contexto se basa en objetos, lo que tampoco es buena opción si se quiere incluir inferencia aunque sea con componentes externos. Elimina el concepto de historial de contexto para simplificar la arquitectura.

3.7. SOCAM

Esta arquitectura de Gu et al. [48] [49] provee de un middleware para aplicaciones Context-Aware dirigido especialmente al prototipado rápido general. Se compone de varios componentes dispuestos en una forma distribuida en lugar de jerarquizada o en capas.

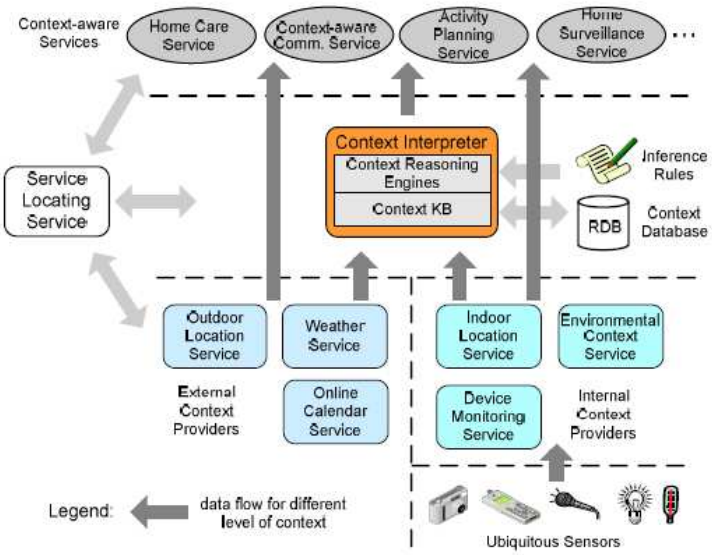

Figura 21. Arquitectura de SOCAM [49]

Los elementos que la componen se describen a continuación:

- Proveedores de Contexto (Context Providers): gestionan la captación de contexto a través de sensores y crean el contexto de bajo nivel. Se hace una distinción entre los proveedores internos, que utilizan los sensores dentro del entorno; y los externos, que utilizan fuentes externas para conseguir la información (tiempo, estado del tráfico, etc.). La información de contexto es representada con una ontología utilizando OWL [50].

- Interpretador de Contexto (Context Interpreter): utiliza el contexto de bajo nivel para inferir contexto de alto nivel. También actúa de proveedor en tanto en cuanto ofrece nueva información de contexto. Está compuesto de un razonador de contexto y una base de conocimiento. El razonador se encarga de la inferencia de nuevo contexto y el mantenimiento del buen estado de la base de conocimiento. Está diseñado para que nuevos razonadores puedan ser añadidos al mismo interpretador. La base de conocimiento es el conjunto de ontologías de contexto almacenadas en Base de Datos, además de un grupo de APIs para añadir y manipular nuevas.

- Servicio de Localización de Servicios (Service Locating Service): Este servicio actúa de índice y directorio de los Proveedores de Contexto existentes en el sistema. Cada uno de ellos se registra en él y da una descripción en OWL del tipo de contexto que ofrece. Cuando una aplicación envía una petición de búsqueda de contexto, manda una petición de búsqueda al servicio, quien aplicará correspondencia semántica [51] para averiguar qué proveedores ofrecen dicho contexto buscado.

- Servicios Context-Aware: Son lo servicios o aplicaciones Context-Aware que usan la arquitectura, buscan proveedores de contexto y los utilizan. Pueden asimismo añadir nuevas reglas de contexto de alto nivel al razonador para obtener la información que necesitan.

Toda la arquitectura está dispuesta en componentes de forma que pueden ser distribuidos por diferentes redes. La diseminación del contexto se puede hacer tanto en modo suscripción/notificación, como en modo consulta.

SOCAM ofrece una buena técnica de razonamiento, con posibilidad de añadir nuevos razonadores y un buen modelado del contexto a través de ontologías. Sin embargo añadir incertidumbre no es tan trivial: es posible añadir un nuevo razonador, permitiera incertidumbre, pero sería necesario modificar la ontología de contexto en OWL para permitirla.

3.8. CORTEX

Cortex fue creado por Biegel y Cahill [52] y básicamente se centra en la creación de lo que ambos llaman Sentient Object. Estos objetos reciben información de sensores, la almacenan, procesan e infieren sobre ella, para generar un output o bien actuar sobre actuadotes.

Para realizar estas tareas, los Sentient Objects están divididos en varios sub-componentes:

- Módulo de captura de sensores: Recoge información e los sensores y crea el contexto de bajo nivel.

- Jerarquía de contexto: Los daos del contexto son organizados dentro de una jerarquía para reducir la complejidad de la información

- Motor de inferencia: Infiere sobre el contexto y obtiene nueva información. Es capaz de tratar incertidumbre.

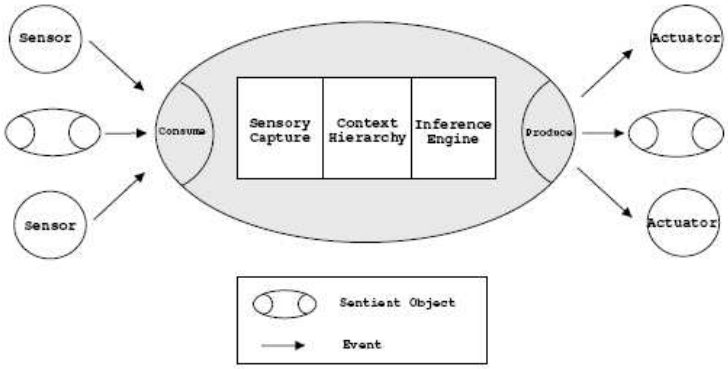

Figura 22. Modelo de Sentient Object en CORTEX [52]

A través de interfaces, estos objetos se comunican con sensores y consumen sus eventos tras tratar la información producen eventos para los actuadotes. Estos objetos pueden ser a su vez consumidores y productores de eventos para otos Sentient Objects, pudiendo así formar cadenas.

3.9. COBRA

Este framework, creado por Chen et al. [53] pretende dar soporte a aplicaciones Context-Aware en lo que los autores llaman Intelligent Spaces o Espacios Inteligentes [54]. Se centra en un componente llamado Context Broker, que será el punto central de acceso al contexto del Espacio Inteligente. Que mantiene centralizado y actualizado el modelo de contexto de los diferentes agentes (aplicaciones y servicios) dispersos en el espacio.

- Este componente obtiene el contexto de los agentes y lo centraliza, manteniéndolo estable y limpio de inconsistencias gracias a su motor de inferencia.
- También se encarga de dispersar la información del contexto a los agentes que lo solicitan, a través de una ontología que lo modela.
- Protege la seguridad y privacidad de los agentes a través de políticas de acceso.

Estos brokers pueden encargarse de una parte del Espacio Inteligente (por ejemplo una habitación dentro de un edificio) y después conectase entre ellos y organizarse de forma jerárquica para mantener el contexto del espacio completo [55].

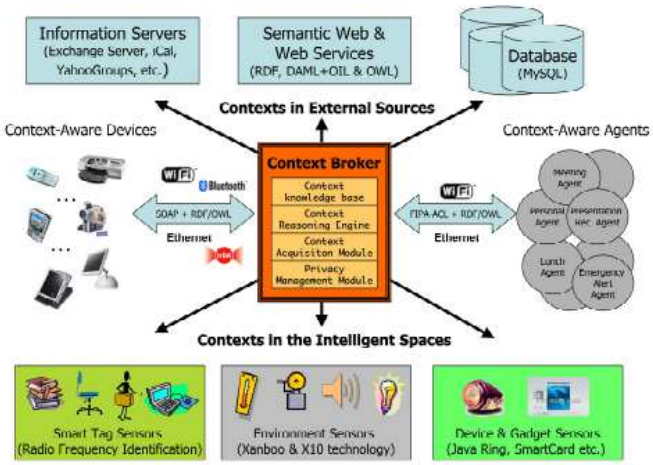

Figura 23. Context Broker en COBRA [55]

El broker consta de los principales componentes:

- Módulo de Adquisición de contexto: Se encarga de abstraer de la capa física de sensores y sus particularidades.

- Motor de Razonamiento: Razona sobre el contexto de bajo nivel para obtener contexto de nivel más alto, también es usado para mantener la consistencia de todo el contexto que viene de muy diferentes fuentes (agentes). Este motor trabaja sobre una representación de contexto basado en ontologías. Se utiliza una ontología (OWL) que describe gente, agentes, lugares y presentación de eventos.

- Base de Conocimiento de Contexto: Base de Conocimiento sobre el que se apoya el Motor de Conocimiento. Esta Base de Conocimiento también se apoya en una Base de Datos (MySQL).

- Módulo de Gestión de Privacidad: Contiene las políticas de acceso por contexto y agente

Esta arquitectura centralizada posibilita que los agentes no requieran dedicar potencia de proceso para obtener el contexto. Sólo han de conectarse al broker que estará situado en un potente servidor para poder realizar un eficiente proceso de datos. Es una aproximación para incluir contexto en dispositivos con limitado poder de proceso. Mantener el contexto centralizado y común para todos los dispositivos también posibilita más información disponible para todos, aunque este framework adolece de las desventajas de todo sistema centralizado: un solo componente es un potencial punto de fallo y además un fallo de conexión deja al dispositivo que lo sufra sin ninguna información de contexto.

3.10. CASS

El framework CASS, de Fahy y Clarke [56] está formado por una arquitectura cliente-servidor, en el que las aplicaciones Context-Aware se conectan a un servidor que les proporciona la información de contexto. Además, hay nodos que disponen de sensores, y que pueden ser tanto móviles como fijos y que proveen de información al servidor. Por su parte, el componente central dispone de amplios recursos para hacer razonamiento sobre el contexto y guardar un gran historial de valores.

Esta arquitectura, por ser centralizada, se apoya en las comunicaciones de red, siendo estas un punto de fallo del sistema. Para solventar esto, el framework provee de mecanismos de caché en los clientes.

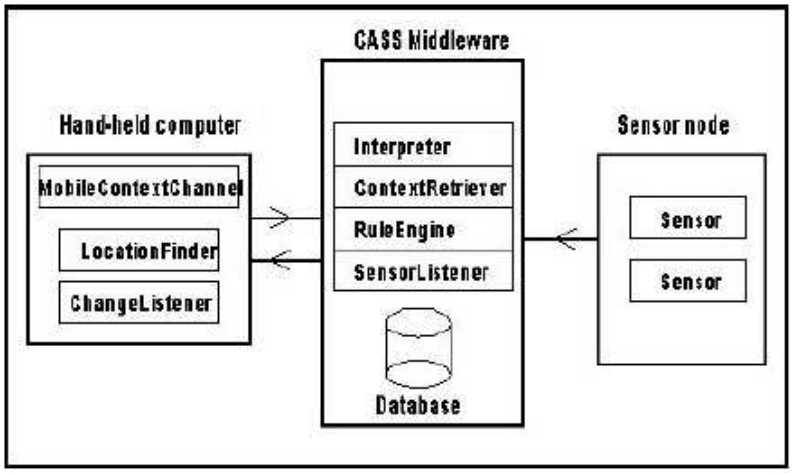

Figura 24. Arquitectura de CASS [56]

En esta arquitectura los componentes son:

- ContextListener es un componente que permite a los dispositivos escuchar notificaciones de cambio de contexto, utiliza para ellos diversos canales, en el componente MobileContextChannel.

- La Base de Datos tiene un doble cometido. Por una parte almacena el contexto obtenido y por otra sirve de base de conocimiento, donde se guardan las reglas que se usarán en la inferencia.

- SensorListener atiende a cambos de estado en los sensores y almacena los nuevos valores en la Base de Datos.

- El Motor de Reglas se basa en la Base de Conocimiento para obtener conclusiones, que normalmente se almacenan en la Base de Datos. Este motor de reglas se ha separado en tres tipos diferentes: Obtención de contexto de alto nivel, Obtención de datos semánticos para etiquetar contenidos y obtención de conclusiones para realizar acciones basadas en el contexto.

- ContextRetriever se encarga de recoger la información de la Base de Datos, atendiendo a consultas y devolviendo la información deseada.

- El Intérprete actúa de traductor de formatos para cada sensor.

Este framework añade la característica de caché en clientes para paliar el problema de fallo de comunicación en una arquitectura centralizada. Además, tiene en cuenta varios tipos de inferencia, tanto para etiquetar contenido, como para inferir nuevo contexto o concluir acciones (reacción al contexto). Su almacenamiento en Base de Datos posibilita un acceso rápido a la información y una consulta fácil a través de SQL.

3.11. CAMUS

CAMUS [57] es un framework dividido en capas que se encarga tanto de la extracción de información como del razonamiento y entrega del contexto.

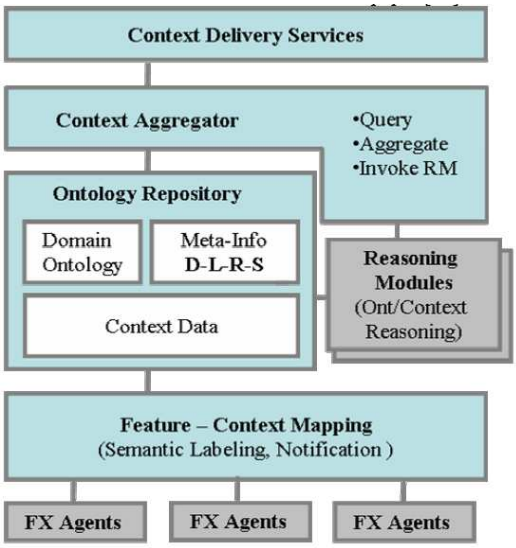

Figura 25. Arquitectura de CAMUS [57]

Todo el contexto se modela a través de ontologías, con lo cual la mayor parte del razonamiento sobre él es asimismo a través de ontologías. La arquitectura consta de los siguientes componentes:

- Agentes de Extracción de Características: Estos agentes recogen el contexto y lo procesan antes de pasarlo a las capas superiores. Extraen las características más relevantes del contexto obtenido y pueden incluso cuantificarlo y segmentarlo, para obtener información con mayor nivel de abstracción y a la vez más expresiva.

- Mapeo de Contexto: Mapea el contexto entre la característica y el valor elemental que puede tener

- Repositorio de Ontologías: Es un repositorio de información que contiene la ontología del dominio completo (conceptos y propiedades), información contextual y meta-información. Así, se almacena tanto el modelo de contexto como los propios datos que lo describen.

- Motor de razonamiento: Un conjunto de posibles motores que pueden adjuntarse y que crean contexto compuesto de alto nivel. Pueden adjuntarse motores que utilicen Redes Bayesianas, Redes Neuronales, motores que usen lógica difusa, etc. Es posible incluso combinarlos entre ellos.

- Servicios de Entrega de Contexto: Se encargan de entregar el contexto deseado a las aplicaciones que lo solicitan.

- Agregador de Contexto: Es responsable de satisfacer ciertas consultas de contexto y proveer de él a las aplicaciones que lo necesiten a través de los Servicios de Entrega de Contexto.

El framework posibilita varias formas de razonamiento sobre el contexto que pueden incluso combinarse, representa toda la información a través de ontologías, lo que permite mucha flexibilidad a la hora de trabajar con un motor de razonamiento y asimismo facilita la inclusión de incertidumbre.

Este framework también realiza extracción de características y cuantización del contexto, yendo un paso más adelante que muchos otros frameworks. Los Servicios de Entrega de Contexto se encargan de propagarlo tanto con consulta como con notificación de cambios.

3.12. CITRON

El Framework CITRON [58] provee a las aplicaciones funcionalidad para adquirir información de contexto y compartirlo entre ellas en un espacio común. La compartición de información tanto dentro de la arquitectura como entre aplicaciones se hace a través del modelo de pizarra, concretamente en una implementación de Tuple Spaces [59] llamada LinuxTuples [60].

Los elementos básicos de CITRON son dos: Citron Worker y Citron Space.

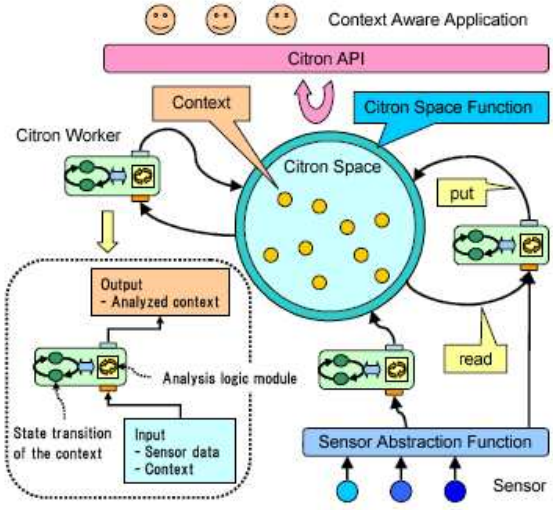

Figura 26. Arquitectura de CITRON [58]

Citron Worker es el módulo de análisis de datos de sensores. Cada Worker obtiene información bien de sensores físicos o del Citron Space y la analiza, para después almacenar su resultado en el Citron Space. Cada Worker se encarga de un tipo de información concreto, existiendo tantos como tipos de información se quieran obtener. Cuando los workers obtienen información de sensores directos en lugar del Citron Space, utilizan un mecanismo llamado Función de Abstracción de Sensores, que es una capa de abstracción de particularidades físicas de cada sensor.

Citron Space es el Tuple Space general para el sistema. También gestiona las peticiones de información tanto de workers como de aplicaciones.

El contexto es el medio de comunicación entre los workers. Pueden combinarse para obtener contexto de más alto nivel. El contexto se representa mediante un formato concreto:

```
Context := {ID, Subject, State, Time, Lag, Interval}
{"ID_walk", "walking", "at_rest", 1107005245, 0, 100}
```

Código 6. Formato del contexto en CITRON [58]

La combinación Sujeto, Estado y Tiempo forman el núcleo de la información, que es aderezada con ciertos metadatos.

Este framework ofrece una forma distribuida de obtención de contexto y una forma eficiente de compartirlo. Aunque está diseñada principalmente para convivir en un solo dispositivo (Muffin [58]), el modelo podría utilizarse fácilmente en implementaciones distribuidas. La forma de obtención de contexto de alto nivel es algo diferente a la tendencia en los frameworks, ya que en lugar de motores de razonamiento y reglas u ontologías, utiliza el encadenamiento de los Citron Workers, lo que puede llegar a resultar limitado en aplicaciones que requieran un gran nivel de razonamiento. Si los Tuple Spaces están centralizados constituyen también un potencial punto de fallo.

3.13. FRAMEWORK DE TAO GU ET AL.

Gu [61] propuso un framework para aplicaciones Context-Aware basado en ontologías, apoyándose en tecnologías como OWL [50], RDF [62] o RDQL [63]. Los componentes básicos de la arquitectura son:

- Adaptadores de Sensores: Capturan datos de sensores lógicos o físicos, y los traducen a formato RDF. Esos datos junto con sus ontologías son la fuente del analizador RDF/OWL.
- Analizador RDF/OWL: Analiza la información de los Adaptadores y comprueba que no haya inconsistencias, tras lo cual construye una estructura en árbol con la información de contexto.
- Extracción del Modelo: Traduce la estructura de datos en árbol a un formato interno más fácil de manejar por el motor de razonamiento.
- Motor sRDQL: Soporta el lenguaje de consulta RDQL para acceder al Repositorio de Contexto. Es una versión reducida que soporta la parte básica de RDQL y está ideada para poder ser soportada por dispositivos de menor capacidad de proceso.
- Motor de Razonamiento: Utiliza un motor de reglas con encadenamiento hacia delante para trabajar sobre las estructuras de datos e inferir nuevas relaciones semánticas, produciendo nuevas tuplas RDF.
- Repositorio de Contexto: Lo constituyen los datos almacenados en Base de Datos sobre el contexto.

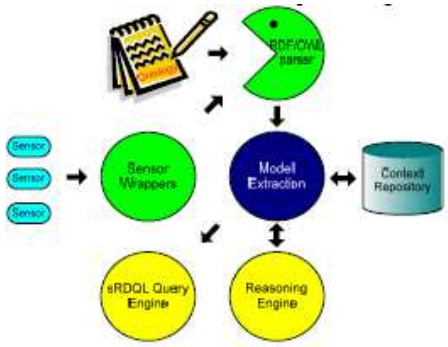

Figura 27. Arquitectura del framework de Gu et al. [61]

El framework tiene una arquitectura básica con los componentes necesarios para captar contexto y razonar sobre él. La utilización de ontologías le proporciona habilidad y flexibilidad para el razonamiento y la comprobación de consistencia del contexto. No soporta añadir nuevos motores de razonamiento ni está preparado para hacer agregación o caracterización, agrupación o cuantización del contexto. Se centra en el razonamiento y extracción pero el propio framework no provee de mecanismos de propagación. Aun así es una buena base para razonadores semánticos.

