Tesis defendida por

Edgardo Avilés López

y aprobada por el siguiente Comité

Dr. José Antonio García Macías

Director del Comité

Dra. Ana Isabel Martínez García

Miembro del Comité

Dra. Marcela Deyanira Rodríguez Urrea

Miembro del Comité

Dr. Jesús Favela Vara

Miembro del Comité

Dr. José Antonio García Macías

Coordinador del Programa de Posgrado
en Ciencias de la Computación

Dr. David Hilario Covarrubias Rosales

Director de Estudios de Posgrado

13 de diciembre de 2012

CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN SUPERIOR DE ENSENADA

Programa de Posgrado en Ciencias
en Ciencias de la Computación

Creación de aplicaciones para ambientes inteligentes: un marco de trabajo basado en fuentes de datos y servicios en la Internet de las Cosas

Tesis
que para cubrir parcialmente los requisitos necesarios para obtener el grado de Doctor en Ciencias

Presenta:
Edgardo Avilés López

Ensenada, Baja California, México
2012

Resumen de la tesis de Edgardo Avilés López, presentada como requisito parcial para obtener el grado de Doctor en Ciencias en Ciencias de la Computación.

Creación de aplicaciones para ambientes inteligentes:
un marco de trabajo basado en fuentes de datos
y servicios en la Internet de las Cosas

Resumen aprobado por:

Dr. José Antonio García Macías

El surgimiento de la Web 2.0 trajo consigo un rico ecosistema de servicios de aplicación, permitiendo a los desarrolladores utilizar la funcionalidad de las aplicaciones Web de última generación en sus propias soluciones personalizadas. Esta situación, junto con los recientes desarrollos en la Internet de las Cosas, están sentando las bases de nuevos ambientes inteligentes basados en protocolos del conjunto IP en el cual las aplicaciones y los servicios pueden ser combinados en formas no posibles anteriormente. Recientemente, la mayor parte de los trabajos de investigación se han concentrado en mejorar las capacidades de conectividad y de interconectividad de la infraestructura de la Web de las cosas y en permitir el acceso a la información para la siguiente generación de servicios. Sin embargo, aún hay dos problemáticas que deben ser atendidas. Primero, cómo será modelada la información y la funcionalidad de los servicios en estos ambientes para facilitar la abstracción y la composición, y segundo, cómo será la interacción de los usuarios con los ambientes para permitir que las aplicaciones se adapten a sus necesidades particulares. En esta tesis se presenta el desarrollo de un marco de trabajo y de un modelo de interacción de usuario para aplicaciones en la Internet de las Cosas basados en las tecnologías de la Web moderna, como una propuesta de solución a ambas problemáticas.

Palabras clave: **Arquitecturas Orientadas a Servicios**, **Conciencia de Contexto**, **Internet de las Cosas**, **Web de las Cosas**, ***Mashups***, **Ambientes Inteligentes**.

Abstract of the thesis presented by Edgardo Avilés López, as a partial requirement to obtain the Doctor in Science degree in Computer Science.

Creating smart environment applications:
a framework based on data sources and
services in the Internet of Things

Abstract approved by:

Dr. José Antonio García Macías

Along with the advent of the Web 2.0 came a rich ecosystem of application services enabling developers to use the functionality provided by Web applications into their own customized solutions. This, together with the current developments on the Internet of Things are laying the foundations of new IP-based smart environments in which applications and services are combined to support users in ways not possible before. Recently, most of the research has focused on improving the networking capabilities of the Internet of Things infrastructure and on enabling the access to the following generation of services. However, there are two more issues that need to be addressed. First, how data and functionality provided by services on these smart environments should be modeled to facilitate abstraction and composition, and second, how users are intended to interact with the environments to make applications support their particular needs. In this thesis, we present the development of a framework and an user-interaction model for Internet of Things applications based on the technologies of the modern Web as a solution proposal for both issues.

Keywords: **Service-Oriented Architectures**, **Context-Awareness**, **Internet of Things**, **Web of Things**, **Mashups**, **Smart-Environments**.

Dedicatoria

A mis papás Alma y Magdaleno.

Agradecimientos

Antes que nada, quisiera agradecer a mi asesor, Dr. J. Antonio García Macías, por su atención, su apoyo incondicional, sus críticas, comentarios y sugerencias durante el desarrollo de esta investigación. Siempre le agradeceré su confianza, paciencia y el haberme aceptado como estudiante de doctorado. También quisiera agradecer a los miembros de mi comité de tesis: Dra. Ana Isabel Martínez García, Dra. Marcela Deyanira Rodríguez Urrea y Dr. Jesús Favela Vara, por su tiempo, consejos, correcciones e invaluables comentarios que me ayudaron a concluir mi trabajo.