3.14. ANÁLISIS CONJUNTO DE ARQUITECTURAS

Las arquitecturas vistas anteriormente emplean diversas técnicas y aproximaciones diferentes para crear aplicaciones Context-Aware. Los frameworks se han ideado para ayudar al proceso de desarrollo de aplicaciones y por tanto pretenden solucionar muchas problemáticas características de estos entornos. Algunos autores han propuesto las áreas que una arquitectura debería cubrir. Dey y Abowd [64] proponen las siguientes áreas de análisis:

- Obtención de datos de sensores: Abstracción y desacoplamiento de los detalles físicos de los sensores con respecto a la lógica de la aplicación.
- Preproceso de los datos: Corrección de errores, formateo primario de datos e incluso un razonamiento limitado sobre los datos obtenidos.
- Almacenamiento: Mecanismos para poder guardar el contexto obtenido.

Estas áreas corresponden la base de lo que todo framework debe contener, y en mayor o menor medida se contemplan en todos los frameworks vistos.

Guan et al. [65] proponen otro conjunto de áreas más completo:

- Obtención de datos de sensores: No solo datos de sensores físicos, sino de cualquier tipo de ellos.
- Modelado del contexto: Cómo se modela y registra el contexto es vital a la hora de poder mantenerlo, actualizarlo y trabajar con el para, entre otros aspectos, razonar sobre él.
- Almacenamiento del contexto: Mecanismos para poder guardar el contexto obtenido.
- Razonamiento sobre el contexto: Esta área comprende técnicas avanzadas de razonamiento para obtener contexto complejo y de alto nivel.
- Descubrimiento de contexto: Un factor importante propuesto por Guan et al. es el hecho de descubrir fuentes de contexto. Siendo tan heterogéneo y pudiendo estar distribuido en componentes, es necesario en ocasiones que existan maneras de averiguar qué componente puede ofrecer un tipo de contexto determinado.

Este conjunto es más completo y pertenece a aplicaciones context-aware más maduras. Muchos frameworks vistos cubren estas áreas.

Otra categorización completa de los requisitos que debe tener una buena arquitectura Context-Aware la da Baldauf [66], que propone:

- Sensorización: No solo datos de sensores físicos, sino de cualquier tipo de ellos.
- Modelado de Contexto: Cómo se modela y registra el contexto es vital a la hora de poder mantenerlo, actualizarlo y trabajar con el para, entre otros aspectos, razonar sobre él.
- Procesado de Contexto: Razonamiento sobre el contexto.
- Descubrimiento de Contexto: Mecanismos para averiguar qué componente puede ofrecer un tipo de contexto determinado.

- Datos Históricos de Contexto: Implica además del almacenamiento de la información de contexto, almacenar un historial con los datos que se han recogido a lo largo del tiempo, para ver la evolución de los datos.

- Seguridad y Privacidad: Los datos de contexto pueden ser de tan amplio tipo que muchas aplicaciones incluirán datos privados de un usuario o sensibles de alguna u otra forma. Es necesario que el framework ofrezca mecanismos de protección de esos datos.

Este último conjunto de áreas es mucho más amplio y ofrece aspectos avanzados propios de aplicaciones Context-Aware más modernas. Sin embargo, una categorización más completa podría ser:

- Sensorización: No solo datos de sensores físicos, sino de cualquier tipo de ellos.

- Pre-proceso de contexto: Se trata del procesado previo que se hace en la capa de sensorización. Manejo de errores, formateo básico y agregación básica.

- Modelado de contexto: Cómo se modela y registra el contexto es vital a la hora de poder mantenerlo, actualizarlo y trabajar con el para, entre otros aspectos, razonar sobre él.

- Almacenamiento: Mecanismos para poder guardar el contexto obtenido.

- Histórico de Contexto: Almacenar un historial con los datos que se han recogido a lo largo del tiempo, para ver la evolución de los datos.

- Compartición de Contexto: Posibilidades de compartición de contexto con otras entidades.

- Razonamiento: Esta área comprende técnicas avanzadas de razonamiento para obtener contexto complejo y de alto nivel.

- Entrega de Contexto: Servicios y mecanismos de entrega y propagación de contexto que ofrece el framework.

- Descubrimiento de Contexto: Mecanismos para averiguar qué componente puede ofrecer un tipo de contexto determinado.

- Seguridad: Mecanismos de protección de datos.

SENSORIZACIÓN

Todos los frameworks ofrecen de alguna u otra forma una capa de sensorización que abstrae de los detalles físicos o particularidades de los sensores, bien sean físicos (componentes Hardware), lógicos (combinación de información) o virtuales (acceso a servicios o aplicaciones externas) [67]. Tras el análisis de diferentes frameworks, se puede observar que la forma de sensorización puede llegar a tener diferentes formas, parecidas a las enunciadas por Baldauf et al. [66]:

- **Acceso directo a sensores**: Se trata de no realizar abstracción y acoplar totalmente el middleware a la capa física. Ningún framework analizado utiliza esta perspectiva.

- **Adaptadores**: Componentes separados que se encargan de obtener la información de contexto. Por regla general, existe un componente separado por tipo de sensor. Estos componentes proveen servicios o APIs con interfaz común para obtener la información deseada. Esta aproximación la utilizan los Widgets de Context Toolkit, los Adaptadores de Hydrogen y el framework de Tao Gu, los Context Providers de SOCAM, los SensorListener de CASS, los Agentes de Extracción de CAMUS y los Servidores de Recursos de Context Managing Framework.

- **Servidores o Componentes de Contexto**: Un componente central se encarga de la abstracción del contexto en general, accediendo a cada sensor. Estos componentes internamente se encargan de las particularidades, pero ofrecen un punto común de acceso al resto de componentes. Esta aproximación es propia del Observador de Contexto de Aura, el Servicio de Contexto de Gaia y el Broker de COBRA.

- **Middleware**: Se trata de una capa especializada en abstracción que usan los componentes del framework para acceder a sensores. De esta forma los Context Workers de Citron acceden a través de la Función de Abstracción de Sensores o el módulo de Captura de sensores es utilizado por los componentes de Cortex.

La aproximación más flexible es utilizar adaptadores, que bien pueden incluso reutilizarse en varios sensores que utilicen los mismos protocolos y/o formatos. Dependiendo de las necesidades, pueden combinarse con un componente central o Servidor de Contexto para centralizarlos.

Esta capa de abstracción ofrece unos servicios a los que se accede para obtener los valores deseados, siendo ésta la responsable de hablar el protocolo adecuado con cada sensor. De esta manera, se consigue un desacoplamiento entre lógica de negocio y capa física, de forma que en el momento de cambio de sensor ninguna aplicación ni otra capa del framework se vea afectada. Debido al gran número de sensores posibles y la inmensa variedad de tipos y protocolos, esta capa es una de las más importantes. Ningún framework la ha pasado por alto, también debido a que el concepto de abstracción de la capa física lleva instaurado en la Ingeniería del Software muchos años. Desde los Widgets de Context Toolkit, al Observador de Contexto de Aura o el Servicio de Contexto de Gaia, pasando los Adaptadores de Hydrogen o los Proveedores de contexto de SOCAM, todo framework se ha conseguido abstraer de la capa física.

PRE-PROCESO DEL CONTEXTO

Este concepto se refiere a un parseo previo de la información recibida desde los sensores. Este tratamiento previo pretende corregir errores, omitir valores fuera de rango o incongruentes y un primer cambio de formato para las capas superiores de un framework. También en esta fase se puede hacer una agrupación de valores en categorías o rangos más difusos si es necesario que no se den valores muy detallados.

Todas estas tareas que se realizan con la información de contexto facilitan enormemente el diseño de los sistemas.

No muchos frameworks lo contemplan. Los Adaptadores de Hydrogen (que además pueden ampliarse), el Servidor de Recursos de Context Managing Framework y los Agentes de Extracción de CAMUS tienen en cuenta este aspecto. En context Managing Framework además se realizan labores de cuantización y extracción de características. En Context Toolkit este trabajo lo realizan los Intérpretes.

MODELADO DE CONTEXTO

Todo framework ha de representar el contexto de una u otra forma. La forma de representación del mismo influye directamente en cómo se maneja, lo flexible que puede resultar y cómo puede obtenerse y modelarse contexto de más alto nivel. En siguientes secciones de este trabajo se detalla este aspecto de modelado.

Los Stick-e Documents utilizan lenguaje de marcado para la representación, Context Toolkit utiliza conjuntos de pares clave-valor para representar valores. Aura y CASS utilizan una Base de Datos relacional. Formas más sofisticadas de contexto son utilizadas en Context Managing Framework (RDF), Tao Gu et al. (RDF/OWL), SOCAM (OWL), CAMUS (OWL) y COBRA (OWL), que utilizan ontologías para el modelado. Gaia ofrece una representación con predicados de lógica que normalmente utilizan 4 parámetros y se convierten casi instantáneamente en reglas. CITRON utiliza un sistema de Tuple Spaces con predicados de lógica. Contando con los meta datos, estos predicados resultan de 6 parámetros. El framework Hydrogen, por su parte, utiliza un modelo orientado a objetos para la representación.

Aunque en posteriores secciones se detalla este razonamiento, las ontologías resultan ser una forma muy flexible y prometedora de trabajo con el contexto.

ALMACENAMIENTO E HISTÓRICO DEL CONTEXTO

El almacenamiento del Contexto es un aspecto básico en muchos frameworks, ya que posibilita y facilita tanto el histórico como la compartición de los datos de contexto.

Context Toolkit, COBRA, CORTEX, SOCAM, CASS, Tao Gu et al. basan sus estrategias de almacenamiento en Bases de Datos relacionales. Esto tiene la ventaja de posibilitar el uso del lenguaje SQL para recuperar y manejar toda la información, con un gran nivel de abstracción. CASS y COBRA además utilizan el concepto de Base de Conocimiento para el almacenaje. CASS utiliza además esta Base de Datos para almacenar todas las reglas que el motor de inferencia utiliza.

Aura y Gaia utilizan un Sistema de Archivos distribuido para almacenar la información. Ambos frameworks están orientados a la migración de tareas de usuario y se diseñaron desde el punto de vista de un Sistema Operativo, por lo que su forma inherente de almacenamiento es un sistema de archivos. El sistema de Gaia además puede obtener toda la información organizada a través de variables de contexto configurables.

Frameworks como CITRON utilizan el concepto de Tuple Spaces [59] para el almacenamiento, en la que todos los extractores de datos y todos los razonadores que se encuentran distribuidos almacenan su información en un espacio de información común. Este Tuple Space está a su vez basado en Base de Datos.

El framework Hydrogen y su modelado en forma de Objetos no posibilita almacenamiento, aunque un sistema de mapeo de objetos a Base de Datos relacional [68] no sería excesivamente difícil de aplicar para ampliarlo. Habría que averiguar sin embargo cómo afectaría a su rendimiento.

Los frameworks CASS, COBRA; Context Toolkit, CORTEX, Gaia, Aura y SOCAM además de almacenar, utilizan este medio para crear un historial de datos. Este historial es muy útil para realizar tareas de recolección de datos pasados, observación de evolución de datos, recogida y obtención de estadísticas e incluso posibilita utilizar técnicas de predicción de datos, para estimar cuáles serán los siguientes valores a obtener. Las técnicas de estimación son muy convenientes para aportar fiabilidad, ya que en caso de fallo puntual de cierto sensor se puede al menos utilizar su valor previsto.

COMPARTICIÓN

Este concepto está muy relacionado con el de almacenamiento y el medio de acceso, ya que en la mayoría de los casos la forma de almacenar el contexto va a marcar la forma de compartirlo.

Los frameworks que utilizan el modelo de **Tuple Spaces o modelo Blackboard** tienen inherentemente un medio de compartición de la información. Los frameworks que utilizan sistemas de **ficheros distribuidos** también tienen automáticamente implementada su forma de compartición.

Respecto al medio de acceso, frameworks como Context Managing framework, Hydrogen, SOCAM, COBRA, CASS y CAMUS utilizan un **medio centralizado** del tipo servidor de contexto, al que varios componentes y aplicaciones acceden para obtener el contexto, utilizan este punto de acceso como medio de compartición.

Hydrogen utiliza un tipo de compartición servidor de Contexto para compartir datos entre aplicaciones, pero este componente reside en el mismo dispositivo donde se ejecutan. Sin embargo también dispone de un **sistema en red** (en este caso un sistema Peer-to-Peer) para compartir datos entre dispositivos.

RAZONAMIENTO

Este concepto implica transformación del contexto y utilización de conocimiento. Estos procesos normalmente requieren de motores de razonamiento que se basan en bases de conocimiento para realizar el trabajo. En frameworks como Gaia los predicados almacenados en reglas se utilizan como base de conocimiento para su motor de inferencia, y así poder extraer nuevo contexto. En Gaia este razonamiento está embebido en el Servicio de Contexto.

Otros frameworks como CASS; COBRA, tao Gu et al. y SOCAM utilizan ontologías para modelar el contexto y sus motores de inferencia son de tipo semántico. En CORTEX todo el proceso se encapsula en los llamados Sentient Objects. Al poder encadenarse unos con otros, puede generarse razonamiento de más alto nivel. Este framework jerarquiza el contexto en forma de árbol de tal forma que en cada momento sólo una parte del árbol está activa. Por tanto, sólo un subconjunto de reglas se tiene en cuenta en cada

momento. Una técnica de encadenamiento parecida utiliza CITRON, ya que sus Context Workers pueden encadenarse (siempre a través de los Tuple Spaces) para obtener razonamiento de más alto nivel.

Los agregadores de Context Toolkit pueden hacer tareas de razonamiento. Hydrogen es uno de los pocos casos en los que el razonamiento se deja de la mano de la capa de aplicación.

ENTREGA DE CONTEXTO

En los frameworks una vez obtenido y almacenado el contexto, ha de entregarse a las aplicaciones y componentes que lo soliciten. Sea por el medio que sea (centralizado en un componente servidor, en un Blackboard o una tipología en red) el contexto ha de entregarse y dispersarse. Tras el análisis de arquitecturas, se hace notar que los medios de entrega utilizados en las arquitecturas son:

- Por **petición**: Ciertas informaciones son requeridas en momentos específicos y para ello el componente interesado hace una petición síncrona, a la que se devuelve el valor. Este modelo es útil si no se desea recibir periódicamente un valor o si es un valor que no cambia con frecuencia. Por regla general, pedir un valor síncronamente que no tiene mucha frecuencia de cambio origina poco tráfico de red.

- Por **notificación**: Se utiliza un medio de suscripción/notificación para ser informado de los cambios de información. Los interesados en informase de ciertos datos de contexto se suscriben al componente pertinente, quien informa a la lista completa de suscritos en cuanto se sufre un cambio de estado. Este medio eficiente en datos que tienen tendencia a cambiar, y el hecho de ser la fuente de contexto quien informe en lugar de ser los peticionarios quienes consulten cada poco tiempo, puede ahorrar mucho tráfico de red y mucho tiempo. Sin embargo, puede que la información que se obtenga periódicamente pierda valor a lo largo del tiempo o en ciertos momentos no sea tan relevante. Para ellos, autores como Brewington y Cybenco [69] han desarrollado modelos para convertir este mecanismo en un mecanismo de petición, pero estimando cuándo es el mejor momento para solicitar la siguiente consulta. Se basan en observaciones de valores pasados (histórico) a lo largo del tiempo para estimar tasas y frecuencias de cambio en la información. Su modelo fue ideado para el mundo web, pero puede ser fácilmente aplicable a cualquier situación en la que se deban monitorizar valores.

- Por **latidos**: se maneja por un proceso de suscripción/recepción en el que los interesados en la fuente de datos se suscriben para recibir la información. El emisor de datos envía periódicamente dicha información, que el componente pasará a todos los suscritos. Es muy parecido al mecanismo de notificación salvo que la información se envía periódicamente haya cambiado o no. Pueden enviarse datos que no sean susceptibles de cambio o simplemente señales. Gaia utiliza este sistema cuando cada aplicación que se está ejecutando manda periódicamente una señal para indicar que está activa. Este mecanismo de latidos, puede convertirse en notificación según las necesidades del receptor. Si éste solo está interesado en el cambio, el componente que actúa de mediador y recibe los latidos, puede limitarse a notificar únicamente cuando el latido cambie de frecuencia o cese.

Todos los frameworks analizados utilizan tanto mecanismos de petición como de notificación. Gaia también soporta latidos.

DESCUBRIMIENTO DE CONTEXTO

Los sensores en un entorno pueden fallar, pueden desaparecer de él o pueden llegar nuevos. Contexto que un tiempo atrás el sistema no era capaz de recoger, con la llegada de nuevos sensores puede posibilitarse. Es conveniente por tanto un mecanismo para descubrir y localizar fuentes de contexto. Hay diversas tecnologías que ofrecen descubrimiento. UPnP [70] dispone del protocolo de descubrimiento SSDP [71], JINI [72] también ofrece descubrimiento de componentes, existen otros protocolos como SALUTATION [73] o SLP [74]. Tecnologías como Bluetooth [75] ofrecen medios para descubrimiento de dispositivos. Los medios más típicos son:

- Mecanismo **distribuido** de descubrimiento: En la que se realiza un descubrimiento descentralizado y una consulta a través de los componentes que se encuentran distribuidos en el entorno.

- Mecanismo de **registro**: Técnica centralizada en la que los componentes que proporcionan datos se apuntan al registro al llegar y se borran de él al finalizar. Para prevenir fallos pueden usarse técnicas de

latido, en la que se envía una señal periódica para indicar que el componente sigue activo. El consumidor consultará el registro para comprobar qué proveedores de contexto hay disponibles.

No todos los frameworks analizados poseen descubrimiento de contexto. Los que lo proporcionan lo hacen a través del mecanismo de registro. Context Toolkit dispone de un Descubridor con un registro para detectar Widgets disponibles. Utiliza también la técnica de latido para comprobar el estado de cada Widget. SOCAM ofrece un servicio de descubrimiento llamado Servicio de Localización de Servicios. En Gaia, los Proveedores de Contexto se registran también en un registro. En Aura, los diferentes Suministradores se registran en el Gestor de Entorno.

SEGURIDAD Y PRIVACIDAD

Los datos de contexto pueden incluir información sensible y personal que debe ser protegida. Un framework debería ser capaz de proveer de mecanismos para asegurar la privacidad de los datos.

Context Toolkit introduce el elemento de Dueño del Contexto, en el que al contexto obtenido se le asigna un dueño. Este dueño puede acceder al contexto de otro dueño siempre y cuando esté autorizado. COBRA incluye un lenguaje de control de acceso llamado Rei [76]. Este lenguaje ofrece conceptos como derechos, prohibiciones, obligaciones y dispensaciones.

Este aspecto es crucial si se tratan grandes volúmenes de información sensible que incluso podría dar problemas legales a aplicaciones. Sin embargo, pocos son los frameworks que tienen en cuenta estos aspectos de seguridad.

Aspectos de autenticación van muy relacionados con los de privacidad, ya que para permitir el acceso a una entidad, ella ha de autenticarse como ella misma inequívocamente. Otros aspectos relacionados son garantizar que la información no ha sido modificada o incluso un transporte seguro utilizando mecanismos de cifrado.

A continuación se ofrece una tabla comparativa que resume todo lo dicho en este apartado.

	SENSORIZ.	PRE-PROCESO	PROCESADO DEL CONTEXTO	ALMACEN.	HISTÓRICO	COMPARTIC.	RAZONAM.	ENTREGA	DESCUBR.	SEGURIDAD
Context Toolkit	Adaptadores	En Intérpretes	Clave-Valor	BD	Sí	Centralizada	Con Agregadores	Petición / Notificacion	Por registro	Dueño del documento
Aura	Servidores de contexto	No	BD	Sistema de Archivos Distribuido	Sí	Distribuida (FS)	-	Petición / Notificacion	Por registro	Control de Acceso
Context Managing Framework	Adaptadores	En Servidores de Recursos	Ontologías (RDF)	BD	No	Centralizada	Razonador Semántico	Petición / Notificacion	Por registro	-
Gaia	Servidores de contexto	No	Predicados lógicos	Sistema de Archivos Distribuido	Sí	Distribuida (FS)	Motor de reglas	Petición / Notificación / Latidos	Por registro	Control de Acceso
Hydrogen	Adaptadores	En Adaptadores	Orientado a Objetos	-	No	Centralizada / En red	-	Petición / Notificacion	-	-
SOCAM	Adaptadores	No	Ontología (OWL)	BD	Sí	Centralizada	Razonador Semántico	Petición / Notificacion	Por registro	-
CORTEX	Middleware de contexto	No	BD	BD	Sí	Centralizada	Encadenamiento de Sentient Objects	Petición / Notificacion	Por registro	-

COBRA	Servidores de contexto	No	Ontolgía (OWL)	Base de Conocimiento sobre BD	Sí	Centralizada	Razonador Semántico	Petición / Notificacion	-	Control de Acceso (Rei)
CASS	Adaptadores	No	BD	Base de Conocimiento sobre BD	Sí	Centralizada	Razonador Semántico	Petición / Notificacion	-	-
CAMUS	Adaptadores	En Agentes de Extracción	Ontología (OWL)	BD	No	Centralizada	Razonador Semántico	Petición / Notificacion	-	-
CITRON	Middleware de contexto	No	Predicados de lógica	Tuple Spaces	No	Distribuida (Tuple Spaces)	Encadenamiento de workers	Petición / Notificacion	-	-
TAO GU et al.	Middleware de contexto	No	Ontologías (OWL/RDF)	BD	No	-	Razonador Semántico	Petición / Notificacion	-	-

Tabla 2. Tabla-resumen de arquitecturas Context-Aware

4. UNA ARQUITECTURA PARA DISPOSITIVOS MÓVILES

Las arquitecturas analizadas mayormente están preparadas o ideadas para equipos y dispositivos de sobremesa, con más potencia y capacidad. Algunas están efectivamente conceptuadas para dispositivos con más limitaciones e incluso móviles, pero se limitan a ajustar el framework a un hardware con pocos recursos de procesador. Crear una arquitectura ligera para poder ser ejecutada en dispositivos móviles es uno de los factores más importantes, pero no es el único. Ninguno de los frameworks expuestos aplaca los problemas que surgen en las aplicaciones Context-Aware para dispositivos móviles. Autores como Hofer et al. [47] argumentan que una aplicación Context-Aware móvil ha de tener las siguientes características:

- **Ligera**: Aunque los dispositivos móviles cada vez tienen más potencia de proceso, en la actualidad una arquitectura para móviles necesita ser mucho más ligera que la de una máquina de sobremesa o un súper servidor.

- **Extensible**: Las limitaciones hardware hacen que un dispositivo móvil no pueda tener todos los sensores que puedan necesitarse, así que un framework debe poder posibilitar una forma de recoger información de sensores remotos además de los locales.

- Robusta frente a **desconexiones**: Un teléfono móvil tiene el inconveniente de conexiones inestables, así que un framework debe estar preparado para esta desventaja.

- Con **meta información de distancia**: Se debe obtener meta información acerca de la distancia del sensor con respecto al dispositivo para poder establecer unas prioridades con respecto a qué sensor está más cerca. Los sensores más lejanos pueden ofrecer información no muy fiable.

- Debe **compartir** el contexto: Un framework para dispositivos móviles debe compartir el contexto con otros dispositivos.

Hofer et al. se acercan a las necesidades de dispositivos móviles. Éstos, por su naturaleza, son de naturaleza muy cambiante. Acompañando al usuario en todo momento, cambian de emplazamiento físico constantemente, donde otros sensores y fuentes de conexión le rodean. Donde el contexto puede variar enormemente e incluso donde la intención de uso del dispositivo por parte del usuario puede variar radicalmente.

En un escenario típico, Bob sale de su lugar de trabajo a las calles de la ciudad para dirigirse a casa. Cuando estaba en el lugar de trabajo, su PDA estaba conectada de forma inalámbrica con la red de la oficina, donde Bob podía acceder a información del tráfico y del tiempo. Un programa instalado tenía acceso a sensores de temperatura situados por la oficina. Ahora en la calle la conexión de la PDA ha cambiado a GPRS y ya no hay acceso a ningún sensor de temperatura existente, con lo que la aplicación instalada ha dejado de dar información.

Un framework para dispositivos móviles ha de estar preparado para cambios de contexto constantes, cambios físicos de lugar, nuevos sensores dentro de la misma localización y sensores que desaparecen, movimiento constante que entra en rango con unos dispositivos y deja fuera de rango a otros, cambios de conexión y un largo etc. Incluso la intención del usuario puede cambiar rápidamente. En el escenario anterior Bob probablemente no quiera realizar las mismas tareas tras salir del trabajo.

Por todo ello, un framework debe contemplar las siguientes áreas:

Ligereza: La arquitectura ha de ser ligera y permitir la ejecución en dispositivos de capacidad limitada. Ello conlleva consecuencias de pérdida de características, pero es necesario que una arquitectura sea capaz de ejecutarse localmente en un dispositivo móvil.

Extensibilidad: Los sensores en un dispositivo son diferentes dependiendo del hardware, ciertos fabricantes implantan unos y dejan carencias de otros diferentes. Con la extensa variedad de medios a los que acceder para obtener la información, una arquitectura de estar abierta a nuevas formas de acceso a sensores.

Robustez ante **desconexiones**: La alta movilidad de estos dispositivos hace que la cobertura de acción cambie con mucha frecuencia. Al cambiar de lugar físico el dispositivo sale del rango de cobertura de ciertos puntos de acceso y entra en cobertura con otros. Esto hace que las conexiones sean muy inestables. Se ha de proveer de formas de asegurar el funcionamiento de las aplicaciones ante estas desconexiones tan frecuentes.

Abstracción del **medio de transmisión de datos**: Relacionado al aspecto de movilidad y reconexión frecuente, puede que un dispositivo se desconecte de un punto de conexión y quede incapacitado para la comunicación hasta que encuentre otro. Además de eso, este siguiente punto de acceso puede proporcionar conexión al dispositivo pero por un medio totalmente diferente, de forma que de una manera frecuente se pase de una conexión de un tipo, por ejemplo Bluetooth, a oto diferente como puede ser Wi-Fi. Además del hecho citado en el punto anterior en el que el framework debe soportar estados de desconexión, se hace necesario que soporte también la comunicación por varios medios de transmisión de datos.

Capacidad para medir la **Calidad del Contexto**: En entornos especialmente cambiantes, el contexto puede ver degradada su calidad. Pérdida de acceso a sensores o pérdidas de conexión son factores muy comunes de pérdida de precisión y calidad en el contexto. Además, una vez se define el modelo teórico del contexto, llevado a la realidad se hace notar que la información no siempre resulta ser 100% fiable. Para poder trabajar (y sobre todo razonar) de manera correcta con el contexto también es necesario saber su calidad definiendo grados de Calidad de Contexto (QoC) [77]. En los siguientes apartados se profundiza más en este aspecto.

Capacidad para tener contexto útil en poco espacio de **almacenamiento**: Debido a las restricciones de espacio de los dispositivos móviles, el framework debe dar facilidades para trabajar sobre un contexto de tamaño más reducido, de forma que aplicaciones que corren sobre hardware con bajas capacidades de almacenamiento y poca memoria, puedan seguir trabajando con el contexto que necesitan.

Capacidad para **razonar** sobre **incertidumbre** y cambios frecuentes de contexto: Al llevar el modelo teórico a la realidad, se hace notar también que la información que se recibe no es 100% segura. La fiabilidad depende, además de la Calidad del Contexto, de la propia realidad donde no puede saberse con el 100% de probabilidad que el contexto de alto nivel inferido es el que se desea tener. La realidad es incierta de por sí y un módulo de razonamiento ha de ser capaz de lidiar con ello, sobre todo en entornos tan cambiantes y dinámicos como los móviles.

Capacidad para recoger contexto altamente **cambiante**: No sólo es necesario que el framework tenga capacidad para razonar en entornos altamente cambiantes, sino que además debe ser capaz de recoger el contexto de manera eficiente, dando herramientas a las aplicaciones para lidiar con conjuntos de sensores que cambian frecuentemente a lo largo del tiempo.

Compartición de contexto: La compartición de contexto es un medio muy útil de que cada dispositivo, con sus limitadas características, sea capaz de obtener contexto que no puede procesar o información a la que no puede acceder. Siempre etiquetando la correcta calidad y fiabilidad del contexto, es conveniente que un framework de facilidades para que un sistema con diferentes dispositivos compartan información entre sí, además de acceder a contextos remotos que otros dispositivos han sabido captar. Bien porque se encuentran a una distancia más adecuada de las fuentes de información, o bien porque su hardware dispone de sensores adecuados para captar la información. En aplicaciones móviles, el software corre sobre hardware de muy diferentes características, donde cada modelo tiene su propio conjunto de sensores, que pueden o no coincidir con el resto de dispositivos. Compartir el contexto maximiza las posibilidades de obtención de datos lo más completos posible.

4.1. UNA ARQUITECTURA QUE CUBRE LAS ÁREAS DE UN FRAMEWORK PARA DISPOSITIVOS MÓVILES

El siguiente modelo de arquitectura propuesto cubre las características anteriormente expuestas para dar servicio a aplicaciones móviles Context-Aware. Esta arquitectura está ideada para dispositivos móviles del tipo PDA y teléfonos móviles, quedando excluida para hardware embebido y restricciones físicas más fuertes.

Esta arquitectura se basa en las comunicaciones Peer-to-Peer, con lo que cada dispositivo está conectado al resto. Además de lograr una interconexión entre dispositivos, cada uno de ellos contiene el framework instanciado, sin necesidad de acceder a servidores para ejecutar las funciones principales. Esto conforma una

arquitectura descentralizada en la que no es necesaria la presencia de servidores centrales, aunque la presencia de servidores no está descartada. Toda la interconexión crea una red Peer-to-Peer de la que el dispositivo podrá beneficiarse, pero de la que no tiene por qué depender.

Al disponer cada punto de toda la funcionalidad de los elementos del framework, siempre y cuando localmente se disponga de suficientes sensores y captadores de información el framework podrá ser independiente de la red.

Sin embargo, si la tiene disponible podrá beneficiarse de ella, ya que podría comunicarse con otros puntos. Para un dispositivo de la arquitectura, el resto de puntos de la red serán fuentes de información de contexto a las que podrá consultar. Estos puntos pueden ser tanto elementos simples que extraigan contexto básico o elementos más sofisticados capaces de inferir contexto de alto nivel. De esta manera, si un dispositivo de forma local no dispone de suficientes sensores, podrá preguntar por la información de contexto en la red.

Asimismo, podrá preguntar tanto por contexto básico como por contexto elaborado de alto nivel. Cada dispositivo podrá razonar de forma local sobre el contexto adquirido gracias a los servicios del framework, y podrá inferir contexto complejo, que posteriormente almacenará y podrá compartir o no dependiendo de sus preferencias. De esta manera se crea un modelo de razonamiento distribuido por toda la red en la que cada punto extrae contexto de forma local o desde otro punto, razona sobre él y lo comparte de vuelta en la red.

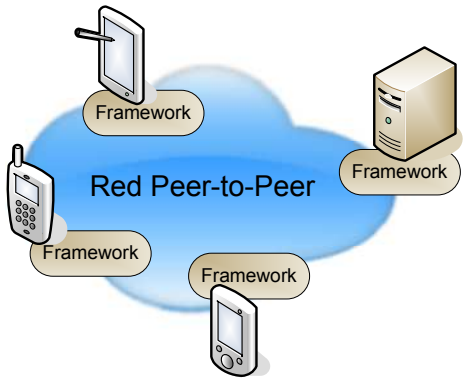

Figura 28. Arquitectura Peer-to-Peer del framework

Aunque la arquitectura está pensada para móviles, está abierta para que otros nodos más potentes puedan usarla y asimismo ser parte de la red punto a punto. Esto es especialmente útil si estos puntos tienen suficiente capacidad de proceso para hacer razonamiento complejo, que más tarde los dispositivos móviles puedan utilizar en su beneficio. Por su parte, estos nodos potentes al formar parte de la red dispondrían de un conjunto de sensores desplegados por el espacio físico haciendo de proveedores de información.

Debido a las características de los teléfonos y PDAs, en numerosas ocasiones no sería necesario desplegar instalaciones de sensores físicos, ya que cada terminal actúa a su vez como un conjunto de sensores que ofrece la información deseada. El simple hecho de existir en la red terminales de estas características, hace que existan todo tipo de sensores disponibles. Asimismo cada dispositivo no necesitará instalaciones extra si dispone de todo el hardware que necesita para su sensorización.

Esta arquitectura distribuida está pensada para tener un pequeño tamaño, suficiente para poder ser ejecutada en PDAs y teléfonos móviles de alta gama. Con los avances de este hardware no es necesario hacer una implementación excesivamente pequeña, peri sí hay que tener consideraciones como hacerla adaptable a diferentes niveles de hardware. Esta arquitectura posibilita esto gracias a su filosofía de plugins, que pueden adjuntarse o retirarse, activarse o detenerse para adaptarse a la potencia del dispositivo. Otro factor importante es el razonamiento, que deberá tener en cuenta las limitaciones de los teléfonos móviles. Esta arquitectura llega a una aproximación similar de plugins con los motores de razonamiento, que pueden conectarse o desconectarse, haciendo que pueda llegar a haber varios motores trabajando o en cambio solo uno de ellos.

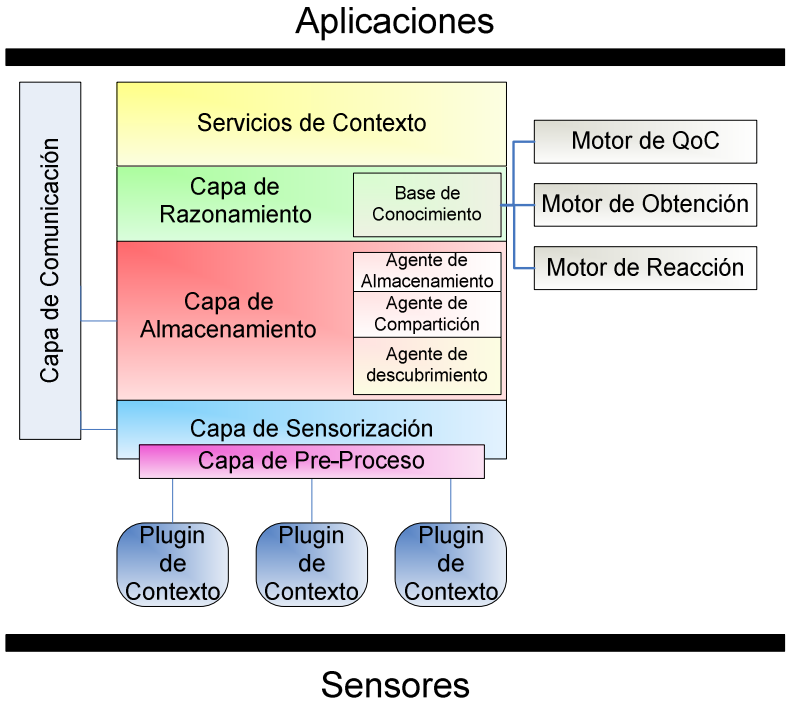

Figura 29. Arquitectura interna del framework

Esta arquitectura, debido a sus características móviles y los servicios que requiere, podría ser válida para plataformas como .NET [78] o J2ME [79]. Esta última parece más adecuada por su mayor aceptación en teléfonos móviles, su soporte en PDAs y su facilidad para ejecutar en servidores. Además, desde el punto de vista del razonamiento, hay trabajo de motores realizado en este lenguaje [80] [81] [82] [83].

El intercambio de mensajes entre los plugins de contexto y las capas superiores, así como el intercambio de mensajes a través de control la red Peer-to-Peer podrá hacerse a través de un lenguaje XML [40] que pueda definir las capacidades de las fuentes de contexto. XML soporta muy bien el añadido de los parámetros de Calidad de Contexto, por tanto la negociación de la misma y la operación por la red Peer-to-Peer también puede estar implementada en XML.

El contexto representado se puede dejar preparado para trabajar con los motores semánticos representándose en OWL [50].

Cada nodo en la red posee su framework de forma que las distintas aplicaciones acceden a él. Dentro de cada dispositivo, el framework tiene una arquitectura interna basada en diferentes capas.

Estas capas son:

- Plugins de Contexto
- Capa de Pre-Proceso
- Capa de Razonamiento
- Capa de Almacenamiento
- Capa de Comunicación
- Servicios de Contexto

PLUGINS DE CONTEXTO

Esta parte de la arquitectura es la más cercana a la capa física del sistema. La arquitectura en esta altura está compuesta de diferentes plugins que se adhieren al sistema para adquirir información de contexto. Cada plugin representa un tipo de información que puede obtenerse a través de un sensor y está compuesto de dos partes. La primera es la parte que lidia directamente con la fuerte de información. Ya sean sensores lógicos, físicos o virtuales, cada sensor tiene sus peculiaridades y necesita de un plugin específico para extraer los datos pertinentes. La segunda parte es un enlace con la capa siguiente de pre-proceso de información. La información específica que se recoge del sensor es de un tipo que solo el plugin conoce, y por tanto solo él sabrá hacer la fase de pre-proceso. En la segunda parte se realiza el pre-proceso necesario para dejar la información en un formato que la aplicación pueda manejar.

Los plugins están ideados como módulos independientes, para poder añadirse o retirarse según las necesidades del framework. De esta forma se detectan los plugins de que se dispone y se cargan solamente los necesarios. Además, es posible extender las capacidades de sensorización añadiendo o creando plugins nuevos para nuevos sensores y tipos de información de contexto. Es una forma dinámica de adaptar la sensorización a la inmensa variedad de tipos de sensores que cada dispositivo móvil en particular posee.

Cada plugin puede acceder a la capa de comunicación a través de los servicios que le provee la capa de sensorización que lo alberga. Esto es muy útil sobre todo en sensores lógicos o virtuales que han de extraer los datos de fuentes de información externas. Estos plugins pueden ser capaces tanto de responder a peticiones de consulta, como notificar los cambios de estado que se den en ellos.

CAPA DE SENSORIZACIÓN

Esta capa posee una visión general de todos los medios de captación de contexto local de que se dispone. Puede activar o desactivar plugins a petición para mantener en memoria los necesarios. De esta manera la memoria se mantiene optimizada y se ofrecen sólo los servicios que realmente se piden. Esta capa es quien se encarga de comprobar el buen funcionamiento de los plugins (por ejemplo, comprobar que existe hardware y capa física suficiente para activar un plugin), darles capacidades de comunicación y consultar la parte de pre-proceso de cada uno de ellos. Aunque podría ser capaz de realizar pre-proceso general, el pre-proceso en sí mismo lo delega a cada plugin, actuando de coordinador. El conjunto de plugins en sus partes de preprocesado conforman la **Capa de Pre-Proceso**, que puede encargarse además de analizar los valores de Calidad de Contexto que los propios sensores ofrecen. La capa de sensorización está preparada para hacer consultas a los plugins y recibir notificaciones de ellos si fuese necesario.

CAPA DE ALMACENAMIENTO

Esta capa es la base donde se almacena toda la información de contexto. Las consultas de información desde cualquier punto irán a parar a esta capa. La arquitectura está preparada para que exista la posibilidad de generar notificaciones a otras capas en el momento en el que cierta información cambie de estado. La capa de sensorización almacenará aquí los datos obtenidos y la capa de razonamiento se nutrirá de este nivel para realizar sus tareas.

Además del almacenamiento, esta capa recibe asimismo el contexto exterior que viene detectado por otros dispositivos de la red Peer-to-Peer. Existe un elemento llamado **Agente de Compartición** que tiene toda la lógica de compartición de contexto a través de la red. Recibe información desde otros nodos y envía la información que el dispositivo desea publicar.

Esta es la altura en la arquitectura donde se tiene la visión general de la información de contexto, en el resto de puntos la información es sólo parcial. Estos datos pueden venir de tres diferentes fuentes:

- Sensores de la capa de sensorización, ya sean físicos, lógicos o virtuales.
- Nuevo contexto generado por la capa de razonamiento y generalmente de más alto nivel
- Contexto que llega desde otros puntos de la red Peer-to-Peer

Toda esta información se almacena en un espacio compartido para todas las aplicaciones del dispositivo. Por ello, en esta capa existe un **Agente de Descubrimiento**. Este elemento recibe consultas de contexto que se quiere encontrar y analiza el estado general del almacenamiento para comprobar si es posible proveer de la información que se pide. Primero se analiza localmente en el repositorio, si no se encuentra se accede a la capa de sensorización por si se diese el caso de tener disponible un plugin adecuado, pero desactivado por no haberse utilizado hasta ahora. En tercera instancia se accede al agente de compartición para indagar si existe esa información a través de la red Peer-to-Peer. Este agente puede encargarse de selección de proveedores adecuados de información en base a métricas de Calidad de Contexto.

Esta comunicación entre el Agente de Descubrimiento y el Agente de Compartición también puede darse en el otro sentido, ya que el segundo puede recibir una petición de contexto, buscarla y al no ser encontrada como tal, decidir que sea necesario averiguar si se tiene o se puede obtener, en cuyo caso accede al Agente de Descubrimiento para preguntarlo.

El acceso al descubrimiento es muy común en ocasiones donde se ha perdido conexión con ciertos sensores. Bien porque un nodo de la red Peer-to-Peer se ha desconectado o porque se ha perdido cobertura física con un sensor local. En estos casos la información de contexto desaparece o su calidad es tan baja que se desestima. Cuando se requiere por una aplicación la información de nuevo, puede lanzarse una búsqueda de nuevo para comprobar si es posible mediante algún otro mecanismo (reconexión con el sensor, conexión con otro sensor diferente, petición a la red, …).

Por último, esta capa también contiene un **Agente de Almacenamiento**, al que se accede para recoger los datos de contexto que se necesiten y para almacenar los que se hayan conseguido. Este agente recibe las consultas y comandos y devuelve las respuestas bien en forma de retorno normal (petición-consulta) o bien en forma de notificación si fuese necesario. Quien realmente transmite las notificaciones de cambios de estado dentro de la capa es este módulo. Los peticionarios pueden ser tanto los agentes (dispositivo local) como la capa de razonamiento (propio framework) o el Agente de Compartición (compartir contexto con nodos de la red).

La causa más común de consulta al Agente de Descubrimiento es un retorno nulo del Agente de Almacenamiento, lo cual quiere decir que no se encuentra almacenada ninguna información del tipo que se pide, y por tanto sería necesario descubrir si se puede obtener.