A mis papás, a quienes dedico este trabajo, gracias por apoyarme en todo momento, por los valores que me han inculcado y por haberme dado la oportunidad de tener una excelente educación en el transcurso de mi vida. Sobre todo por ser tan buen ejemplo. A mis hermanos, cuñada, mis sobrinitos Gianni y Miguelito, y al resto de mi gran familia. Muchas gracias por su cariño, paciencia, apoyo y amor incondicional.

A mis amigos Carlos, Iván, Leonardo, Argelia y Rolando, por confiar y creer en mí, por haber compartido conmigo sus conocimientos, por aguantarme y apoyarme en los momentos difíciles y sobre todo por su amistad. Gracias por los grandes momentos.

Mi agradecimiento al Centro de Investigación Científica y de Educación Superior de la ciudad de Ensenada, Baja California. Muchas gracias al personal que allí labora, sobre todo a Carito y Lydia, por su apoyo y cariño. También quiero agradecer al Dr. David Hilario Covarrubias Rosales, por el apoyo recibido durante la etapa final de mi trabajo de investigación.

También, quiero agradecer a Christian P. García Martínez, por su confianza, por abrirme las puertas de Ubilogix, por haberme dado la oportunidad de crecer profesionalmente y por tenerme paciencia durante la etapa final de mi tesis.

Muchas gracias a todas aquellas personas que de una u otra forma colaboraron o participaron en la realización de esta investigación. También, a los compañeros y amigos que tuve la fortuna de conocer durante la maestría y el doctorado, muchas gracias a todos.

Finalmente, quiero agradecer al Consejo Nacional de Ciencia y Tecnología, por su apoyo económico sin el cual no hubiera sido posible este trabajo de investigación.

Contenido

Lista de figuras

Lista de tablas

Capítulo 1

Introducción

En los últimos años se ha venido trabajando en lo que algunos llaman la próxima Internet (Vasseur y Dunkels, 2010), una red formada por objetos de la vida diaria a los que se les han incorporado sensores, actuadores, capacidades de comunicación y de cómputo. Los ahora llamados objetos inteligentes (o *smart-objects*), tienen la capacidad de procesar información, identificarse a sí mismos, estar conscientes del ambiente en el que se encuentran y de compartir información de manera automática unos con otros. Esto es posible gracias a la reducción de costos, dimensiones, y a los recientes avances en sistemas embebidos y comunicaciones inalámbricas que brindan las herramientas para aumentar e interconectar los objetos del mundo físico con la Internet de manera efectiva, práctica y económica. Hoy en día prácticamente cualquier dispositivo electrónico puede ser conectado a la red. Todas estas ideas describen la visión de la Internet de las Cosas (Schoenberger, 2002; Gershenfeld *et al.*, 2004; Rellermeyer *et al.*, 2008).

La amplia adopción de la tecnología móvil digital ha permitido a los usuarios contar con una experiencia de comunicación inalámbrica casi ubicua. La Web, una de las muchas aplicaciones de la Internet, ha impactado fuertemente la forma en la que las personas interactúan con las computadoras y también ha creado una cultura que es más susceptible a escenarios de aplicación que antes sólo eran posibles en la ciencia-ficción. La evolución de la Web ha traído consigo nuevas tecnologías de programación distribuida que permiten reutilizar los contenidos y la funcionalidad de los sitios Web en otras aplicaciones. La Web ya no es utilizada solamente por personas sino también por otras computadoras. La Internet de las Cosas está cada vez más cerca. No es raro entonces que algunos investigadores se hayan propuesto retomar los estándares y tecnologías bien conocidas, probadas

y aceptadas de la Web para formar la plataforma común necesaria para integrar los objetos inteligentes de lo que habla la Internet de las Cosas. A estos esfuerzos se les describe como la Web de las cosas.