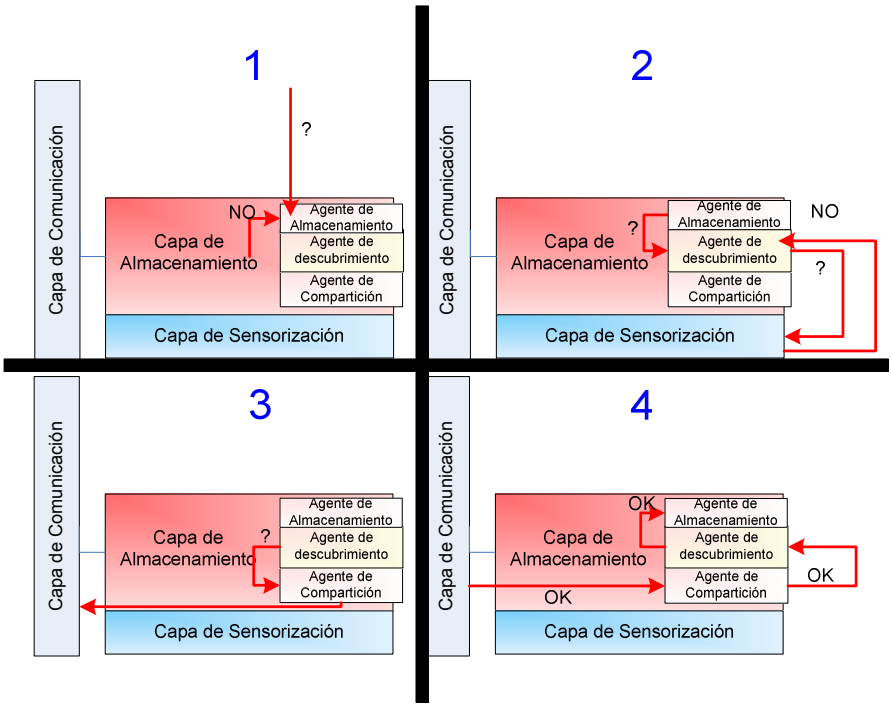

Figura 30. Proceso de descubrimiento de contexto

En la figura anterior se expone un caso de uso posible dentro del framework. Una aplicación pide cierto tipo de contexto (1). El Agente de Almacenamiento busca en el repositorio, el cual devuelve un retorno negativo indicando que no existe almacenado un dato así. El Agente de Almacenamiento entonces decide indagar si es posible obtener el dato (2), con lo que pregunta al Agente de Descubrimiento. Éste a su vez comienza buscando en la Capa de Sensorización. No se encuentra, activado o desactivado, ningún plugin capaz de obtener el dato, así que la primera consulta devuelve un valor negativo. El Agente de Descubrimiento consulta entonces al Agente de Compartición (3), quien indaga en la red si es posible conseguir el dato. Este módulo descubre que es posible conseguir la información a través de la red (4). Se informa al Agente de Descubrimiento, quien a su vez informa al Agente de Almacenamiento que le preguntó.

Estos datos de retorno vienen acompañados de metadatos que indican, entre otros, si la información proviene de forma local o remota. Una vez descubierta la fuente de información será posible hacerle consultas o incluso suscribirse a sus cambios.

CAPA DE RAZONAMIENTO

Esta capa realiza todo el razonamiento que se da lugar en el framework. Utiliza para ello la información de contexto general, tanto la que se ha conseguido localmente en el dispositivo como la recolectada de la red. Para ello, se abstrae de todos los recolectores de contexto, siendo la única fuente de información la capa de almacenamiento. Esta capa hará uso del Agente de Almacenamiento para recuperar la información y para almacenar la que infiere.

El razonamiento dentro del framework puede ser de distinta índole:

- Realizar un mantenimiento del contexto: Como se ha mencionado anteriormente y se detallará en siguientes apartados, tan diversas, dinámicas y móviles fuentes de contexto pueden no tener toda la fiabilidad que se desea. Además, el factor de distancia es muy importante en un framework que puede conseguir información de nodos remotos. En una enorme habitación, conseguir la temperatura de un dispositivo situado a 60 metros de distancia no es tan relevante como conseguirla a nivel local. Un sensor que ha dejado de enviar información debido a una desconexión provoca que un dato en el almacenamiento lleve demasiado tiempo desactualizado, etc. Muchos son los posibles casos en los que la calidad del contexto puede degradarse y es necesario tenerla en cuenta. Mantener en buen estado el contexto es una labor vital y que además ayuda a eliminar contexto innecesario. El factor espacio es vital en dispositivos con limitaciones hardware en este sentido. Este motor trabaja para mantener en buen estado el contexto almacenado:

  ▷ Calcular el grado de Calidad del Contexto (QoC) para averiguar cuán buena y fiable sigue siendo la información y asegurarse de mantener un alto grado de calidad [84].

  ▷ Eliminar inconsistencias e informaciones demasiado desactualizadas en el repositorio.

  ▷ Definir qué contexto es más relevante e incluso hacer filtrado del mismo.

  ▷ Realizar tareas de agrupación, segmentación, cuantización y caracterización.

- Obtener nuevo contexto: El motor de razonamiento obtiene contexto de alto nivel a través de contexto de bajo nivel u otro contexto de alto nivel almacenado en la capa de almacenamiento. El contexto sobre el que se trabaja puede venir de otros dispositivos de la red, que proporcionen contexto a su vez de alto nivel. Razonar sobre éste provoca un funcionamiento en cadena dando lugar a un **razonamiento distribuido**, muy útil para repartir grandes trabajos en tareas más pequeñas a lo largo de la red. Esto resulta crucial al tratarse de dispositivos que no pueden albergar grandes motores de razonamiento debido a sus limitaciones. En la red pueden existir además nodos potentes que ayuden en las tareas pesadas de razonamiento.

- Reaccionar y tener un comportamiento específico sobre el contexto: Esto proporciona la posibilidad de razonar sobre la información de contexto no para obtener nueva, sino para decidir qué acciones tomar e informar de ello a las aplicaciones que lo soliciten. De esa forma se evita que cada aplicación tenga que utilizar su propio mecanismo de razonamiento sobre el framework. Varias instancias de aplicaciones

podrían dar lugar a varios motores de razonamiento ejecutándose además de los propios del framework. De esta forma se aprovechan recursos y se posibilita que el motor de acciones pueda tener acceso más directo a información de contexto. Por otra parte sería posible simplemente no cargarlo y que cada aplicación implemente su mecanismo.

Para ello, esta capa soporta una arquitectura de motores que pueden añadirse parecida al mecanismo de plugins. Esta división en varios motores que pueden añadirse o quitarse tiene dos ventajas principales:

- Desacopla enormemente el resto de componentes de la Capa de Razonamiento con el tipo e implementación del motor concreto.

- Dividir en varios motores permite desactivar los que no sean necesarios, según necesidades o posibilidades del dispositivo.

De esta manera, los motores pueden trabajar en conjunto, pero también es posible activar un solo motor que en realidad haga las tres tareas mencionadas anteriormente. Si se deciden activar varios, cada uno de ellos puede trabajar con un subconjunto dado de la Base de Conocimiento, de tal forma que se dividan las tareas.

SERVICIOS DE CONTEXTO

En esta capa se encuentran en forma de APIs de programación todas las capacidades del framework accesibles para las aplicaciones. Este es el punto de entrada para ellas. Los servicios del resto de capas son envueltos en esta con librerías para ofrecer toda la funcionalidad que se requiere. Aquí hay librerías para acceder al contexto, hacer consultas en el repositorio, realizar suscripciones de información de contexto, configurar el framework, acceder al nivel de razonamiento para manejar la Base de Conocimiento, descubrir nuevas fuentes de información, etc.

Este es el punto de entrada de las aplicaciones, que utilizarán esta capa para comunicarse con el framework. Esta capa provee principalmente de tres interfaces:

- **Consulta** de contexto: Las aplicaciones podrán hacer consulta del contexto que desean obtener y consultar. También podrán elegir el mecanismo de obtención de contexto, que podrá ser consulta-respuesta o bien notificación.

- **Razonamiento**: El framework provee de mecanismos para añadir y manejar el contenido de la base de conocimiento, para completarlo, personalizarlo o cambiarlo si lo desean. Como la capa de razonamiento también provee de motor de reacción a determinados estados del contexto, es posible a través de esta interfaz añadir o indicar el comportamiento que se desea obtener de este motor, de forma que hasta cierto punto sea personalizable.

- **Calidad** de Contexto: Sobre esta interfaz se negocia la calidad de contexto deseada y se configura qué información de calidad se desea obtener junto con la información de contexto.

CAPA DE COMUNICACIÓN

El objetivo principal de esta capa es proveer de acceso a la red de comunicaciones de forma que exista una abstracción del medio físico de transmisión de datos. En un dispositivo móvil es muy común que en el momento de realizar un cambio de localización física se pierda cobertura de puntos de acceso y se gane cobertura de otros. Esto provoca una inestabilidad en el estado de la conexión. Además, es posible que se pierda el alcance a todos los puntos de acceso que dan conexión por un medio de transmisión de datos (por ejemplo Wi-Fi) y se alcance un punto de acceso a otro medio de transmisión (por ejemplo Bluetooth).

Para garantizar una estabilidad en la conexión, la Capa de Comunicación ha de proveer de medios para mantener una comunicación fluida, de forma que una desconexión repentina no ocasione una cancelación de la transmisión de datos. Esta transmisión de datos deberá mantenerse mientras la Capa de Comunicación busca un medio alternativo de envío: otro punto de acceso de la misma tecnología o incluso un cambio a otro medio

de transmisión. Una vez reestablecida una conexión continuará con la transmisión de datos, todo ello sin que el usuario de la Capa de Comunicación se vea afectado.

En el caso de que una transmisión se mantenga retenida demasiado tiempo por no haber disponible ningún punto de acceso en ninguna tecnología disponible, podrá devolverse un error de conexión al usuario de la capa.

Para poder proporcionar varias formas de comunicación por varios medios, esta capa puede contener plugins de comunicación que, de la misma forma que en la capa de sensorización, se puedan cargar y descargar dinámicamente. Cada plugin proveerá de un mecanismo de comunicación a través de una tecnología.

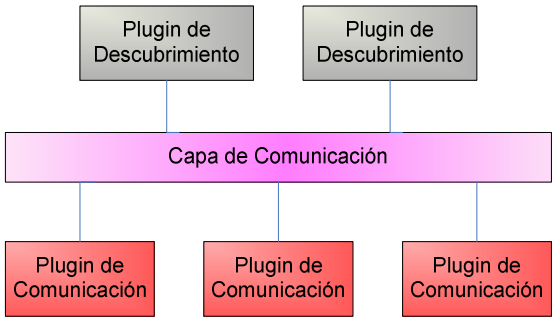

Figura 31. Arquitectura interna de la Capa de Comunicación

Además de estos plugins, pueden existir otros de otro tipo distinto, destinados al descubrimiento de nodos a través de la red. Tecnologías como Bluetooth [75] vienen con mecanismos de descubrimiento que en ocasiones incluso son vitales para realizar conexiones. Cada uno de estos plugins puede implementar uno de los muchos mecanismos de descubrimiento existentes [73] [74] [71] [72].

Esta capa de comunicación del modelo teórico, puede utilizar un modelo práctico que se ajusta a las necesidades: Seamless Communication o Comunicación sin Costuras [85]. Grace et al. [86] en su proyecto REMMOC proponen un mecanismo de comunicación sin costuras entre los componentes, capaz de sobreponerse a pérdidas de conexión e incluso utilizar otras tecnologías de comunicación en la misma conexión lógica. Dadas las capacidades heterogéneas de la red, este mecanismo proporciona fiabilidad estabilidad a las conexiones. Siegemund et al. [87] proponen un mecanismo de descubrimiento para los objetos de una red, en la que las condiciones cambiantes hacen que las direcciones no puedan ser fijas, los autores proporcionan formas de descubrimiento para poder superar el problema de los altos cambios en los nodos de las redes. Ghay et al. [88] también proponen una arquitectura de comunicación sin costuras, pero ideada especialmente para nodos (teléfonos) en redes móviles. Se compone de una arquitectura en la que existe una jerarquía de 2 niveles de nodos fijos que ayudan a localizar a los móviles. Unen en grupos las diferentes localizaciones donde es más probable que un nodo móvil se encuentre para poder encontrarlo más eficazmente. Los mensajes a los nodos móviles son emitidos en difusión a los nodos fijos de la primera jerarquía, donde cada uno analizará los grupos para ver si pueden llevar el mensaje al nodo móvil en cuestión. Bakshi et al. [89] también proponen un mecanismo sin costuras para redes móviles, pero aplicando mensajes multicast y empleando técnicas probabilísticas para averiguar a quiénes hay que enviar el mensaje para que llegue al destino. EMI2Lets [90] contiene un modelo de comunicación sin costuras, al que los autores llaman modelo agnóstico del medio de transmisión de datos. Los nodos dentro de esta arquitectura de comunicación pueden comunicarse entre ellos estableciendo conexiones lógicas y abstrayéndose de los detalles de conexiones físicas. Las conexiones físicas manejan el medio de transmisión de datos más adecuado en cada momento dependiendo de las tecnologías disponibles (Wi-Fi, Bluetooth, GPRS, etc.). Si se interrumpe una conexión física, no se interrumpe la lógica, sino que se busca otro medio de transmisión alternativo para continuar, de forma que para los extremos de comunicación sea transparente.

COMPARACIÓN CON OTRAS ARQUITECTURAS

Esta arquitectura está inspirada en el framework Hydrogen [47] en cuanto a que se mantiene una base de almacenamiento local y se mantiene toda la arquitectura dentro del dispositivo, para reforzar el framework frente a desconexiones, pero a su vez se permite que ese permite compartir el contexto entre dispositivos.

Además, la arquitectura completa el framework añadiendo una forma de compartición de contexto que posibilita el razonamiento distribuido, y permite que la arquitectura sea ejecutada en servidores que, siendo parte de la red, pueden ejecutar tareas de razonamiento intensas y compartirlas con el resto de la red. La arquitectura añade también posibilidades de almacenamiento del contexto obtenido.

Recoge ideas de CAMUS [57] en cuanto a que ambas arquitecturas poseen un mecanismo de motores adjuntables, que todo lo que necesitan un componente que envuelva al motor y actúe de interfaz con el framework, lo que da la posibilidad de adjuntar incluso varios motores. CAMUS sin embargo no posee la característica de compartición de contexto que posee esta arquitectura. Comparte con CITRON [58] la idea de tener un espacio distribuido donde esté el contexto. La arquitectura de este trabajo lo hace posible compartiendo los contextos locales en una red Peer-to-Peer.

Las arquitecturas mencionadas anteriormente no poseen técnicas de descubrimiento, elemento que esta arquitectura provee a través del Agente de Descubrimiento, que localiza fuentes de contexto tanto de forma local como a través de la red.

Analizando las características de la arquitectura bajo los criterios con que se han evaluado a otras arquitecturas, obtenemos:

- **Sensorización**: A través de plug-ins de contexto
- **Pre-Proceso**: Capa de pre-proceso
- Procesado del **contexto**: Ontologías. Se procesan a través de un modelo de red Peer-to-Peer donde se comparte la Base de Conocimiento.
- **Almacenamiento**: Red distribuida. Datos almacenados en BD.
- **Histórico**: Es posible añadirlo para dispositivos que lo soporten, el motor de Calidad de Contexto puede añadir las técnicas de agrupación y resumen de contexto que se verán en siguientes apartados.
- **Compartición**: Distribuida.
- **Razonamiento**: Motores de razonamiento semántico.
- **Entrega**: Notificación y consulta-retorno.
- **Descubrimiento**: Distribuido a través de Agentes de Descubrimiento.
- **Seguridad**: La arquitectura debe añadir políticas de control de acceso.

4.2. ¿CÓMO SE ADAPTA EL FRAMEWORK A LAS ÁREAS DE ANÁLISIS?

La arquitectura presentada cubre las áreas presentadas a analizar al comienzo de esta sección:

- Ligereza: El framework dispone de una arquitectura ligera, cuya ejecución en un dispositivo móvil es viable, pero a la vez flexible para adaptarse a distintas potencias de proceso. Sus capas de comunicación y sensorización están basadas en plugins de forma que se mantiene activa en memoria sólo la parte necesaria. Su capa de razonamiento está también formada por motores adjuntables, de forma que dispositivos menos potentes puedan albergar un motor simple y hardware con más capacidad pueda tener varios motores distintos para varias tareas.
- Extensibilidad: El framework es extensible tanto a nivel tecnológico, adjuntando más plugins para nuevo hardware y nuevos motores de diferentes tipos, como a nivel sensorial. Esto último se consigue gracias a su compartición de contexto que posibilita la adquisición de información de contexto remota extraída por otros dispositivos. La capa de sensorización provee a los plugins de conexión a la red en caso de necesitar información exterior.

- Robustez ante desconexiones: La Capa de Comunicación provee de un mecanismo de mantenimiento de conexión que incluso intenta buscar medios alternativos a través de otras tecnologías antes que devolver un error de comunicación. Si lo que se desconecta es un sensor, el mecanismo de descubrimiento intentará reconectar el mismo o bien intentar buscar una nueva fuente de información. En el caso de quedar totalmente fuera de conexión, la arquitectura sigue siendo funcional por estar de forma completa dentro del dispositivo, sin requerir de acceso a servidores centrales para su funcionalidad básica.

- Abstracción del medio de transmisión de datos: La Capa de Conexión provee de esta abstracción a través de plugins de comunicación.

- Capacidad para analizar la Calidad del Contexto: Existe la posibilidad de adjuntar un motor para analizar el grado de Calidad de Contexto.

- Capacidad para tener contexto útil en poco espacio de almacenamiento. El motor de razonamiento para mantenimiento del contexto puede tener técnicas de selección y agrupación de contexto. Sobre este concepto se habla en posteriores apartados.

- Capacidad para razonar sobre incertidumbre y cambios frecuentes de contexto. La capa de razonamiento está ideada para que puedan usarse mecanismos como Redes Bayesianas para el razonamiento, lo cual posibilita la incertidumbre. Motores como SMILE [91], una Red Bayesiana para dispositivos móviles, pueden ser envueltos y añadidos a la arquitectura.

- Capacidad para recoger contexto altamente cambiante: El Agente de Descubrimiento está ideado para reaccionar ante cambios en las fuentes de información de contexto. Reacciona ante cambios de sensores y desconexiones.

- Compartición de contexto: La arquitectura está ideada para formar una red Peer-to-Peer. Existe un Agente de Compartición que se encarga de las tareas de Compartición de contexto a través de los nodos de la red.

5. ASPECTOS DESTACADOS DEL ANÁLISIS

Tras el análisis de arquitecturas y el enunciado de un modelo para dispositivos móviles, varios aspectos relevantes salen a la luz relativos al tratamiento del contexto de una aplicación Context-Aware. Estos aspectos de detallan a continuación.

5.1. MODELADO DEL CONTEXTO

Las aplicaciones Context-Aware tienen como epicentro de sus sistemas toda la información que se refiere al contexto. Sobre ella trabajan, razonan y procesan. Emiten conclusiones y se apoyan en toda la información de sus bases de contexto. Por tanto, no es de extrañar que la forma en que esta información se modela es crucial para el buen funcionamiento de las aplicaciones y frameworks. Una mala elección del sistema de modelado o un mal diseño de los datos pueden llevar a un bajo rendimiento, un mal funcionamiento o una baja escalabilidad.

Por otra parte todos los frameworks vistos utilizan su propia forma de almacenar el contexto, sin llegar a acuerdos o formas comunes de modelado. Todos los formatos son diferentes y eso imposibilita la compartición de datos entre diferentes sistemas. En secciones anteriores se señala la compartición de contexto como un valor clave de cualquier arquitectura context-aware. La diversidad de formatos que cada framework adopta dificulta mucho la tarea de importar, exportar y compartir contexto.

APROXIMACIONES BÁSICAS

Respecto a formatos, Chen et al. enuncia que las formas más utilizadas son [92]:

- Pares **clave-valor**: La información de contexto se resume básicamente en pares de clave-valor, el tipo de dato de contexto actúa de clave y su valor correspondiente actúa como el valor que en ese momento ese tipo de formación adquiere. Esta simple estructura es suficientemente potente como para soportar técnicas de reconocimiento de patrones. Sin embargo, carecen de suficiente sofisticación para poder organizar la información.

- Lenguajes de **etiquetas**: El contexto se representa etiquetado en lenguajes de marcado como XML [40]. Un claro ejemplo son los stick-e notes [36], cuya información de contexto está representada en un lenguaje SGML [37] (luego su representación de contexto evolucionaría hacia ConteXtML[93]). Según XML se fue estableciendo como el estándar de lenguajes de marcado, los frameworks que optaban por este formato de representación lo elegían. Estas etiquetas pueden agruparse y organizarse de manera jerárquica, lo que posibilita una fácil división de tipos de información. Frameworks que necesitan concentrarse en un subconjunto concreto de toda la información en cada momento, pueden utilizar esta aproximación para quedarse con ramas concretas de contenido. Lenguajes como CC/PP [94] o UAProf [95] actúan de base para lenguajes de marcado para contexto. CC/PP (Composite Capability / Preferente Profiles) es un lenguaje de marcado para especificar preferencias y capacidades de dispositivos, su tipo de hardware, servicios que soportan etc. Para que los dispositivos puedan describirse a sí mismos. UAProf es una alternativa similar, usando lenguaje de marcado y definiendo también capacidades, pero centradas principalmente en dispositivos inalámbricos. Ambas alternativas se idearon para que los proveedores de contenido supieran adaptar el formato del mismo a los dispositivos que lo reciben. Otro lenguaje de marcado creado es CSCP (Comprehensive Structured Context Profiles) de Henricksen et al. [96], que permite estructurar la información de contexto pero sin marcar una estructura predefinida. Cualquier información puede colocarse en cualquier punto de la jerarquía y organizarse como cada framework desee, de forma que su interpretación depende de la posición dentro del árbol jerárquico. Permite también colocar etiquetas iguales en diferentes puntos sin generar ambigüedades, ya que siempre dependen de las etiquetas padre que las completan de definir.

- Modelado **Orientado a Objetos**: El contexto se modela como una jerarquía de objetos dentro del paradigma de Orientación a Objetos. El objetivo es aprovechar las ventajas que esto da al modelado, como herencia, encapsulación, reusabilidad y relaciones entre objetos. Una de las principales arquitecturas que

utiliza esta aproximación es Hydrogen [47], que organiza toda la información jerárquicamente en objetos y accede a algunos de ellos directamente para obtener un sub-árbol jerárquico, y de esa manera ofrecer al motor de razonamiento una parte de la información en cada instante. El proyecto GUIDE [97], siendo una guía turística, está focalizado en localización y dispone de un modelo de objetos para definir los aspectos relacionados con el posicionamiento de usuarios. Los estados de atributos dentro de los objetos son utilizados para describir el contexto, y los objetos proveen de métodos para acceder a toda la información, realizar tratamientos de la misma e incluso proveer de información relacionada a través de las conexiones a otros objetos. Esta aproximación puede combinarse con un mapeador de objetos a Bases de Datos relacionales [68] para conseguir almacenamiento y carga automáticos del modelo.

- **Modelado lógico**: Usando el modelo lógico, se expresan hechos que vienen derivados de otra serie de expresiones y hechos, a través de un proceso llamado razonamiento o inferencia. La forma de expresión de estos hechos se formaliza a través de reglas. Concretamente McCarty et al. [98] introducen este concepto de formalización de contexto basado en reglas. Así, se definen expresiones como "*ist(c, p)*", que afirma que la expresión "p" es cierta en el contexto "c". Se pueden definir de esta manera expresiones como *ist(contexto("Historias de Sherlock Holmes"), "Holmes es un detective")*. Esta forma de modelado hace muy sencillo el razonamiento, ya que puede ser introducido directamente en motores de inferencia por reglas, que a su vez producirán nuevas expresiones enunciando cambios de estado o conclusiones. Frameworks como GAIA [45] utilizan esta estructura para almacenar su información de contexto. La introducción de nuevas reglas automáticamente además aumenta la capacidad de razonamiento de los motores de inferencia. Gray y Salber proponen otro modelo [99] llamado "Sensed Context Model", que usa predicados de primer orden para definir el contexto de la aplicación.

MODELADO A TRAVÉS DE ONTOLOGÍAS

Además de estas aproximaciones, Strang y Linnhoff-Popien [100] proponen otra más: **ontologías**. A medida que las aplicaciones Context-Aware evolucionan, se hacen más complejas y amplían su ámbito de acción. Por ello necesitan de mecanismos más completos y flexibles para modelar el contexto. Las ontologías se presentan como un mecanismo prometedor para ofrecer potencia y flexibilidad a los frameworks Context-Aware [101].

Una ontología utilizada en estas aplicaciones es FOAF (Friend of a Friend) [102] que permite modelar personas y sus relaciones entre ellas. Se especifican datos para detallar información sobre personas y se establecen relaciones entre las mismas, de forma que se pueda modelar asimismo el mapa social que rodea a una persona.

SOUPA (Standard Ontology for Ubiquitous and Pervasive Applications) [103], una ontología diseñada para entornos de Computación Ubicua. Comprende dos grandes partes: SOUPA core, vocabulario universal para aplicaciones de Computación Ubicua y SOUPA extensión, para aplicacions específicas. SOUPA core define el vocabulaio necesario para expresar personas, sus intenciones y creencias, sus acciones, su contexto de tiempo y espacio y los eventos que generan, además de las políticas de seguridad relacionadas con toda la información.

Esta parte central actúa de núcleo y puede extenderse para adaptarse a las necesidades de otras aplicaciones. Conscientes de que cada aplicación tiene su propia forma de representación, los autores han optado por dejar un núcleo común pero extensible. Además, se ha creado un ejemplo de extensión para un ámbito concreto de aplicaciones: espacios inteligentes. Así, se ha creado SOUPA extensión.

- Esta extensión define vocabularios para:
- Reuniones y planes
- Documentos
- Captura de imágenes
- Relaciones en el espacio
- Localización

Esta ontología se utiliza en frameworks como SOCAM [48].

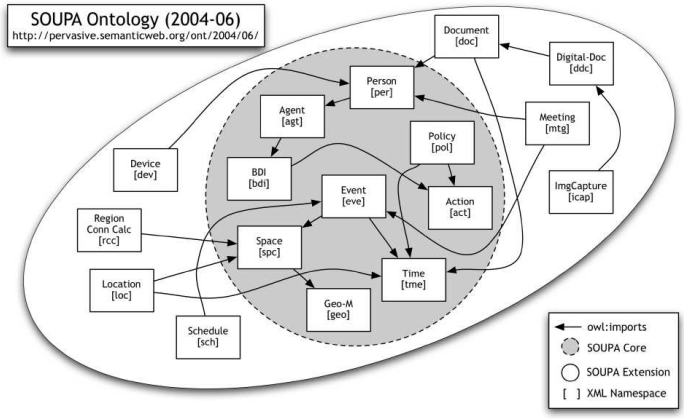

Figura 32. Ontología SOUPA [103]

CONON (Context Ontology), creada por Wang et al. [104] es una ontología utilizada también para el modelado del contexto. Al igual que SOUPA, define un conjunto básico para cualquier aplicación, y deja extenderlo para crear dominios específicos. Según los autores, el vocabulario básico de una aplicación Context-Aware debe comprender entidad Computacional (relacionada con dispositivos y computadores), localización, personas y actividades que realizan dichas personas.

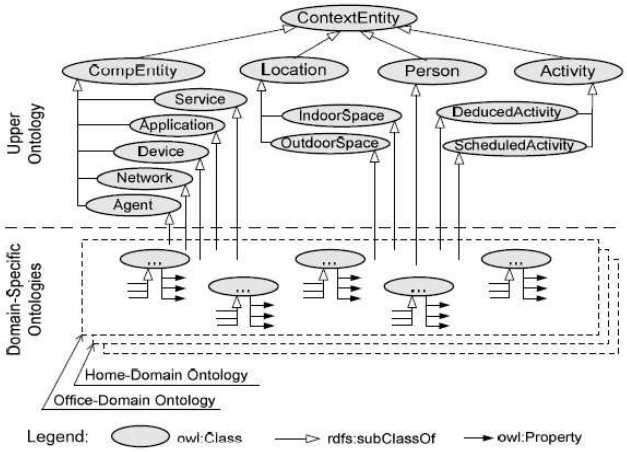

Figura 33. Ontología CONON [104]

El dominio específico se modela creando nuevas entidades que heredan de las básicas. Esta herencia se hace a través de la propiedad "*subClassOf*".

Como puede observarse por los trabajos anteriormente expuestos, las ontologías son un mecanismo flexible que puede sentar las bases para definir subconjuntos básicos de contexto, de forma que pueda llegarse a una estandarización de un núcleo de contexto. Además, si las ontologías están descritas en el mismo lenguaje es posible que diferentes aplicaciones puedan traducir su información de contexto para otras.

En [66] se apunta que los requisitos de las ontologías cuando son utilizadas en Computción ubicua son:

- Simplicidad: Las expresiones han de ser lo más simples posible para facilitar el trabajo a los descriptores del contexto y motores de inferencia semántica.

- Flexibilidad y extensibilidad: La ontología descrita debe ser flexible y proporcionar medios para extenderla.

- Generalidad: La ontología no debe limitarse a un ámbito concreto del contexto, sino que debe soportar un abanico tan amplio como sea posible.

- Expresividad: El modelo debe ser capaz de describir todos los elementos de contexto que se necesiten

EVALUACIÓN

La aproximación de pares clave-valor resulta demasiado simple para aplicaciones de este tipo, no permitiendo una forma estructurada de información ni un medio de organizarla. Tampoco es un formato fácil de traducir para motores de razonamiento ni puede validarse fácilmente.

Los lenguajes de marcado proveen de una buena forma de jerarquizar y estructurar la información. Además, es posible contrastarlos para comprobar su validación sintáctica, disponiendo de herramientas diferentes dependiendo del lenguaje que se utilice para este sentido. Niveles semánticos pueden ser comprobados también d forma parcial mediante herramientas. Sin embargo, no proveen de ningún mecanismo para comprobar ambigüedades. Aunque es difícil mezclar información de diferentes aplicaciones, sí es cierto que aproximaciones como XML pueden proveer de herramientas de traducción desde un lenguaje XML a otro.

Los modelos de orientación a objetos presentan muchas ventajas desde el punto de vista de la compartición y distribución del contexto. Si estos objetos se basan en modelos distribuidos de componentes, resulta muy fácil acceder a ellos abstrayéndose de su localización física o el lenguaje en el que cada uno esté implementado. Como anteriormente se ha comentado también, si este modelo se apoya en sistemas de mapeo a Bases de Datos, se proporciona una forma fácil de almacenamiento. También soportan validación sintáctica a través de la compilación. Sin embargo, modelos orientados a objetos típicamente requieren grandes recursos software y hardware. Elementos que muchas aplicaciones no pueden permitirse en tanta medida.

Los modelos lógicos poseen un alto nivel de formalidad, y pueden distribuirse fácilmente. Sin embargo son muy propensos a ambigüedad e inconsistencias. No tienen sistemas completos de validación y es difícil añadir metadatos a las expresiones. Aunque no dispongan de una forma natural de estructuración de la información, están en un formato ideal para servir de entrada y salida a motores de razonamiento.

Los modelos basados en ontologías, debido a su parecido a los modelos orientados a objetos en cuanto a su definición, presentan una buna forma de compartición. Existen asimismo muchas herramientas para validación de ontologías. Es una de los modelos que más fácilmente admite añadir metadatos y atributos a la información. Tienen además un alto grado de formalización.

En la siguiente tabla-resumen puede verse de forma general una comparativa entre distintas aproximaciones, con valores de 0 a 5:

	VALIDACIÓN	COMPARTICIÓN	METADATOS	ESTRUCTURACIÓN	RECURSOS	RAZONAMIENTO
Clave-Valor	0	1	1	0	5	0
Lenguajes de marcado	3	2	4	5	4	1
Lógica	2	3	2	2	2	5
Orientación a Objetos	4	3	5	5	2	3
Ontologías	4	3	5	5	3	5

Tabla 3. Tabla-resumen de tipos de modelado de contexto

5.2. CALIDAD DEL CONTEXTO

Calidad del Contexto (Quality of Context, QoC) establece un valor de la calidad que tiene la información de contexto. Dar por hecho que la información disponible sobre el entorno es 100% fiable es un error en el que no se debe caer. Por varios motivos, la información almacenada no siempre es adecuada para poder inferir conclusiones. Crear un modelo de contexto teórico resulta fiable, pero a la hora de implantarlo en la realidad muchos factores intervienen para que la información pierda algo de credibilidad o utilidad.

En los entornos reales, pueden existir diferencias entre lo diseñado en la teoría y lo que ocurre en la realidad, una simple diferencia en el tiempo de extracción de datos puede suponer una pérdida de valor en la información. Además, en un entorno real pueden ocurrir fallos impredecibles: sensores físicos pueden fallar, conexiones pueden romperse, la información puede quedar inaccesible, etc. Todo ello hace que la probabilidad de que la información sea fiable vaya decreciendo.

En el ámbito móvil, esta probabilidad de decrecimiento de la calidad se dispara: las conexiones son inestables, los sensores cambian, los alcances de cobertura hacen que se elijan constantemente nuevos sensores o que información desaparezca de pronto. El contexto físico puede cambiar totalmente debido a la posibilidad de moverse físicamente de un espacio a otro. Con un contexto tan rápidamente cambiante las probabilidades de fallo son mucho mayores.

Además de fallos reales de hardware y conexiones, otras causas de pérdida de calidad pueden ser la aparición de inconsistencia de datos debido a varias fuentes que dan la información del mismo tipo y el desconocimiento de cuándo fue la última vez que un sensor notificó su último valor.

Definir una forma de paliar este problema es importante en cualquier aplicación Context-Aware, pero en una aplicación móvil esta importancia se eleva a niveles imperativos. Un perfecto motor de razonamiento puede dar conclusiones disparatadas por la sola razón de basarse en un contexto que no es válido.

La dificultad estriba en definir qué es contexto no válido y cuándo se puede dar. Es necesario dar unos valores cuantificables para explicar y dar sentido al abstracto concepto de “no válido”. Para ello se ha forjado el concepto de Calidad de Contexto. Buchholz et al. [77] lo definen como la información que define la calidad de la información que es usada como contexto. Así, la Calidad de Contexto se refiere a la información en sí y no al proceso o componentes hardware que puedan proveer la información.

Pero una cifra general de “Calidad” no es suficiente tampoco, ya que la calidad es a su vez otro concepto abstracto. Es necesario cuantificar ese valor en otras categorías que puedan servir a una entidad que quiera medir el contexto. Bu et al. [84] argumentan que las principales áreas que miden la calidad del contexto son:

- Tiempo de retardo: Es el tiempo desde que la situación ocurre en el mundo real hasta que se registra y posiblemente guarda dentro del elemento de computación. Es especialmente importante en entornos de tiempo real.

- Probabilidad de corrección del contexto: los sensores físicos no son exactos al 100% y tienen errores de precisión. Almacenando un historial de valores y ejecutando funciones estadísticas se puede llegar a un valor de probabilidad de exactitud de cada sensor.

- Probabilidad de consistencia del contexto: La probabilidad de que no existan inconsistencias en la información de contexto que se maneja.

Los autores argumentan que estos valores están relacionados, ya que información con gran retraso de llegada y demasiado poco actualizada es normalmente incorrecta y puede generar inconsistencias.

Ebling et al. [105] se refieren a este concepto como “Calidad de la Información”, que a efectos de validez de los datos en aplicaciones Context-Aware, puede ser tratado de sinónimo. Para ellos el contexto se mide en factores de:

- Actualización: El grado de actualización de la información, será más grande cuanto menor sea el tiempo en que se actualizó el valor por última vez.

- Confianza: El grado de fiabilidad que ofrece la fuente de información.

Por otra parte, Van Sinderen et al. [106] proponen las siguientes áreas:

- Exactitud: El grado de exactitud y precisión que la fuente de información es capaz de dar acerca del dato. Por ejemplo un sensor de temperatura siempre tendrá un ratio de inexactitud.

- Probabilidad de corrección: Probabilidad de corrección que la fuente de contexto estima que da acerca del dato.

- Confiabilidad: Probabilidad de corrección estimada por el receptor del dato, se contrasta con la probabilidad de corrección.

- Actualización: La edad del dato recibido, normalmente descrita por una marca de tiempo que indica cuándo fue la última vez que se actualizó.

Buchholz et al. [77] definen un espectro de áreas más completo:

- Precisión: Los autores definen precisión como el grado en que el sensor representa la realidad. Por ejemplo un sensor GPS tendrá 10 metros de precisión, mientras que un sistema de localización GSM puede dar valores que como media ronden los 200 metros. El grado de precisión debería ser representado en una escala común o en un porcentaje.

- Probabilidad de corrección: Probabilidad de que un sensor falle y de un valor incorrecto.

- Confiabilidad: Probabilidad que el receptor de la información de contexto estima del sensor dados sus fallos de corrección medios. Un sensor puede dar un grado de corrección del 90%, pero el receptor estimar una confiabilidad del 80% por haber recibido información poco exacta repetidas veces.

- Resolución: La granularidad de la información. Considerando un sensor de temperatura, se pueden recibir datos de la temperatura de una habitación, pero el sensor es incapaz de dar datos de la temperatura de una esquina concreta de la misma, donde por ejemplo pueda haber un calentador.

- Actualización: La edad de la información de contexto. Puede venir por una marca de tiempo, con lo que emisor y receptor deben tener sus relojes sincronizados, o puede ser un contador de tiempo en el receptor.

Como se ha remarcado anteriormente, el factor de calidad del contexto viene dado por la información de calidad en sí, no por el hardware o los procesos que intervienen. Por lo tanto, factores como la calidad del hardware o la calidad del software son independientes a este factor de Calidad del Contexto.

Asimismo, Buchholz et al [77] definen la calidad relacionada con el hardware como Calidad del Dispositivo (Quality of Device, QoD), información acerca de las características y limitaciones técnicas del hardware.

Se define la calidad en el software a nivel de procesos como Calidad del Servicio (Quality of Service, QoS), información que indica cuán bien un servicio desarrolla su función. Este valor está muy ligado al valor de QoD, ya que el servicio verá afectada su calidad si corre bajo hardware con baja QoD.

Gray y Salber [99] enumeran unos aspectos relevantes que definen lo que llaman calidad de la información:

- Cobertura: la cantidad de información que potencialmente se puede dar. Si se habla de información de tiempo, la cobertura sería todos aquellos estados del tiempo que es capaz de reconocer.

- Resolución: La información más pequeña que se puede llegar a percibir. En términos de temperatura una grado de precisión puede ser si se perciben decimales o por el contrario son saltos de uno o varios enteros.

- Exactitud: La probabilidad de dar información incorrecta. Es un factor diferente a no ser capaz de dar la información (resolución).

- Repetibilidad: Estabilidad de la información a lo largo del tiempo. Si el contexto no cambia un nueva consulta al proveedor de información debería dar un valor igual.

- Frecuencia: Cada cuánto tiempo se es capaz de refrescar la información.

- Temporabilidad: El ratio de error debido a retrasos en el tiempo.

Estos valores pueden llegar a ser útiles para llegar a una cifra general de calidad, pero es importante mantenerlos aparte del concepto de Calidad del Contexto, que se refiere a la información en sí misma.

La forma de estimar esta calidad puede diferir en función de quién o qué quiera informarse del valor de calidad de contexto. Un valor promedio de la serie de factores descritos puede ayudar a ciertas entidades que usen el contexto. Para otras sin embargo este valor no tendría ningún sentido y se centrarían en los aspectos concretos. En general, calcular un valor promedio de calidad puede ayudar para elegir una fuente de datos en general, pero en la práctica tanto para realizar un buen razonamiento como para hacer una elección correcta de una fuente de datos, se ha de utilizar cada criterio por separado en función de las necesidades.

Los usos de la Calidad del Contexto son, según Buchholz et al. [77]:

Negociación de Calidad de Contexto: Ciertas aplicaciones pueden requerir mucho nivel de exactitud o en general alta calidad en la información de contexto que utilizan. Especialmente si obtienen los datos de proveedores externos, necesitan tener garantías de que van a poder fiarse de la información que obtienen. Aplicaciones en Tiempo Real necesitan altos niveles de actualización de información, aplicaciones que utilizan extensivamente la actividad de razonamiento pueden ver sus conclusiones distorsionadas por datos de poca fiabilidad o poca exactitud. Aplicaciones de seguridad o monitorización de sistemas críticos requieren alta resolución de las fuentes de datos para enviar informes detallados. En general, usar la calidad del contexto en un proceso de negociación o elección resulta muy beneficioso para todo tipo de aplicaciones. Al igual que en los casos de Calidad del Servicio (QoS), donde se establecen los niveles mínimos requeridos y se procura que la información que se obtiene sea mejor o al menos igual a ellos, se puede negociar el nivel de Calidad de Contexto. Sin acuerdos de este tipo, valores críticos que las aplicaciones necesitan quedan expuestos a la suerte y a posibilidades aleatorias, situación claramente inaceptable para muchos sistemas.

Corrección o estimación de resultados: En el momento de enunciar conclusiones extraídas de información de contexto (dar restaurantes recomendados, concluir que está teniendo lugar una reunión, etc.), si no se dispone de una información de alta calidad, puede ser útil enviar al usuario la probabilidad de acierto o el grado de calidad de la información. De esta manera el usuario puede sacar sus propias conclusiones o dar ciertas correcciones a los datos. De la misma forma que una información sobre el tiempo puede venir adjunta con una probabilidad de acierto, estas conclusiones pueden venir acompañadas de más información que ayude a interpretar mejor los resultados. Por otra parte si el grado de calidad es suficientemente bueno no sería necesario enviar nada al usuario. Si la calidad es buena y las conclusiones no se ven afectadas por ella, los datos de cómo se ha llegado a la conclusión pueden resultar redundantes.