El trabajo de investigación y desarrollo tanto en la Internet como en la Web de las Cosas está sentando las bases para finalmente hacer realidad escenarios de aplicación tan dinámicos y avanzados como los escenarios descritos en la visión del cómputo ubicuo (Weiser, 1999). Sin embargo, antes que esto pase, existen tres metas muy claras que se deben atender. Primero, el mundo físico debe ser «aumentado» a través del aprovisionamiento de dispositivos heterogéneos que posibiliten distintas formas de interactividad, una meta muy parecida a lo que se está logrando con la Internet de las Cosas. Segundo, las prácticas de la vida diaria de las personas deben ser comprendidas y apoyadas en concordancia. Y finalmente, los dispositivos en red deben ser orquestados para proveer una experiencia de usuario holística (Abowd y Mynatt, 2002), es decir, que los usuarios puedan interactuar con los ambientes como un todo y no como entidades de operación independientes.

Para lograr cumplir estas metas expertos en dominios de aplicación e investigadores enfocados en la adquisición y modelado de contexto, interacción humano-computadora, y en otras muchas más áreas de características no principalmente técnicas, deben experimentar construyendo un gran número de prototipos de aplicaciones. Esto es inherente a la naturaleza de la investigación en el cómputo ubicuo, en donde la construcción de prototipos es una parte fundamental e intrínseca de la metodología de investigación (Weiser, 1993).

Tecnológicamente hablando, existen dos problemas aún más precisos. El primero es que la infraestructura actual de la Internet no está diseñada, o más bien lista aún, para dar soporte a requerimientos y características tales como la calidad de servicio, privacidad, reducción de latencia y otros muchos problemas que deben ser resueltos para que los ambientes inteligentes puedan ser realmente efectivos en la práctica. En cuanto a este punto existen algunas propuestas en la forma de arquitecturas, algoritmos, protocolos y

sistemas *middleware* que permiten subsanar algunas de las debilidades de la infraestructura de la Internet actual. El segundo problema es, que una vez que los servicios y aplicaciones estén disponibles y proveídos en una misma red, los recursos y la funcionalidad que estos provean necesitarán ser agregados y procesados con algoritmos y modelos sofisticados ya que deberán de funcionar con la enorme cantidad de datos generados en dichos ambientes. En la Web actual existe ya evidencia que apunta hacia la solución de ambos problemas; el concepto mismo de la Web de las cosas se basa en el hecho de que la Web es en realidad una plataforma de aplicaciones en donde los sitios ya no sólo ofrecen sus contenidos a humanos sino también a sistemas de cómputo a través de interfaces de programación, abiertas y públicas, de fácil acceso. El abstraer un ambiente inteligente en entidades o recursos de la Web permite concentrarse en la lógica de aplicación mientras que los problemas de infraestructura pueden ser atendidos en capas de bajo nivel.

Un ejemplo de tecnología disponible actualmente en cuanto a la necesidad de métodos de agregación y procesamiento son los sistemas *mashups* (Hartmann *et al.*, 2008). Un concepto originado en la música que en las ciencias computacionales se refiere a aplicaciones creadas a partir de combinaciones y modificaciones de diferentes fuentes de datos e interfaces de aplicación. Un ejemplo muy común de *mashup* es una aplicación que utiliza Google Maps (2012) para mostrar la ubicación de las fotografías más recientes de la cuenta de un usuario de Flickr (2012). El problema de este tipo de sistemas es que la creación de un *mashup* requiere de experiencia para realizar tareas tales como la extracción de información de páginas Web (que tal vez no fueron diseñadas para ello), procesamiento de texto, correlación de patrones y formatos de representación de datos (Wong y Hong, 2006) entre otros retos que podrían cambiar dramáticamente dependiendo de los requerimientos específicos del problema o del escenario de aplicación en el que se esté trabajando. Otro problema de los *mashups* es que a pesar de sus beneficios, su uso en aplicaciones para ambientes inteligentes aún no ha sido explorado suficientemente.

1.1 Infraestructuras para cómputo ubicuo

El diseño de infraestructuras para escenarios de cómputo ubicuo no es un problema de investigación reciente. Existe una gran diversidad de trabajos, sobre todo en la exploración del cómo los dispositivos de *hardware* y componentes de *software* deben ser integrados para que colaboren en el apoyo automático y sin distraer a las personas de sus actividades diarias. Aunque no sólo este problema es importante, también hay mucho trabajo en aspectos tales como la interacción humano-computadora, modelos de comunicación, conceptualización, creación de lenguajes de programación, entre otros.

A continuación se analizan brevemente algunas de las herramientas e infraestructuras de *software* que han sido propuestas hasta ahora, así como dos de los principales modelos de diseño. De igual manera se analizan algunas de las propuestas realizadas por el World Wide Web Consortium (2012) en particular, por parte del grupo de trabajo Ubiquitous Web Applications (2012). El presente listado de proyectos está basado en el trabajo de Endres *et al.* (2005) el cual se ha verificado, actualizado y extendido con la inclusión de nuevos proyectos no considerados inicialmente. Para cada uno de ellos se incluye el periodo de tiempo en el cual permanecieron o permanecen activos, y las diferentes características de cada uno en aspectos tales como el lenguaje de programación y la plataforma para la cual están diseñados.

La Figura 1.1 muestra el periodo de tiempo de actividad de cada uno de los proyectos. La línea de vida de un proyecto inicia en la fecha de la primera publicación o en la fecha reportada por los autores como el inicio de la investigación. Como se puede observar, muchos de los proyectos surgieron entre los años 1999 y 2001. Los motivos más comunes del cierre de muchos de estos son dos: la falta de disponibilidad de recursos (*hardware*, presupuesto, asignación de tiempo, etc.) y la finalización de los estudios de posgrado de los autores. Se pudo observar que los proyectos más longevos son aquellos en los que existe una gran cantidad de investigadores o apoyo por parte de la comunidad (gracias al