Selección de proveedores de información adecuados: Cuando en los entornos existen numerosos sensores o fuentes de contexto, es probable que varios de ellos sean capaces de ofrecer el mismo tipo de información. Si un captador de información se muestra interesado en unos datos y encuentra varias posibles fuentes que pueden ofrecérselos, lo más probable es que cada uno de ellos los ofrezca con diferentes niveles de calidad. Utilizar el grado de Calidad de Contexto que cada uno de ellos provee es una buena forma de elegir la fuente de datos adecuada. Poniendo como ejemplo un proveedor de estado del tiempo, en un determinado momento pueden encontrarse dos posibles fuentes. La primera la ofrece el propio dispositivo móvil en el que se encuentra la información, con sus sensores propios. La segunda es un proveedor del tiempo a nivel nacional que ofrece su información a través de la red. Podría darse el caso de estimar más oportuna la información del proveedor nacional ya que sus datos, al venir de diversas fuentes y poseer grandes mecanismos de tratamiento de información, son más fiables. El móvil podría dar un nivel muy bajo de fiabilidad de sus sensores por verse afectados por circunstancias locales. Por otra parte puede también existir el caso de que el proveedor nacional esté demasiado desactualizado y no disponga de suficiente precisión. El proveedor local en el

dispositivo podría tener un alto grado de precisión y resolución. Todas estas circunstancias pueden verse reflejadas en la información de Calidad del Contexto y por tanto ser muy valiosas para una decisión final.

Control del razonamiento sobre el contexto: La obtención de contexto de alto nivel a partir del contexto básico, la cribación, refinamiento y filtrado de los datos, la eliminación de inconsistencias o la obtención de conclusiones son tareas propias de una fase de razonamiento. Esta tarea puede verse afectada por información de baja calidad, por lo que puede perder el sentido completamente si se utilizan malos datos de base. Si se quiere obtener la distancia entre los personas, se obtiene la posición de cada una de ellas. Sin embargo el resultado al que se llegue depende en gran medida de la precisión que haya en cada una de las localizaciones. Si una de las personas posee un GPS con alta precisión y la otra sólo puede ser localizada a través de la red móvil que la da cobertura, con una precisión de 200 metros, la información de distancia pierde mucha precisión y eso ha de tenerse en cuenta.

Adaptación de la diseminación del contexto: El factor de Calidad de Contexto puede determinar en gran medida la forma de diseminación y entrega de la información de contexto. Información muy costosa de calcular puede recalcularse cada cierto tiempo y dejarse almacenada para que otros la consulten. Sin embargo esto afecta mucho a su calidad, sobre todo en el concepto de actualización. El dato puede no estar lo suficientemente actualizado. Si en un proceso de negociación de Calidad del Contexto se ha adquirido el compromiso de llegar a un alto grado de actualización, es posible que el proveedor de datos decida cambiar su tiempo de refresco de información aunque cueste más capacidad de cómputo, o que incluso cambie enteramente su política y cree un mecanismo de notificación. Claramente conocer la Calidad de Contexto ofrecida y esperada es un factor de decisión a la hora de diseminar el contexto.

Creación de políticas de privacidad con más detalle: Para controlar factores de privacidad, un proveedor de información puede limitar a ciertos usuarios el acceso a su información a través de políticas de seguridad. Sin información de Calidad de Contexto, sólo se puede limitar a los usuarios el acceso a ciertas partes de la información. Con información acerca de Calidad de Contexto sin embargo, el acceso puede ser de más bajo nivel. Por ejemplo, se puede ofrecer información de localización pero sólo si el grado de precisión es inferior a cierto porcentaje, o si tiene cierta antigüedad. De esta forma la calidad puede ser un factor determinante para decidir cuánto afecta a la privacidad una información.

Además de estos usos que proponen los autores, una buena medición de calidad de contexto ayuda a obtener un **grado de incertidumbre** en el razonamiento. Los motores de razonamiento que soportan grado de incertidumbre pueden beneficiarse de este modelo de calidad para averiguar cuán cierta o segura es una información que están obteniendo. Localizar a una persona en un cierto lugar a través de un mecanismo de localización que informa de una precisión baja e incluso de no demasiada fiabilidad, implica añadir al mecanismo de conclusión un alto grado de incertidumbre al respecto.

TRABAJO REALIZADO EN LA CALIDAD DEL CONTEXTO

Varios autores han llevado el concepto de Calidad de Contexto al campo de frameworks y prototipos, pero no han conseguido dar una formalización general sobre el concepto. Dey et al. [107] se acercan a un concepto de calidad admitiendo que la información que se consigue del contexto puede llegar a ser ambigua, y argumentan que para aplacar esta ambigüedad pueden usarse además de Redes Bayesianas y otros conceptos de Inteligencia Artificial como la mediación [108]. Su concepto ha sido llevado a la práctica, pero únicamente en el campo de interfaces de usuario. Han modificado el framework Context Toolkit [39] para soportar técnicas de Inteligencia Artificial como las mencionadas anteriormente. Al detectar ambigüedad en la información, utilizan un feedback del usuario y dichas técnicas para reducir ambigüedades. Aunque es un esfuerzo que media con la calidad de la información, necesita de la intervención de un usuario, lo cual es inviable en los entornos típicos de Computación Ubicua, donde existen grandes cantidades de información sensorial que además cambia constantemente, más aún en entornos móviles. Es demasiada molestia para el usuario y muchas veces es incluso más información de la que podría manejar.

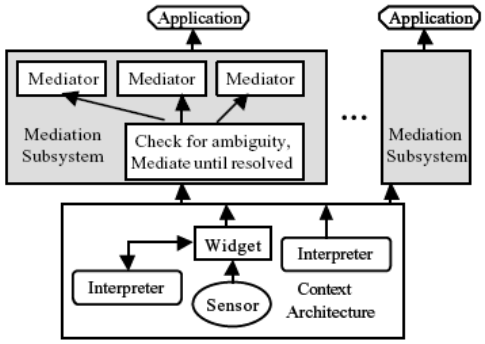

Figura 34. Context Toolkit modificado por Dey et al. [107]

Castro et al. [109] se centran en medir la calidad de la localización como información de contexto. Relacionan la calidad con la exactitud de la fuente y la confiabilidad en ella. Aunque estos valores son algo limitados, sí que han realizado una implementación utilizando Redes Bayesianas. Su aproximación es consultar un punto de acceso de localización hasta encontrar una información con el nivel requerido de lo que llaman Calidad de Información (o quedarse sin puntos de acceso disponibles). Para medir la exactitud usan un modelo heurístico de datos offline con el que han entrenado a la red. Esta aproximación, aunque ha sido llevada a la práctica y puede ser generalizada para otros tipos de información, realmente tiene en cuenta pocos aspectos de calidad, además de centrar todo el trabajo en el receptor de información, quien tiene que usar Redes Bayesianas e incluso entrenar con datos de cada tipo de sensor.

Henricksen et al. [96] proponen un modelo expresado formalmente que contiene atributos de calidad en las relaciones entre entidades. Se trata de un modelo para su dominio de trabajo (personas, dispositivos y canales) y se modela una manera formal, en forma de campos que definan calidad, añadiéndose los atributos que se estimen oportunos para cada relación. Es una aproximación para poder medir ciertos datos que indiquen calidad, expresados formalmente.

En este modelo, los atributos se definen primero con un campo que indica el tipo de atributo de que se trata, para después definir un sub-tipo que, marca la métrica con la cual se mide ese atributo.

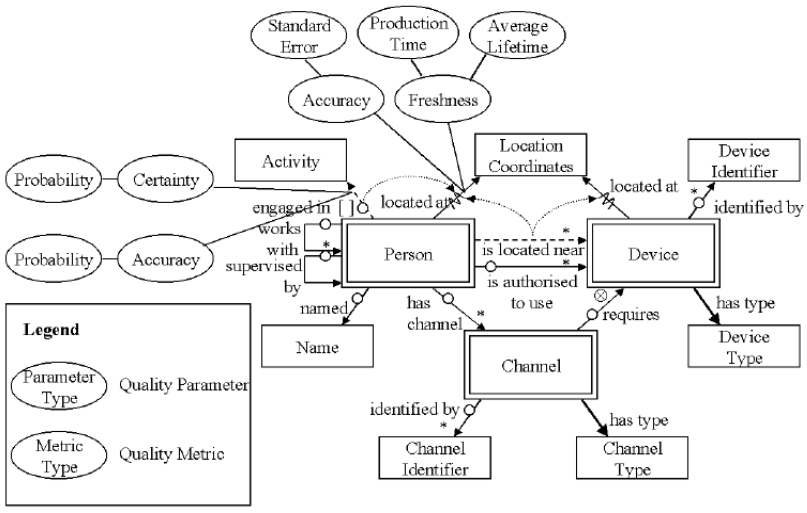

Figura 35. Modelo formal de atributos de calidad de Henricksen et al. [96]

Gu et al. [49] proponen en su arquitectura SOCAM una ontología para Calidad de Contexto, de forma similar a la propuesta por Henricksen et al. [96]. Definen tipos de atributos y métricas de los mismos. A partir de estas definiciones pueden incluirse parámetros de calidad en el razonamiento.

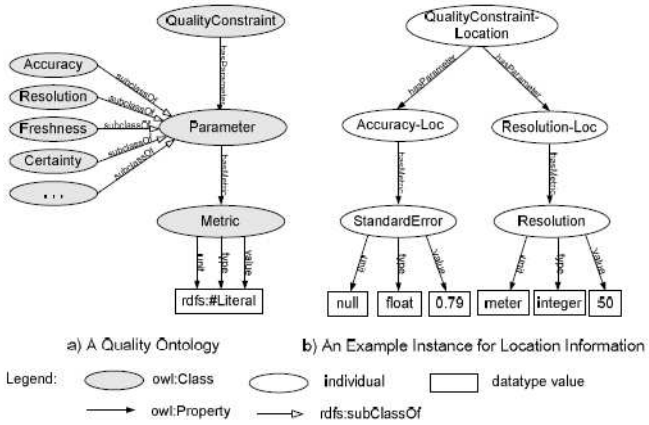

Figura 36. Ontología para modelar Calidad de Contexto de Gu et al. [49]

Tang et al. [110] proponen otra ontología para modelar calidad de contexto, yendo además más allá, proponen un mecanismo para utilizarla u tratar así la información con Calidad de Contexto. La ontología posee la clase Calidad, relacionada con la clase Parámetros (parámetros de calidad). La lista de parámetros es primariamente la identificada por Buchholz et al [77] (corrección, precisión, actualización, confiabilidad y resolución). Cualquier propiedad está relacionada con la clase Calidad si desea tener atributos que definan su Calidad de Contexto.

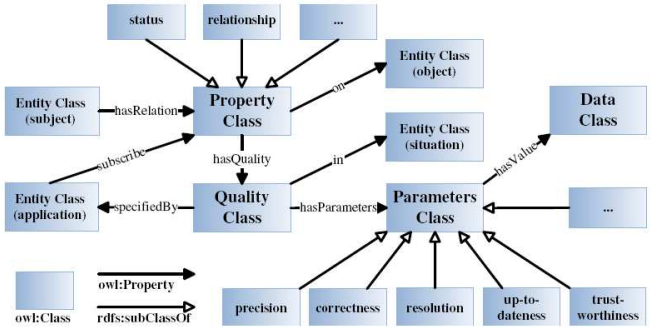

Figura 37. Ontología para modelar Calidad de Contexto propuesta por Tang et al. [110]

Basándose en esta ontología, los autores presentan un mecanismo de rectificación en caso de detectar una baja calidad. Para empezar, definen en la cadena de valor del contexto tres actores principales:

- Proveedor de Contexto: El que obtiene los datos crudos de contexto
- Servicio de Información de Contexto: Media en el proceso de obtención de contexto. Utiliza los proveedores y mezcla, agrega y razona para obtener contexto de alto nivel
- Servicio Context-Aware: Ofrece servicios a los usuarios. Se suscribe a los Servicios de Información de Contexto para obtener la información de contexto que necesita. También puede devolver un feedback de la información que ha obtenido.

Los proveedores ofrecen información, en la que en principio los servicios de información confían. Esta información es entregada a los Servicios Context Aware, quienes pueden devolver un feedback acerca del mismo. El feedback suele significar que la información no es tan buena como se pensaba que era. Bien porque el Servicio Context Aware está suscrito a más servicios de información, porque tiene su propio razonamiento o simplemente porque tiene una visión más general del dominio de su aplicación, puede devolver una respuesta. Dependiendo del tipo de respuesta el servicio de información actuará en consecuencia. Una respuesta del tipo "Incorrecto" hace bajar el grado de corrección. "Inadecuado" hará bajar el grado de resolución de la información. Ajustando los valores de calidad, el servicio de información puede llegar a tomar la decisión de recoger la información de otro proveedor que tenga disponible.

Esta técnica trabaja ante todo en el campo de "**Selección de proveedores de contexto adecuados**". Si bien no trata todas las áreas posibles, propone técnicas para llevar a cabo las tareas del campo concreto, no quedándose sólo en el enunciado de una ontología.

Proponiendo una estructura de proveedores y servicios de información similar a la enunciada por Tang et al. [110] y centrándose en el mismo campo de selección de proveedores, Krause y Hochstatter [111] proponen un mecanismo de selección basado en reputación. El usuario debería fijarse en el grado de reputación que ofrece el servicio de contexto. Este valor de reputación es sin embargo guardado por las aplicaciones y está relacionado con la ausencia de contradicciones con respecto a otros servicios que la aplicación disponga, y la consistencia de la información. Un servicio de información que se comporta correctamente gana credibilidad y es por tanto elegido frente a otros que no gocen de tanta. Esa aproximación sin embargo deja actuar a cada proveedor a su libre albedrío sin proveerle de información para que se auto corrija. Además pasa el problema de cálculo de calidad a cada aplicación, ignorando las experiencias de otras aplicaciones.

Preuveneers y Berbers [112] también añaden atributos relativos a Calidad de Contexto a una ontología de modelo de contexto. Su campo de acción es asimismo la selección de proveedores de contexto. Utilizan técnicas de Modelo de Espacio Vectorial para identificar los proveedores irrelevantes. Además de los valores expuestos por Tang et al. [110] para medir calidad, enuncian unos valores que llaman subjetivos, porque son las aplicaciones o entidades de más alto nivel quienes los definen por su propia experiencia con el sensor:

- **Objetividad**: Cuán exacto es el sensor, comparando el grado de exactitud que el mismo sensor afirma, con el valor que da el resto de sensores similares.

- **Disponibilidad**: El grado en el que el sensor está disponible cada ver que se requiere su información. Sobre todo en entornos móviles, los sensores pueden entrar y salir del radio de cobertura muy a menudo.

- **Completitud**: El grado en el que la información viene acompañada de suficientes parámetros de calidad.

Si los proveedores tienen un nivel mínimo de exactitud deberían emitir resultados muy parecidos, así que se detectan los valores que se salen del promedio para marcarlos como incorrectos, utilizando el test iterativo de Grubb [113]. Si se detecta un valor pico fuera de la media, el valor de objetividad se reduce. Si no viene la información acompañada de suficientes metadatos de calidad para la aplicación, se reduce su completitud. Si el sensor no ha podido enviar la información, se reduce su disponibilidad.

Esta aproximación adolece de la misma desventaja que la aproximación de Krause y Hochstatter [111] en cuanto a que se pasa a cada aplicación gran parte del peso de realizar la estimación de calidad. Sin embargo, si se desea el punto de vista subjetivo de cada aplicación es necesario que parte del proceso esté en los servicios usuarios del contexto.

Van Sinderen et al. [106] incluyen en su arquitectura aspectos de Calidad de Contexto: Exactitud, probabilidad de corrección, fiabilidad y grado de actualización. Aunque lo añaden en forma de metadatos a la información del contexto, no proponen ningún mecanismo formal para definir Calidad de Contexto ni ningún medio específico para utilizarla. Los autores se limitan a añadir atributos de metadatos a la información y trabajar con ellos en las reglas del motor de razonamiento de una forma totalmente personalizada. Además, el valor de calidad es sacado del razonamiento, no teniendo en cuenta valores que las propias fuentes de contexto de bajo nivel pueden dar de sí mismas, su exactitud, precisión etc.

De una forma similar, Bu et al. [84] añaden en su arquitectura el valor de corrección de un dato, pero como en la arquitectura anterior, este valor es sacado del razonamiento ignorando valores de las propias fuentes de datos. Ambos modelos requieren de entrenamiento sobre tipos de sensores para extraer información que muchas veces los propios sensores podrían dar. Los ámbitos de calidad están además limitados a un par de características.

El Context Managing Framework [44] añade algunos atributos a la información que puede obtener del contexto. No obtiene la noción de Calidad de Contexto o similar, pero la obtención de contexto de alto nivel se beneficia de atributos de metadatos como Confianza y marcas de tiempo. El valor de confianza es un valor único en forma de probabilidad. El uso mayoritario de este atributo es para añadir incertidumbre al razonamiento.

Todas las aproximaciones vistas intentan acercarse de una manera u otra al tratamiento de la calidad del contexto. La mayoría se limitan a completar una ontología o incluir algunos parámetros de metadatos, mientras que otras ofrecen técnicas y aproximaciones para tratar estos parámetros de calidad y poner en funcionamiento los modelos propuestos, si bien los campos de aplicación son aún muy limitados. Las técnicas vistas no cubren todos los campos de aplicación de Calidad del Contexto, limitándose la mayoría de ellas a la selección de proveedores de información.

5.3. SELECCIÓN Y FILTRADO DEL CONTEXTO

Para llegar un poco más cerca del concepto Computación Ubicua, muchos frameworks se han creado para facilitar la creación de aplicaciones. La mayoría de ellos incluyen de alguna u otra forma el razonamiento sobre el contexto, para una o varias tareas. Sin embargo, hasta ahora se ha hecho la asunción de que todo el contexto que se obtiene y toda conclusión a la que se llega, se basa en información de contexto que siempre está disponible, cuando realmente no es así en la realidad.

Por tratarse de información que depende de muchos sensores desplegados por el entorno, pueden darse desconexiones o fallos de hardware que hagan la información no disponible. Por otra parte, valores muy bajos de Calidad de Contexto, pueden hacer que el dato que se ha obtenido sea inutilizable. Cuando estas circunstancias no se tienen en cuenta, los razonadores seleccionan de forma demasiado confiada información que no está en buen estado y que potencialmente puede llevarles a conclusiones erróneas o degradadas.

Para evitar gran parte de este problema, es posible identificar qué valores de contexto son más relevantes que otros para enunciar ciertas conclusiones. Esto es llamado Selección de Contexto [65].

Esta selección no está relacionada con la selección de proveedores de contexto adecuados que se mencionaba en el apartado de Calidad de Contexto. Aunque se trate en ambos casos de selección y esté relacionada con información de contexto, este tipo de selección no afecta a la fuente de proveedores desde donde se obtiene el contexto básico, sino que afecta a los razonadores seleccionando de entre el contexto actualmente disponible en el repositorio, el más relevante para llegar a conclusiones.

Si llegado un momento en que se detecta información inaccesible, en mal estado o de baja calidad, habiendo realizado esta selección de contexto, dicha información puede desecharse por tratarse de muy poco relevante, o en cambio parar todo el proceso de razonamiento debido a que es vital. También sería posible en este segundo caso llegar a conclusiones pero bajando su nivel de calidad o informando al usuario de los valores escasos sobre los que se ha trabajado.

Como se puede apreciar, esta técnica de Selección de Contexto está muy alineada con el concepto de Calidad del Contexto, pudiendo combinarse entre ellas para dar lugar a un razonamiento de mejor calidad. Un contexto de baja calidad será uno de los factores primordiales para comprobar si realmente es relevante y necesario mantener ese tipo de información en el repositorio.

Además, esta selección de contexto tiene otras muchas ventajas [22]:

- **Ahorro de sensores**: Al seleccionar sólo la información relevante, se pueden descartar los sensores que se consideren irrelevantes para las conclusiones. Ahorrando así el despliegue de los mismos.

- **Ahorro de espacio de almacenamiento**: Eliminar la información no relevante reduce las necesidades de espacio de almacenamiento. Esta es una gran ventaja en dispositivos que tienen escasos recursos de almacenamiento.

- **Ahorro de trabajo computacional**: Eliminar información irrelevante también ahorra tiempo de proceso, ya que los razonadores dispondrán de un conjunto menor de información que manejar. Paradójicamente sus conclusiones alcanzarán un mayor grado de exactitud.

- **Reducción de incertidumbre de las conclusiones**: Mucha información descartada por irrelevante podría venir adjunta de información que añada aún más incertidumbre a las conclusiones (por ejemplo, una baja calidad en exactitud). Eliminando esta información se reduce el grado de incertidumbre.

En [22] a esta relación entre el contexto y las conclusiones la llaman “Relación de Contexto”, y define qué grado de relación tiene cada atributo de contexto con respecto a cada conclusión que se obtiene, lo que marca el grado de relevancia.

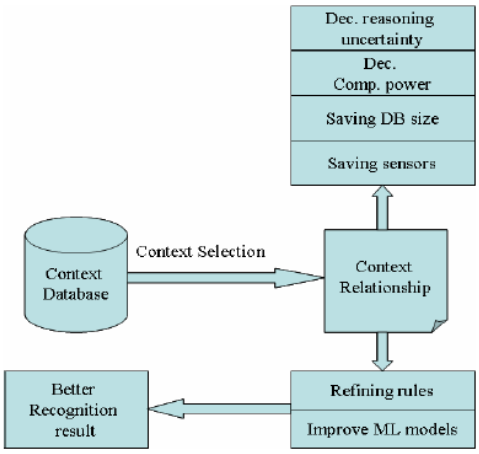

Figura 38. Concepto de Relación de Contexto de [22]

Muchas veces el propio usuario conoce qué factores de contexto son relevantes y por tanto él mismo puede seleccionarlos. Sin embargo esto se da únicamente en los casos más sencillos. En dominios más complicados como concluir el estado de ánimo de una persona, donde numerosos factores entran en juego, esta tarea es muy difícil de realizar manualmente. Por ello, es necesario llegar a técnicas más automáticas.