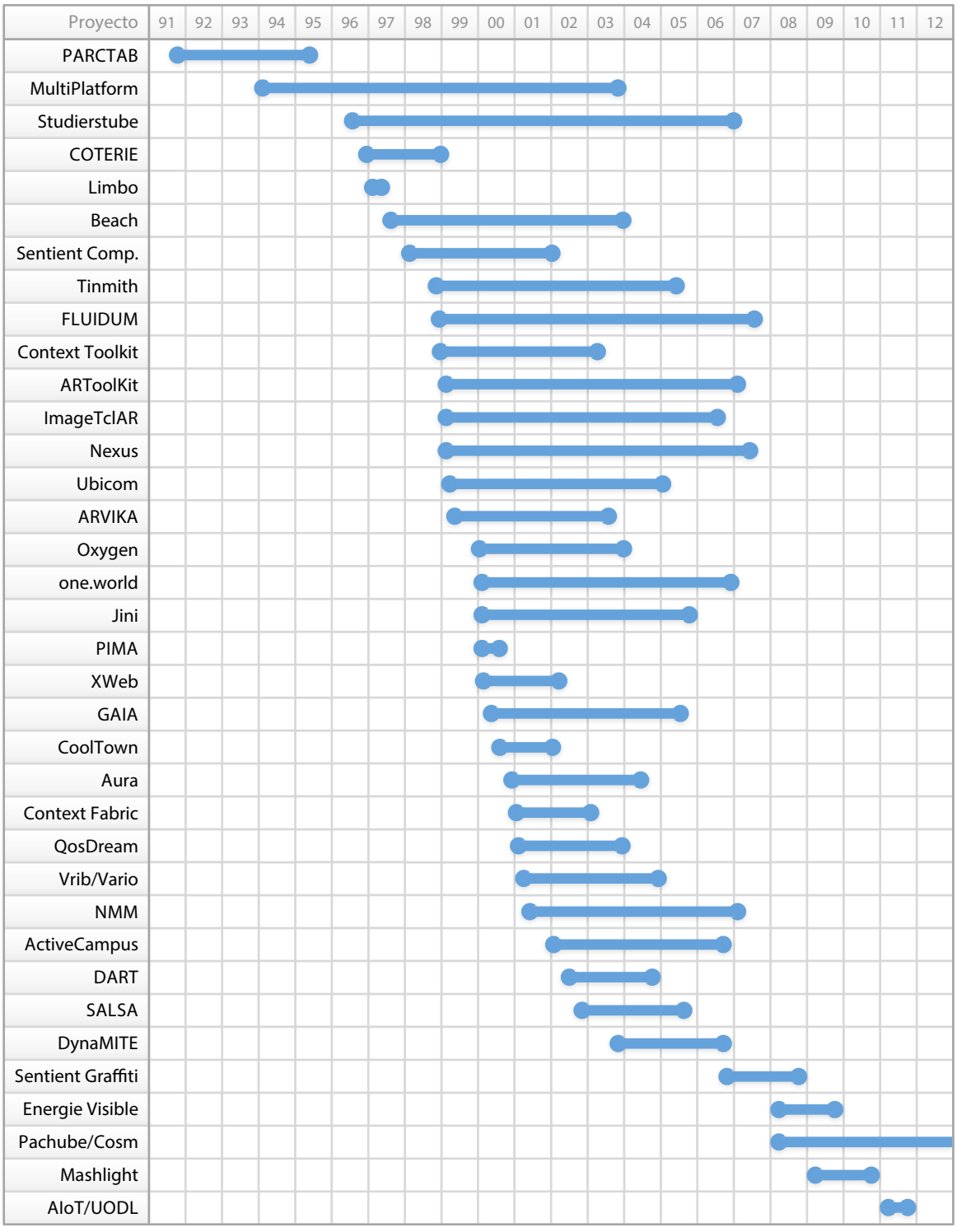


Figura 1.1: Escala de tiempo, vida de los proyectos.

Proyecto	Lenguajes de Programación						Plataforma				
	C/C++	Java	Perl	Python	.NET	Otro	Win	Unix	Mac	Multi	Otra
ActiveCampus	●	○	○	○	○	○	○	○	○	○	●
AIoT	○	○	○	○	○	●	○	○	○	●	○
ARToolKit	●	●	●	●	●	●	●	●	●	○	●
ARVIKA	●	●	○	○	○	○	●	●	○	○	●
Aura	●	●	○	○	○	●	●	●	○	○	○
Beach	○	○	○	○	○	●	●	○	○	●	○
Context Fabric	○	●	○	○	○	○	○	○	○	○	●
Context Toolkit	◐	●	○	◐	◐	◐	○	○	○	○	●
CoolTown	●	●	●	●	●	●	○	○	○	●	○
COTERIE	○	○	○	○	○	●	●	●	○	○	○
DART	●	○	○	○	○	●	●	○	●	●	○
DynaMITE	○	●	○	○	○	○	○	○	○	●	○
Energie Visible	●	●	○	○	○	●	●	○	○	○	○
FLUIDUM	○	●	○	○	○	○	○	○	○	●	○
Gaia	●	●	●	○	○	●	○	○	○	●	○
ImageTclAR	●	○	○	○	○	○	●	●	○	○	○
Jini	○	●	○	○	○	○	●	●	●	●	○
Limbo	○	○	○	○	○	●	●	○	○	●	○
Mashlight	○	○	○	○	○	●	○	○	○	●	○
Multiplatform	●	●	●	○	●	○	●	●	○	○	○
Nexus	○	○	○	○	○	●	○	○	○	●	○
NMM	●	◐	○	○	○	○	○	●	○	○	●
one.world	●	●	○	○	○	○	○	○	○	●	○
Oxygen	○	●	○	○	○	○	●	●	○	○	●
Pachube/Cosm	○	○	○	○	○	●	○	○	○	●	○
PARCTAB	○	○	○	○	○	●	○	○	○	○	●
PIMA	○	○	○	○	○	●	○	○	○	○	●
QoSDream	○	●	○	○	○	○	○	○	○	●	○
SALSA	●	●	○	○	○	○	●	●	○	○	●
Sentient Comp.	●	○	○	●	○	○	○	○	○	●	○
Sentient Graffiti	○	●	○	○	○	○	○	○	○	●	○
Studierstube	●	○	○	●	○	●	●	●	●	○	○
Tinmith	●	○	○	○	○	○	●	●	○	○	●
Ubicom	○	○	○	○	○	●	○	○	○	○	●
Vrib/Vario	●	●	○	○	○	○	○	○	○	○	●
XWeb	○	○	○	○	○	●	○	○	○	○	●

Tabla 1.1: Lenguajes de programación y plataformas por proyecto.

continuo desarrollo de mejoras en los sistemas). La fecha de finalización de los proyectos corresponde a la del último artículo publicado, actualización a su sitio Web o distribución pública del sistema. La Tabla 1.1 muestra un concentrado de los lenguajes de programación con los cuales se pueden desarrollar aplicaciones utilizando las distintas herramientas y sistemas *middleware* y también las plataformas apoyadas por cada uno. Como se puede observar, existen muchas alternativas viables para el desarrollo de aplicaciones en escenarios del cómputo ubicuo. Sin embargo, han pasado más de 20 años desde que Weiser planteó su visión y el cómputo ubicuo aún nos sigue pareciendo ciencia-ficción debido a la falta de aplicaciones tangibles. La razón principal es la falta de integración entre los sistemas, dispositivos y tecnologías actuales, lo cual es agraviado aún más por la gran heterogeneidad de sistemas, hardware y las diferentes especificaciones de los fabricantes (Davies y Gellersen, 2002). Adicionalmente, también está la necesidad de facilidad de uso para los desarrolladores, que son los usuarios de tales sistemas. La mayoría de las propuestas aquí mencionadas requieren de esfuerzos considerables de personalización por parte de expertos en instalación y diagnosis, incluso algunos fueron diseñados como sistemas-propietario por lo que su uso es muy restringido. Aún hay oportunidades de investigación para soluciones que sean fáciles y sencillas de utilizar, efectivas en costo y en recursos y que al mismo tiempo logren ofrecer un soporte adecuado para este tipo de escenarios.