En [65] se llega a una aproximación mediante selección de características [114]. Como se muestra en la figura, la selección de características se usa para cribar la información de contexto y reducir su espacio de almacenamiento, dejando el contexto más relevante. Con él, se utilizan las técnicas de Machine Learning adecuadas para obtener el modelo de conocimiento, que por provenir de contexto refinado ha costado menos esfuerzo procesar y conseguir, y es más exacto [65]. Con este conjunto de contexto refinado, puede compararse el conjunto de reglas hechas por el usuario y hacerse un proceso de refinamiento, por el que las reglas quedan reducidas a aquellas que realmente no dependen crucialmente de contexto irrelevante. Con todo este resultado se llega a un modelo de conocimiento mucho más eficiente.

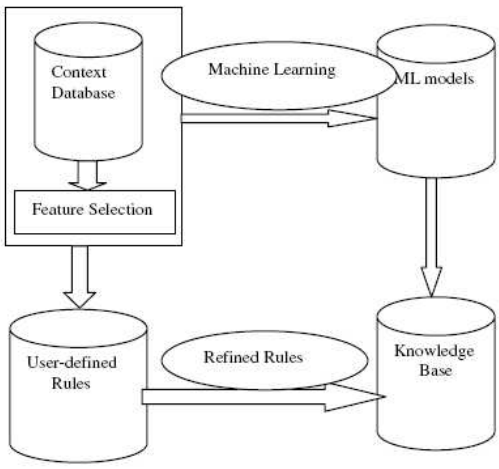

Figura 39. Proceso de refinamiento de contexto [65]

En [22] la selección del contexto se hace mediante el uso de la ganancia de información [115] en cada atributo de información de contexto que se almacena, tal y como propone Quinlan [116]. Este valor se utiliza para definir la relevancia de cada atributo y poder así realizar la selección de contexto relevante. Después utilizan Redes Bayesianas de Retropropagación (BP) [117] como algoritmo de Machine Learning para obtener el Modelo de Conocimiento.

Strang et al. [118] consideran de importancia la relevancia del contexto para una tarea, no para el sistema completo. Definen un modelo de contexto en el que se tiene en cuenta este aspecto, especificando condiciones de relevancia. De esta forma se obtiene un filtrado externo del contexto, es decir, no se filtra tras almacenarse, sino que se filtra en el momento de recuperarse para comenzar a razonar sobre él o realizar tareas. Se tienen en cuenta ciertos valores de relevancia para directamente filtrar a quién se utiliza para obtener contexto y almacenarlo.

Según los autores, estas características de relevancia se dividen en dos tipos:

- Relevancia fuera de la fuente del contexto: Perteneciente a parámetros más dentro de la lógica de negocio y del ámbito de la propia aplicación.

- Relevancia dentro de la fuente del contexto: Perteneciente al ámbito de contexto, caen bajo esta categoría aspectos como obtener los valores de localización de otras entidades donde el valor sea cerca.

Rasheed et al. [119] proponen técnicas de resumen de contexto, tanto para eliminar espacio como valores que no son relevantes para el sistema. Los detalles que no son relevantes se diluyen y pasan a formar parte de un contexto resumido, más manejable y fácil de procesar. Esta información se compacta y se pasa por procesos de agregación para lograr información relevante, sin pérdida del significado principal y ocupando menos espacio. La técnica de resumen no es en tiempo real, puesto que se basa en un repositorio de contexto que previamente se rellena con la información recogida. Los ejemplos más clásicos son agrupar datos de temperatura en datos medios situados en franjas horarias y fechas, en lugar de muestras tomadas cada pocos minutos.

La pérdida de datos singulares provoca una pérdida de exactitud, que se procura sea mínima. Puede complementarse con datos de Calidad de Contexto aportando fiabilidad, exactitud, etc. Para implementar esto, los autores han modificado el framework CAMUS [57], aunque no han eliminado el repositorio original, duplicándolos. El original se conserva para aquellas consultas que aún siendo muy poco frecuentes, requieran 100% fiabilidad.

Las diferentes técnicas de selección y filtrado de contexto son muy útiles y de especial importancia en el campo de dispositivos móviles, donde mucha cantidad de información y potencia de proceso puede ahorrarse, haciendo que sistemas difíciles de llevar a terrenos móviles puedan beneficiarse de los resultados y conseguir entrar en este tipo de dispositivos.

En muchos casos el trabajo extra que requiere el filtrado puede resultar un inconveniente aún mayor de sobrecarga, sin embargo en otras aproximaciones la mayoría del trabajo pesado se hace en el entrenamiento y obtención de un modelo de conocimiento a través de machina learning. Este proceso puede ser incluso hecho externamente, haciendo que los dispositivos móviles aprovechen las ventajas que aporta y no se vean excesivamente afectados por los inconvenientes.

6. REALITY MINING

Este concepto de Reality Mining se ha acuñado recientemente por Sandy Pentland en el MIT (Massachussets Institute of Technology) [120] para indicar una iniciativa en la que se analicen datos percibidos del entorno para averiguar patrones sociales. En realidad, los datos involucrados, son todos aquellos que se refieren al entorno de usuarios. Los datos que rodean a las personas en su día a día son la base para el análisis, y el MIT los acuña como "realidad". En términos de sensores, los datos que se analizan conforman la realidad de un usuario en su día a día. Por otra parte esta cantidad de datos es inmensa, impensable de manejar por seres humanos, por lo que se hace necesario el proceso de esta ingente cantidad de información a través de máquinas. El término Reality Mining viene derivado de hacer Minería de Datos [121] a los datos que conforman la realidad de un usuario día tras día.

Hasta la fecha, el dispositivo que mejor se adecua y puede adaptarse al concepto de Reality Mining es un teléfono móvil. Los móviles están provistos de todo tipo de sensores integrados y son capaces de actuar como un verdadero grupo de sensores embebido en un hardware de poco tamaño. Además, si no disponen de uno, indudablemente poseen las características de comunicación necesarias para alcanzarlo y comunicarse con él. Todo esto ya se ha visto en anteriores apartados del trabajo, pero quizá lo más importante de un teléfono en el terreno de Reality Mining es su propiedad para acompañar a un usuario durante todo el día y recoger las acciones que hace. Cada vez que usamos el teléfono, dejamos bits de información registrados en él: el propio teléfono contacta con las estaciones de cobertura más cercanas, revelando su localización. Incluso puede venir provisto de GPS. Tanto el proveedor como el teléfono recogen la información de las llamadas, tanto de su duración como del número a quien se llama o desde el que viene la llamada.

Con más tipos de información, pueden averiguarse hechos y patrones de comportamiento muy interesantes y que pueden beneficiar a todos los usuarios de teléfono, desde ayudar a la dinámica diaria del lugar de trabajo hasta la identificación de epidemias [122]. Es posible averiguar un modelo del entorno social de un usuario registrando información sobre sus llamadas, sus contactos, su localización y la proximidad con otras personas (otros teléfonos de otras personas)

En [122] también se habla de otras posibilidades, como averiguar patrones de contagio de epidemias. La mayoría de modelos epidemiológicos no tiene en cuenta cuándo y con quién ha estado una persona a lo largo del día. También es posible recurrir a sensores de teléfonos móviles para, a través de micrófonos y sensores avanzados, analizar cuándo una persona está deprimida.

Según el Laboratorio de Reality Mining del MIT [123], los usos generales de esta técnica pueden caer bajo las siguientes categorías:

- ¿Cómo evolucionan las redes sociales a lo largo del tiempo?
- ¿Cómo de predecibles son las vidas de los seres humanos?
- ¿Cómo fluye la información?
- ¿Cómo pueden cambiarse las interacciones entre personas dentro de un grupo para mejorar su comunicación y funcionamiento?
- ¿Cómo puede obtenerse un modelo de comportamiento de los seres humanos?
- ¿Cómo pueden analizarse las conexiones sociales y las relaciones de los individuos entre sí?

El trabajo de Sandy Pentland también ha dado como resultado un valioso repositorio de información [124] en el que se ha grabado información de uso de teléfonos móviles de 100 estudiantes del MIT en un total de 350.000 horas (unos 40 años aproximadamente) de continuo datos continuos sobre el comportamiento humano. Estos datos también han quedado disponibles online para la comunidad científica [125].

6.1. APLICACIONES Y TRABAJO REALIZADO EN REALITY MINING

El término es reciente, pero el trabajo realizado viene de unos años atrás. La minería de datos es algo que ha estado activo mucho tiempo sobre todo para campos de Marketing y análisis de mercado, pero la tendencia de llevarlo a campos personales también es reciente. Algunos proyectos han tenido en cuenta este análisis de datos y han producido prototipos interesantes.

AnyDiary [81] es una aplicación que pretende registrar la actividad de un usuario en forma de cómic y dibujo animado. Para ello, utiliza aplicaciones que registran en logs todo lo que hace un usuario y tras esto las analiza con Redes Bayesianas para crear modelos de conocimiento. Con estos modelos, junto con información de lugar y demás contexto, los autores pretenden crear en un futuro un sistema que reconozca el estado de ánimo, y la actividad que se está realizando en cada momento y la gente relacionada. Esa información se crea y muestra a través de viñetas de cómic.

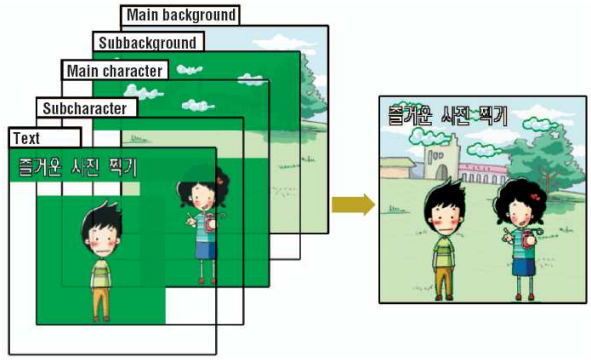

Figura 40. Composición de viñetas en AniDiary [81]

Para recoger esta información de log, han utilizado estudiantes que recogían a través de su móvil información de lo que hacían durante cada día.

Otro trabajo [126] realizado en el MIT analiza las conversaciones telefónicas, pasando por un proceso de detección de conversación y otro de análisis de la misma, en la que se analizan las señales de audio para averiguar emociones y estados de ánimo. Uno de los usos que se le da es analizar si se está llamando a un amigo o a un desconocido, y así marcar a través de llamadas la red social de un individuo. Este análisis se utiliza también para medir el índice de interés de los participantes.

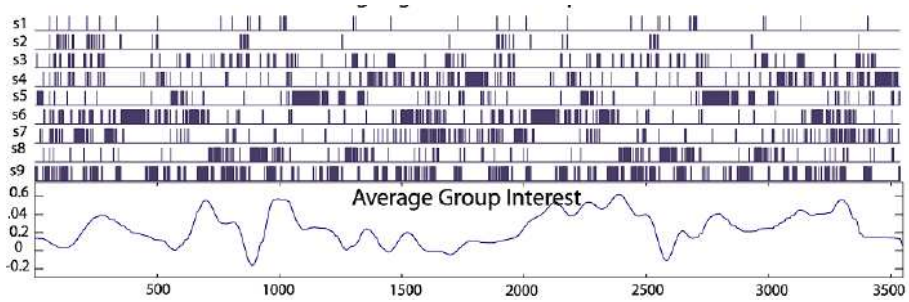

Figura 41. Análisis del audio para medir el índice de interés [126]

En otra fase, se pretende analizar el contexto de la conversación, invirtiendo técnicas de voz a texto para analizar los comentarios de las llamadas e inferir la temática y contexto general.

Según los autores, este sistema puede tener las siguientes aplicaciones:

- Analizador de reuniones: En el que se mide el interés y feedback de los asistentes.

- Métricas de opinión: Analizar los comentarios que dan lugar a opiniones diversas y la controversia que generan.

- Mapear grafo social: Analizando llamadas, a quién se hacen y el nivel de interés / desenfado, se puede inferir un grafo social de cada individuo.

Otra aplicación es Serendipity [127], que analiza los patrones de cercanía entre personas para analizar compatibilidades y gente que no se conoce pero se debería conocer. Analizando a través de Bluetooth [75] quién y en qué momento del día está en cada lugar. Se despliegan balizas Bluetooth que recogen qué teléfonos están cerca de ellas y envían la información a servidores que analizan los datos.

Estos datos son después contrastados para analizar varias características de cada usuario: dónde suele pasar ciertos momentos del día, qué personas hay junto a él y cómo se relacionan con él. El tiempo de permanencia juntos los autores lo toman como un factor crucial, así como el lugar en el que se da la situación. Información de cercanía alrededor de una máquina de café no tiene el mismo significado que información de cercanía en una mesa de cafetería.

Con hechos como estos, Serendipity calcula las relaciones interpersonales y la gente que podría o debería conocerse entre ella.

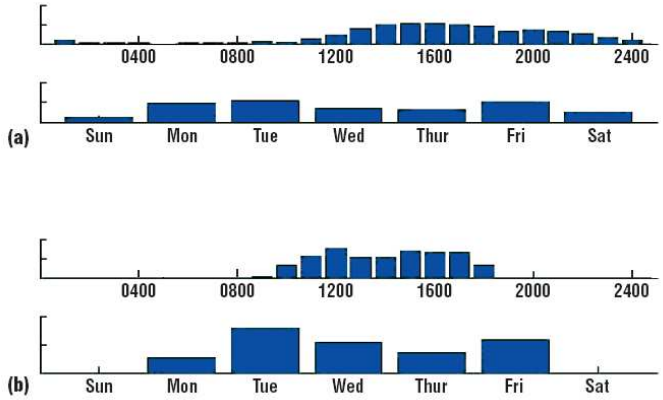

Figura 42. Análisis realizado a un individuo en Serendipity [127] acerca de la cercanía con un amigo (a) y un compañero de oficina (b)

Esta aplicación tiene según los autores tres usos principales:

- **Empresa**: Analizar las relaciones sociales dentro de una empresa para, entre otros, poder averiguar la comunicación entre las personas de una organización y poder situar cerca de las que más necesidad tienen de comunicarse entre ellas.

- **Citas**: Analizar compatibilidades entre diferentes personas para poner en común gente que podría ser sentimentalmente compatible.

- **Conferencias**: Analizar intereses similares entre asistentes a conferencias para poner en común gente con los mismos objetivos de investigación y que podrían colaborar juntos en proyectos.

Miklas et al. [128] proponen un simulador de red Social móvil que mide encuentros entre personas. El simulador mide aspectos clave tanto en cuestión de tiempo como de relaciones sociales, como encuentros con amigos, encuentros con extraños y cómo estos tipos de encuentros varían según la hora del día y el día de la semana. Contrastan los encuentros entre amigos y extraños con modelos de conocimiento para el análisis. Los modelos de conocimiento son diferentes dependiendo de si son extraños o no. Los autores afirman que los patrones de comportamiento para con gente conocida y extraña son muy diferentes y justifican así la utilización de dos modelos.

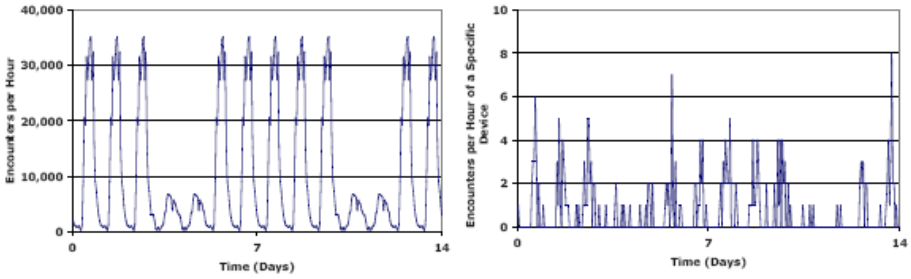

Figura 43. Análisis de encuentros en el simulador de Miklas et al. [128]

Congleton y Nainwal [129] analizan los datos de Reality Mining obtenidos por el MIT para obtener grafos sociales derivados de las llamadas telefónicas que se realizan entre estudiantes de dicho instituto americano. Los grafos se obtienen analizando la ubicación de los estudiantes que llamaban a otros estudiantes participantes del experimento cercanos. La localización se analiza a través del CellID.

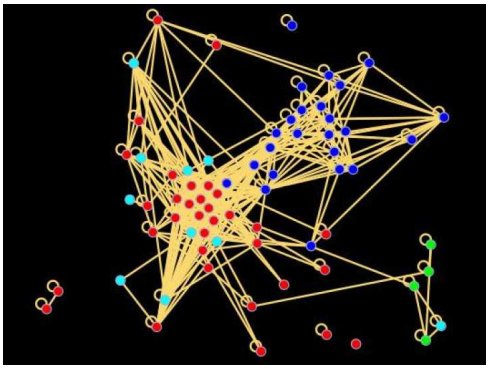

Figura 44. Grafo de conexiones entre estudiantes. Uno de los resultados del estudio de Congleton y Nainwal [129]

Wu et al. [130] analizan las relaciones cara a cara de trabajadores en un entorno de oficina y las comparan con las relaciones a través de otros medios como email. Para ello, proveen a 70 trabajadores de una oficina con un dispositivo que deben llevar consigo. Este dispositivo es llamado “Sociometric Badge” y dispone de sensores de infrarrojos para detectar presencia de encuentros cara a cara, sensores gíreles para detectar proximidad, acelerómetro para captar movimiento y micrófono para captar patrones de habla. Toda la información captada se lleva a servidores centrales que procesan la información.

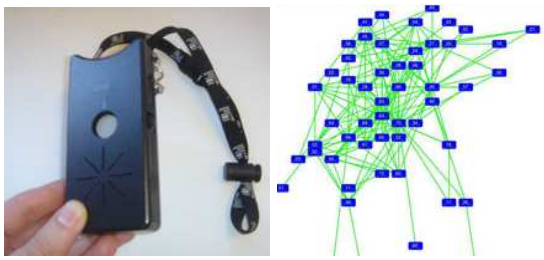

Figura 45. Sociometric Badge y el grafo social que genera en el proyecto de Wu et al. [130]

Tras el análisis en los servidores, se generó un grafo de relaciones sociales de los participantes que se comparó con otro grafo disponible de los mismos participantes, pero utilizando e-mail y otras formas de comunicación electrónica. Los autores llegaron a la conclusión de que el grafo de las relaciones cara a cara ofrece más cohesión por expresar personas con las que realmente ha habido un encuentro. El correo electrónico tiene por qué implicar relación en todos los casos. Los dos grafos resultaron bastante diferentes. Los autores argumentan también la conclusión de que no se deben pasar por alto los diferentes tipos de relaciones y se ha de conocer que grado de cohesión interpersonal tiene cada uno de los medios de comunicación y relación.

6.2. IMPLICACIONES EN PRIVACIDAD

Estas técnicas de análisis de datos implican trabajar con datos personales muy sensibles. Si no se trata con cuidado el asunto de la privacidad, una de las primeras reacciones que generará el Reality Mining será rechazo social, ya que a nadie le gusta averiguar que está siendo constantemente evaluado y analizado, y que muchos de sus datos personales quedan registrados.

Además de ello, existen también implicaciones legales que prohíben el almacenamiento de ciertos tipos de datos u obligan al mantenimiento de políticas de datos para almacenar ciertas informaciones en sistemas informáticos. Por todo esto, el Reality Mining ha de ser escrupuloso en el tratamiento de la privacidad de los usuarios.

- La primera cuestión que se ha de cubrir, ha de ser la transferencia de información. Los datos para analizarse o almacenarse se han de transferir a otros computadores, normalmente servidores con más capacidad de almacenamiento y proceso. Esta transferencia implica utilizar técnicas de cifrado de información para asegurarse de que nadie no autorizado tendrá acceso a los datos ni podrá cambiarlos.

- Bajo los términos de almacenamiento y tratamiento, se hace necesario definir el marco legal y ético de este tratamiento de información.

- Otra cuestión importante es el análisis de información personal dentro de los teléfonos móviles (u otros dispositivos) que debe quedarse en ámbito local, por lo que no puede siquiera transmitirse a otros. Es necesario definir políticas que determinen qué datos pueden transmitirse libremente, qué datos pueden transmitirse bajo adecuadas políticas de seguridad y qué daos no pueden exponerse a otros. Estos últimos pueden servir para obtener otras conclusiones que puedan transferirse.

- Una vez en los servidores se almacenan los datos, sigue habiendo gran cantidad de información sensible. Para muchas entidades esta información es necesaria, y si está autorizada podría acceder a ellos. Sin embargo otras entidades no deberían llegar a ese nivel de detalle, por lo que se hace necesario hacer una diferenciación de este segundo tipo, a la que pueden pasársele datos agregados y anonimizados.

- El usuario ha de ser siempre consciente de qué tipo de datos se está analizando y publicando.

- El usuario ha de ser también capaz de controlar el grado de privacidad que desea en cada momento y qué datos pueden y no pueden utilizarse en base a ello.

Por todas estas razones se han de cuidar con especial de atención las políticas de acceso y seguridad de la información en los dispositivos móviles. Este aspecto concreto de privacidad está abierto a futura investigación, ya que no se han llegado a análisis ni conclusiones concretas de este campo.