1.1.1 Modelos para sistemas ubicuos

Así como existe una gran diversidad de proyectos de infraestructuras, también lo hay de modelos o patrones arquitectónicos. Esto se puede observar en los modelos en que fueron basados los proyectos analizados. El siguiente es un listado de los proyectos agrupados según su modelo:

Cliente/Servidor Arvika (Wohlgemuth y Triebfürst, 2000), UbiCom (Lagendijk, 2000), ActiveCampus (Griswold *et al.*, 2003), CoolTown (Caswell y Debaty, 2000), Nexus (Nicklas *et al.*, 2001), QoSDream (Naguib *et al.*, 2001), Sentient Computing (Newman *et al.*,

2001), XWeb (Olsen *et al.*, 2001), Energie Visible (Guinard y Trifa, 2009), Pachube/-Cosm (2012), Mashlight (Albinola *et al.*, 2009) y AIoT (Zhang y Mitton, 2011).

Componentes Distribuidos Ubicom (Lagendijk, 2000), Vrib/Vario (Schönfelder *et al.*, 2002) y Gaia (Roman *et al.*, 2002).

Data Flow Tinmith (Piekarski y Thomas, 2003) y NMM (Rodríguez y Favela, 2003).

Agentes de *Software* Xerox PARCTAB (Want *et al.*, 1995), SALSA (Rodríguez y Favela, 2003) y Sentient Graffiti (López-de-Ipiña *et al.*, 2007).

Publish/Subscribe MultiPlatform (Herzog *et al.*, 2003) y QoSDream (Naguib *et al.*, 2001).

Scene Graph AR Toolkit (Kato y Billinghurst, 1999), Coterie (MacIntyre y Feiner, 1996) y Studierstube (Schmalstieg *et al.*, 2002).

Tuple Spaces Limbo (Davies *et al.*, 1997) y one.world (Grimm *et al.*, 2001).

La selección del modelo varía según las necesidades en las que cada proyecto fue enfocado. Es realmente difícil decidir un solo modelo que pueda ofrecer soporte para diferentes escenarios, contextos y dominios de aplicación. Sin embargo, de entre todos los modelos disponibles algunos autores consideran que 2 de ellos tienen mayor afinidad para este tipo de ambientes: el modelo de orientación al contexto y las arquitecturas basadas en servicios. Ambos han sido propuestos y estudiados por distintos proyectos de investigación debido a sus facilidades y características que hacen de ellos buenos cimientos para implementar ambientes ubicuos programables (Yang *et al.*, 2006; Helal, 2005). También, es importante notar que todos los modelos anteriores no son mutuamente excluyentes, se pueden integrar para explotar sus capacidades como se ejemplifica más adelante.

Modelo orientado al contexto

La mejor manera de describir este modelo es utilizando un ejemplo: una casa inteligente es considerada como un ambiente de frontera en el sentido que lo que está dentro

puede ser observado y lo que está fuera no puede ser influenciado. Un espacio puede estar en un estado que es descrito utilizando lógica descriptiva, estados simples tales como frío, caliente, húmedo, entre otros que pueden ser directamente obtenidos de lecturas de sensores. Dichos estados son conocidos como contexto atómico o primario. El contexto atómico puede generar otros contextos más complejos (contexto secundario) (Baader *et al.*, 2003). Por ejemplo, si un cuarto de una casa está caliente y con humo es muy posible que el cuarto esté en llamas. En un momento dado se puede tomar un muestreo del estado actual observado y se puede clasificar esta muestra de acuerdo a la lógica descriptiva. Siguiendo este modelo los desarrolladores deben definir la descripción de los contextos que son de su interés. También deben definir el mapeo entre las lecturas de sensado y los contextos atómicos para ayudar a que los datos tengan sentido para el ambiente.

La ventaja más importante del modelo es lo explícito que es. Al describir los contextos posibles en un ambiente con la lógica descriptiva, el sistema y todas sus entidades pueden identificar sin problemas qué contexto está activo. Por lo mismo, también promueve la interoperabilidad. Otra de sus ventajas es la escalabilidad. Contrario al modelo basado en servicios, donde cada servicio está encapsulado en una entidad individual, en la orientación al contexto sólo una entidad es responsable de activar las acciones.

Modelo basado en servicios

En este modelo cada entidad de *software* se encarga de una pequeña parte de la funcionalidad total constituyendo un servicio que es independiente del resto de las entidades. Los servicios pueden ser proveedores o consumidores y para encontrarse unos a otros existe un directorio. Un servicio debe ser registrado en el directorio para permitir que los consumidores lo puedan encontrar. Este modelo es mucho más tradicional y puede ser descrito como proactivo, de procedimientos o implícito.

La ventaja es que ofrece un control mucho más fino al proveído por la orientación al contexto. La proactividad de los servicios permite la definición sencilla de pasos de control complejos basados en la lectura de sensores. Otra ventaja importante es su amplia aceptación y factibilidad comprobada. Además, desde la perspectiva de los desarrolladores, es mucho más fácil crear servicios ya que utilizan los mismos mecanismos y herramientas de programación en las que están acostumbrados.

Modelo mixto en la Web

Afortunadamente, ambos modelos no son mutuamente excluyentes por lo que se puede tener un sistema modelado con base en servicios con notificaciones cuando el contexto cambie, ó a la inversa, tener un sistema basado en contexto en el cual las entidades que realizan el modelado actúen como servicios individuales. El modelo de orientación a servicios es algo común en la evolución de la programación de sistemas distribuidos en la Web. Recientemente, las arquitecturas orientadas a recursos, a menudo referidas como REST (Fielding, 2000), han tenido mucha aceptación. En un modelo de orientación a recursos las entidades de un sistema no son los actuadores en una aplicación sino la información misma. La funcionalidad de una aplicación está modelada en términos de recursos. Por ejemplo, en un sistema basado en servicios que lleva un listado de los libros disponibles en una biblioteca, la funcionalidad para agregar un libro estaría cubierta por una entidad registrador a la que se le pediría que agregue un libro. En el modelo de recursos por otra parte, la funcionalidad para agregar un libro estaría cubierta por una entidad libros a la que se agregaría un nuevo libro. Es decir, se provocaría un cambio en el recurso libros haciendo uso de una de las únicas cuatro operaciones disponibles (agregar, modificar, actualizar o eliminar). En el modelo basado en recursos sigue existiendo el concepto de directorio, proveedor y consumidor, sin embargo, se basa en qué es lo que está disponible, qué se provee o qué se consume, contrario al concepto de quién provee o quién consume una funcionalidad en particular. La notificación de eventos en la Web es una funcionalidad para la que

existen muchas soluciones diferentes aunque en los sistemas REST una notificación se refiere al aviso de la actualización de un recurso determinado.

1.1.2 La Web ubicua y las tecnologías de la Internet

Como se menciona al inicio de este capítulo, los avances tecnológicos han logrado que prácticamente cualquier dispositivo pueda tener conectividad con Internet. Actualmente, no sólo las computadoras de escritorio y los teléfonos celulares tienen conectividad ahora las cámaras de video, las fotocopiadoras, artículos personales, domésticos y muchos más pueden tener esta facilidad lo que introduce algunas problemáticas tales como la visualización. Incluso este problema tiene dos perspectivas: del lado del cliente, ¿cómo visualizar las páginas y hacerlas amigables al usuario en diferentes dispositivos con diferentes capacidades?, del lado del servidor, ¿cómo diseñar sitios y aplicaciones para que obtengan lo mejor del cliente y su contexto de entrega?

Pero no solamente los celulares tienen conectividad, existen muchos más clientes posibles. Lo cual lleva al hecho de que hay una sola Internet pero muchas formas de usarla. Este problema en particular y muchos otros relacionados son abordados dentro del World Wide Web Consortium (W3C) en el grupo de trabajo Ubiquitous Web Applications (UWA). Este grupo inició en Marzo de 2007 con la visión de reducir los costos de desarrollar aplicaciones que involucran dispositivos ubicuos en red pero basándose en los estándares del W3C para las representaciones. Entre los objetivos del grupo se encuentran:

- Permitir a las aplicaciones utilizar múltiples dispositivos para influir en el mundo físico, en aspectos tales como en la seguridad, el monitoreo y control ambiental, sistemas de entretenimiento, grupos de trabajo distribuidos, mantenimiento, etc.