7. REFERENCIAS

1. Weiser, M., The Computer of the Twenty-First Century. Scientific American, 1991. 265(3): p. 91-104.

2. Abowd, G.M., ED., Charting past, present, and future research in ubiquitous computing. ACM Trans. Comput.-Hum. Interact., 2000. 7(1): p. 29-58.

3. Milner, R., Ubiquitous Computing: Shall we Understand It? Comput. J., 2006. 49(4): p. 383-389.

4. Eungyeong, K., Hyogun, Y., Malrey, L., Gatton, T. M., The User Preference Learning for Multi-agent Based on Neural Network in Ubiquitous Computing Environment.

5. Ley, D., Ubiquitous Computing. ITU Internet Reports 2005: The Internet of things (ITU 2005 7th edition), 2006: p. 63-80.

6. Kahn, J.M., Katz, R.H., Pister, K. S. J. , Next century challenges: mobile networking for "Smart Dust", in Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking. 1999, ACM: Seattle, Washington, United States.

7. UK Computing Research Comitee. Available from: http://www.ukcrc.org.uk/.

8. Chalmers, D., Chalmers, M., Crowcroft, J., Kwiatkowska, M., Milner, R., et al. Ubiquitous computing Grand Challenge Manifesto. 2006; Available from: http://www-dse.doc.ic.ac.uk/Projects/UbiNet/GC/manifesto.html.

9. Benerecetti, M., Bouquet, P., Ghidini, C., Contextual Reasoning Distilled. Journal of Experimental & Theoretical Artificial Intelligence, 2000. 12: p. 279-305.

10. Giunchiglia, F.B., P., Introduction to contextual reasoning. An Artificial Intelligence perspective, in Perspectives on Cognitive Science, B. Kokinov, Editor. 1996, NBU Press. p. 138–159.

11. Schilit, B., Theimer, M., Disseminating Active Map Information to Mobile Hosts. IEEE Network, 1994. 8(5): p. 22-32.

12. Ryan, N., Pascoe, J., Morse, D., Enhanced Reality Fieldwork: the Context-Aware Archaeological Assistant., in Computer Applications in Archaeology 1997, V.G.a.M.v.L.a.S. Exxon, Editor. 1998, Tempus Reparatum.

13. Brown, P.J., Bovey, J.D. Chen, X., Context-Aware Applications: From the Laboratory to the Marketplace. IEEE Personal Communications, 1997. 4(5): p. 58-64.

14. Dey, A.K., Context-Aware Applications: The Cyberdesk Project, in AAAI 1998 Spring Symposium on Intelligent Environments. 1998.

15. Brown, P.J., The Stick-e Document: a Framework for Creating Context-Aware Applications. Electronic Publishing, 1996. 8(2): p. 259-272.

16. Schilit, B., Adams, N. Want, R., Context-Aware Computing Applications, in 1st International Workshop on Mobile Computing Systems and Applications. 1994. p. 85-90.

17. Prekop, P., Burnett, M., Activities, Context and Ubiquitous Computing. Special Issue on Ubiquitous Computing Computer Communications, 2003. 26(11): p. 1168–1176.

18. Abowd, G.D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., Steggles, P., Towards a Better Understanding of Context and Context-Awareness, in Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing. 1999, Springer-Verlag: Karlsruhe, Germany.

19. Abowd, G.D., et al., Towards a Better Understanding of Context and Context-Awareness, in Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing. 1999, Springer-Verlag: Karlsruhe, Germany.

20. Gustavsen, R.M. Condor – an application framework for mobility-based context-aware applications. in Proceedings of the Workshop on Concepts and Models for Ubiquitous Computing. 2002. Goeteborg, Sweeden.

21. Hofer, T., et al., Context-Awareness on Mobile Devices - the Hydrogen Approach, in Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 9 - Volume 9. 2003, IEEE Computer Society.

22. Donghai, G., Weiwei, Y., Sungyoung, L., Young-Koo, L., Context Selection and Reasoning in Ubiquitous Computing, in Proceedings of the The 2007 International Conference on Intelligent Pervasive Computing. 2007, IEEE Computer Society.

23. Gwizdka, J., What's in the Context?, in Computer Human Interaction 2000 (CHI2000). 2000: Hague, The Netherlands.

24. Want, R., Hopper, A., Falcao, V., Gibbons, J., The active badge location system. ACM Trans. Inf. Syst., 1992. 10(1): p. 91-102.

25. Hull, R., Neaves, P., Bedford-Roberts, J., Towards Situated Computing, in Proceedings of the 1st IEEE International Symposium on Wearable Computers. 1997, IEEE Computer Society.

26. Rekimoto, J., Ayatsuka, Y., Hayashi, K., Augment-able Reality: Situated Communication through Physical and Digital Spaces, in Proceedings of the 2nd IEEE International Symposium on Wearable Computers. 1998, IEEE Computer Society.

27. Fickas, S., Korteum, G., Segall, Z., Software Organization for Dynamic and Adaptable Wearable Systems, in 1st International Symposium on Wearable Computers. 1997. p. 56-63.

28. Pascoe, J., Adding Generic Contextual Capabilities to Wearable Computers, in Proceedings of the 2nd IEEE International Symposium on Wearable Computers. 1998, IEEE Computer Society. p. 92-99.

29. Johnson, R.E., Frameworks = (components + patterns). Commun. ACM, 1997. 40(10): p. 39-42.

30. Pree, W., Meta Patterns - A Means For Capturing the Essentials of Reusable Object-Oriented Design, in Proceedings of the 8th European Conference on Object-Oriented Programming. 1994, Springer-Verlag.

31. Wirfs-Brock R.J., J.R.E., Surveying Current Research in Object-Oriented Design. Communications of the ACM, 1990. 33(9).

32. Gamma, E., Helm, R., Johnson, R., Vlissides, J., Design Patterns: Elements of Reusable Object-Oriented Software, ed. Addison-Wesley. 1995.

33. CORBA - Common Object Request Broker Architecture.

34. Microsoft COM Technology.

35. Brown, P.J., The Stick-e Document: a Framework for Creating Context-Aware Applications. Proceedings of the Electronic Publishing, 1996. 8(2): p. 259-272.

36. Pascoe, J., The stick-e note architecture: extending the interface beyond the user, in Proceedings of the 2nd international conference on Intelligent user interfaces. 1997, ACM: Orlando, Florida, United States.

37. SGML- Standard Generalized Markup Language. Available from: http://www.w3.org/MarkUp/SGML/.

38. Dey A. K., A.G.D. The Context Toolkit: A toolkit for context-aware applications. Available from: http://www.cs.cmu.edu/~anind/context.html.

39. Dey, A.K., Abowd, G. D., The Context Toolkit: Aiding the development of context-aware applications., in Workshop on Software Engineering for Wearable and Pervasive Computing. 2000: Limerick, Ireland.

40. XML - Extensive Markup Language. Available from: http://www.w3.org/XML/.

41. HTTP - Hypertext Transfer Protocol. Available from: http://www.w3.org/Protocols/.

42. Sousa, J.P., Garlan, D., Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments, in Proceedings of the IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture: System Design, Development and Maintenance. 2002, Kluwer, B.V. p. 29-43.

43. Garlan, D., Siewiorek, D., Smailagic, A., Steenkiste, P., Project Aura: Toward Distraction-Free Pervasive Computing. IEEE Pervasive Computing, 2002. 1(2): p. 22-31.

44. Korpiää, P., Mäntyjärvi, J., Kela, J., Keränen, H., Malm, E-J., Managing context information in mobile devices. IEEE Pervasive Computing, 2003: p. 42-51.

45. The Gaia Project. Available from: http://gaia.cs.uiuc.edu/.

46. Roman, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R.H. and Nahrstedt, K., A Middleware Infrastructure for Active Spaces. IEEE Pervasive Computing, 2002. 1(4): p. 74-83.

47. Hofer, T., Schwinger, W., Pichler, M., Leonhartsberger, G., Altmann, J., Retschitzegger, W., Context-Awareness on Mobile Devices - the Hydrogen Approach, in Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 9 - Volume 9. 2003, IEEE Computer Society.

48. Gu, T., Keng Pung, H., Qing Zhang, D., A middleware for building context-aware mobile services, in IEEE 59th Vehicular Technology Conference (VTC 2004). 2004. p. 2656- 2660.

49. Gu, T., Wang, XH., Pung, HK., Zhang, DQ. An Ontology-Based Computer Model in Intelligent Environments. in Communication Networks and Distributed Systems Modeling and Simulation Conference. 2004.

50. OWL - Web Ontology Language. Available from: http://www.w3.org/TR/owl-features/.

51. Giunchiglia, F., Shvaiko, P., Semantic matching. Knowl. Eng. Rev., 2003. 18(3): p. 265-280.

52. Biegel G., C., V., A Framework for Developing Mobile, Context-aware Applications, in Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04). 2004, IEEE Computer Society.

53. Chen, H., An Intelligent Broker Architecture for Pervasive Context-Aware Systems. 2004, University of Maryland: Baltimore County.

54. Chen, H., Finin, T., Joshi, A. An Intelligent Broker for ContextAware Systems. in Proceedongs of Ubicomp. 2003. Washington, USA.

55. Chen, H., Finin, T., Joshi, A., An ontology for context-aware pervasive computing environments. Knowl. Eng. Rev., 2004. 18(3): p. 197-207.

56. Fahy, P., Clarke, S., CASS – a middleware for mobile context-aware applications., in Workshop on Context Awareness, MobiSys. 2004.

57. Shehzad, A., Ngo, H.Q., Anh Pham, K., Y. Lee, S. Formal Modeling in Context Aware Systems. in Proceedings of The 1st International Workshop on Modeling and Retrieval of Context (MRC'2004). 2004. Ulm, Germany.

58. Yamabe, T., Takagi, A., Nakajima, T., Citron: A Context Information Acquisition Framework for Personal Devices, in Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications. 2005, IEEE Computer Society.

59. Carriero, N., Gelernter, D., Linda in context. Communications of the ACM, 1989. 32(4): p. 444-458.

60. Will, W. LinuxTuples. Available from: http://linuxtuples.sourceforge.net/.

61. Gu, T., Kwok, Z., Keong Koh, K.,Keng Pung, H. A Mobile Framework Supporting Ontology Processing and Reasoning. in Proceedings of the 2nd Workshop on Requirements and Solutions for Pervasive Software Infrastructures, in conjunction with the 9th International Conference on Ubiquitous Computing (Ubicomp '07). 2007. Austria.

62. RDF - Resource Description Framework. Available from: http://www.w3.org/RDF/.

63. RDQL - A Query Language for RDF. Available from: http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/.

64. Dey, A.K., Abowd, G. D., A conceptual framework and a toolkit for supporting rapid prototyping of context-aware applications. Human-Computer Interactions (HCI) Journal, 2001: p. 97–166.

65. Donghai, G., Weiwei, Y., Seong, J. C., Gavrilov, A., Young-Koo, L., Sungyoung, L. Devising a Context Selection-Based Reasoning Engine for Context-Aware Ubiquitous Computing Middleware. in International Conference on Ubiquitous Intelligence and Computing (UIC 2007, LNCS). Hong Kong, China.

66. Baldauf, M., Dustdar, D., A Survey on Context-Aware systems. Int. J. Ad Hoc and Ubiquitous Computing, 2007. 2(4): p. 263-278.

67. Indulska, J., Sutton, P., Location management in pervasive systems. Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003-Volume 21, 2003: p. 143-151.

68. Ambler, S., Mapping Objects to Relational Databases. URL: http://www. ambysoft. com/mappingObjects. pdf, 1997.

69. Brewington, B.E., Cybenko, G., Keeping Up with the Changing Web. Computer, 2000. 33(5): p. 52-58.

70. UPnP - Universal Plug and Play. Available from: http://www.upnp.org/.

71. SSDP - Simple Service Discovery Protocol. Available from: http://quimby.gnus.org/internet-drafts/draft-cai-ssdp-v1-03.txt.

72. JINI Technology. Available from: http://www.jini.org/wiki/Main_Page.

73. SALUTATION Service Discovery. Available from: http://salutation.org/.

74. SLP - Service Location Protocol. Available from: http://srvloc.sourceforge.net/.

75. Bluetooth technology. Available from: www.bluetooth.org.

76. Kagal, L.F., Joshi, T.A., A policy language for a pervasive computing environment. Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop on, 2003: p. 63-74.

77. Buchholz, T., Küpper, A., Schiffers, M. Quality of Context: What It Is and Why Wee Need It. in Proceedings on Wokshopp HP OpenView Univ. Assn. 2003. Geneva, Switzerland.

78. Microsoft .NET Framework. Available from: http://www.microsoft.com/NET/.

79. J2ME - Java 2 Micro Edition. Available from: http://java.sun.com/javame/index.jsp.

80. de Ipiña, D.L., Katsiri, E., An ECA Rule-Matching Service for Simpler Development of Reactive Applications. Published as a supplement to the Proceedings of Middleware, 2001.

81. Cho, S., Kim, KJ., Sung Hwang, K., Song, I-J., AniDiary Daily Cartoon-Style Diary Exploits Bayesian Networks. Pervasive Computing, 2007. 6(3): p. 66-77.

82. Kleemann, T., Towards Mobile Reasoning. 2006 International Workshop on Description Logics DL'06, 2006: p. 231.

83. Sinner, A., Kleemann, T., KRHyper–In Your Pocket. Proc. of the International Conference on Automated Deduction (CADE-20), 2005. 3632: p. 452–458.

84. Bu, Y., Gu, T., Tao, X., Li, J., Chen, S., Lu, J., Managing Quality of Context in Pervasive Computing, in Proceedings of the Sixth International Conference on Quality Software. 2006, IEEE Computer Society.

85. Satyanarayanan, M., Pervasive computing: vision and challenges. IEEE Personal Communications, 2001. 8(4): p. 10-17.

86. Grace, P., Blair, G.S., Samuel, S., A reflective framework for discovery and interaction in heterogeneous mobile environments. SIGMOBILE Mob. Comput. Commun. Rev., 2005. 9(1): p. 2-14.

87. Siegemund, P., Krauer, T. Integrating Handhelds into Environments of Cooperating Smart Everyday Objects. in Proceedings of Ambient Intelligence: Second European Symposium, EUSAI 2004. 2004. Eindhoven, The Netherlands.

88. Ghai, R., Singh, S., Maintaining seamless communication between mobile users: An Architecture and Communication Protocol for picocellular networks. IEEE Personal Communications, 1994. 1(3).

89. Bakshi, B.S., Krishna, P., Pradhan, D.K., Vaidya, N.H., Providing seamless communication in mobile wireless networks, in Proceedings of the 21st Annual IEEE Conference on Local Computer Networks. 1996, IEEE Computer Society.

90. Sainz, D., Almeida, A., Valdés, J., de Ipiña, D.L. Towards a communication agnostic middleware for Ambient Intelligence. in IADIS International Conference on Ubiquitous Computing. 2006.

91. SMILE - Structural Modeling, Inference and Learning Engine. Available from: http://genie.sis.pitt.edu.

92. Chen, G., Kotz, D., A Survey of Context-Aware Mobile Computing Research. 2000, Dartmouth College.

93. ConteXtML - Markup Language for Context Representation. Available from: http://www.cs.kent.ac.uk/projects/mobicomp/fnc/ConteXtML.html.

94. CC/PP - Composite Capabilities / Preferences Profile.

95. UAProf - User Agent Profile. Available from: http://www.wapforum.org.

96. Henricksen, K., Indulska, J., Rakotonirainy, A., Modeling Context Information in Pervasive Computing Systems, in Proceedings of the First International Conference on Pervasive Computing. 2002, Springer-Verlag.

97. Cheverst, K., K. Mitchell, and N. Davies, Design of an object model for a context sensitive tourist GUIDE. Computers & Graphics, 1999. 23(6): p. 883-891.

98. McCarthy, J., Notes on formalizing context.

99. Gray, P.D., Salber, D., Modelling and Using Sensed Context Information in the Design of Interactive Applications, in Proceedings of the 8th IFIP International Conference on Engineering for Human-Computer Interaction. 2001, Springer-Verlag.

100. Strang, T., Linnhoff-Popien, C., A context modeling survey, in Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004. 2004.

101. Gruber, T.R., A translation approach to portable ontology specifications. Knowl. Acquis., 1993. 5(2): p. 199-220.

102. FOAF - Friend of a Friend. Available from: http://www.foaf-project.org/.

103. Chen, H., Perich, F., Finin, T., Joshi, A. SOUPA: standard ontology for ubiquitous and pervasive applications. in Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on. 2004.

104. Wang, X., Zhang, DQ., Gu, T., Pung, HK., Ontology Based Context Modeling and Reasoning using OWL, in Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops. 2004, IEEE Computer Society.

105. Ebling, M., Hunt, G.D.H., Lei, H. Issues for Context Services for Pervasive Computing. in Middleware 2001 Workshop on Middleware for Mobile Computing. 2001. Heidelberg, Germany.

106. van Sinderen, M., van Halteren, A.T., Wegdam, M., Meeuwissen, HB., Eertink, EH., Supporting context-aware mobile applications: an infrastructure approach. Communications Magazine, IEEE, 2006. 44(9): p. 96 - 104.

107. Dey, A., Mankoff, J., Abowd, G. and S. Carter, Distributed mediation of ambiguous context in aware environments, in Proceedings of the 15th annual ACM symposium on User interface software and technology. 2002, ACM: Paris, France.

108. Manko, J., Abowd, G.D., Hudson, S.E., OOPS: A toolkit supporting mediation techniques for resolving ambiguity in recognition-based interfaces. Computers and Graphics, 2000. 6(24): p. 819-834.

109. Castro, P., Chiu, P., Kremenek, T., Muntz, R.R., A Probabilistic Room Location Service for Wireless Networked Environments, in Proceedings of the 3rd international conference on Ubiquitous Computing. 2001, Springer-Verlag: Atlanta, Georgia, USA.

110. Tang, S., Yang, J., Wu, Z., A Context Quality Model for Ubiquitous Applications, in Proceedings of the 2007 IFIP International Conference on Network and Parallel Computing Workshops. 2007, IEEE Computer Society.

111. Krause, M., Hochstatter, I., Challenges in Modelling and Using Quality of Context (QoC), in Mobility Aware Technologies and Applications, Springer, Editor. 2005, Springer Berlin / Heidelberg: Berlin. p. 324-333.

112. Preuveneers, D., Berbers, Y., Architectural Backpropagation Support for Managing Ambiguous Context in Smart Environments, in Universal Access in Human-Computer Interaction. Ambient Interaction, Springer, Editor. 2007. p. 178-187.

113. Hodge, V., Austin, J., A Survey of Outlier Detection Methodologies. Artif. Intell. Rev., 2004. 22(2): p. 85-126.

114. Peng, H.C., Long, F., and Ding, C., Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2005. 27(8): p. 1226-1238.

115. Mitchell, T.M., Machine Learning. 1997: McGraw-Hill Higher Education. 414.

116. Quinlan, J.R., Induction of Decision Trees. Mach. Learn. 1(1): p. 81-106.

117. Anderson, J.A., An introduction to neural networks. 1995: MIT Press.

118. Strang, T., Linnhoff-Popien, C., Frank, K. CoOL: A Context Ontology Language to enable Contextual Interoperability. in Proceedings of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS2003). 2003.

119. Rasheed, F., Lee, Y-K., Lee, S., Measuring the Benefits of Summarizing Quantitative Information in Pervasive Computing Systems, in Proceedings of the 5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering,Software Architecture and Reuse. 2006, IEEE Computer Society.

120. MIT - Massachusetts Institute of Technology. Available from: http://web.mit.edu.

121. Han, J., Kamber, M., Data mining: concepts and techniques. 2000: Morgan Kaufmann Publishers Inc. 550.

122. Grrene, K., The 10 Emerging Technologies of 2008. Technology Review, 2008.

123. Pentland, A. MIT Reality Mining Laboratory. Available from: http://reality.media.mit.edu/.

124. Eagle, N., Pentland, A., Reality mining: sensing complex social systems. Personal and Ubiquitous Computing, 2006. 10(4): p. 255-268.

125. Pentland, A. Reality Mining Data Model. 2006; Available from: http://reality.media.mit.edu/download.php.

126. Eagle, N., Pentland, A., Social Network Computing. UbiComp 2003: Ubiquitous Computing: 5th International Conference, Seattle, WA, USA, October 12-15, 2003: Proceedings, 2003.

127. Eagle, N., Pentland, A., Social serendipity: mobilizing social software. IEEE Pervasive Computing, 2005. 4(2): p. 28-34.

128. Miklas, A., Gollu, K.K., Chan, K.K.W., Saroiu, S., Gummadi, K.P., de Lara, E. Exploiting Social Interactions in Mobile Systems. in Proceedings of UbiComp 2007: Ubiquitous Computing, Springer-Verlag Lecture Notes in Computer Science 4717. 2007.

129. Congleton, B., Nainwal, S., Mining the Mine: Network Analysis of the Reality Mining Dataset. CSCS 535 Networks: Theory and Application, 2007.

130. WU, L., WABER, B.N., ARAL, S., BRYNJOLFSSON, E., PENTLAND, A., Mining Face-to-Face Interaction Networks using Sociometric Badges: Predicting Productivity in an IT Configuration Task. 2008.