- Una solución multicapa para las interfaces de usuario que separen las preocupaciones del desarrollador de aplicaciones de los detalles de cómo la interfaz de usuario es implementada en dispositivos específicos.

- Desacoplar a las aplicaciones de los protocolos de transporte, y por lo tanto, permitir a las aplicaciones correr sobre una mezcla de tecnologías de red diferentes y en evolución constante.

- Proveer descripciones ricas de los dispositivos y servicios, y los medios para exponerlas a las aplicaciones y permitirles adaptarse dinámicamente a los cambios en las preferencias del usuario, capacidades de los dispositivos y condiciones ambientales.

- Ver más allá de los navegadores Web a nuevos tipos de aplicaciones basadas en modelos de objetos distribuidos, donde una aplicación ejecutándose en un dispositivo está acoplado a una interfaz de usuario ejecutándose en otro vía un intercambio de eventos.

- Proveer un mecanismo para vinculación de recursos que promueva la Web Semántica para administrar confianza, identidad, privacidad y seguridad a través de descripciones ricas de las capacidades y políticas, el cual se base en protocolos de coordinación de dispositivos tales como Zeroconf (IETF Zeroconf Working Group, 2012), UPnP (UPnP Forum, 2012) y DLNA (Digital Living Network Alliance, 2012).

- Utilizar objetos en los modelos de objetos de documento como *proxies* para recursos locales o externos, permitiendo a las aplicaciones acceder a dichos recursos al definir administradores de eventos o al provocar eventos en los *proxies*.

UWA partió de la premisa de que el navegador Web se ha convertido en la manera ubicua para acceder a todo tipo de información y servicios. Es ubicuo porque se pierde de tal manera que uno no necesita aprender a usarlo cada vez que desea visitar un nuevo sitio

Web. La Web Ubicua busca ampliar las capacidades de los navegadores para permitir nuevos tipos de aplicaciones Web, particularmente aquellas que involucran la coordinación con otros dispositivos.

Internet ha demostrado ser una plataforma muy estable, su uso ha crecido exponencialmente desde su introducción, lo que ha permitido que los protocolos hayan ido madurando al punto que actualmente se los considera como una muy buena opción para construir y desarrollar nuevas tecnologías. Protocolos como HTTP y XML han ganado confiabilidad y demostrado su efectividad lo que puede observarse con el surgimiento de la Web 2.0, un conjunto de herramientas que han provocado la proliferación de aplicaciones y sistemas interconectados y que cada vez más se han integrado en la vida diaria de las personas. Un ejemplo del uso de la Web como impulsor son las arquitecturas orientadas a servicios que ya tenían tiempo de haber sido introducidas. El uso de la Web como medio de comunicación (servicios Web) las llevó a ser adoptadas y muy aceptadas. Estos servicios, como se describen anteriormente, son entidades de *software* simples que cuentan con una interfaz de programación similar a la de la programación con objetos e incluso con la modular. En este aspecto, algunos autores han propuesto técnicas para encapsular a los servicios Web en componentes que permitan la automatización y orquestación de servicios en ambientes dinámicos. Una de estas propuestas son los Servicios Declarativos (Cervantes y Hall, 2003). El problema con esta tecnología es que introduce problemáticas no triviales que dificultan su utilización. Problemáticas tales como la validación de la composición en tiempo de diseño (Riba y Cervantes, 2007).

Con la aparición de la Web de las cosas, el desarrollo para sistemas en los ambientes inteligentes se puede beneficiar de manera importante de la tecnología de *mashups*. Con ella, se puede identificar la funcionalidad común para una variedad de escenarios en un dominio en particular y aislarla en un servicio Web independiente y agnóstico al lenguaje de programación. Después, se pueden combinar estos recursos para componer la funcio-

nalidad necesaria para dar soporte a un nuevo escenario o tarea. Adicionalmente, como se puede esperar que el número de dispositivos y recursos crezca exponencialmente, los *mashups* pueden ayudar para delegar el procesamiento (para minería o agregación de datos) y generar nuevos recursos más manejables y escalables. Además, y mucho más importante, un sistema *mashup* es mucho más fácil de utilizar que el componer los servicios y recursos manualmente en algún lenguaje de programación. Incluso, existen algunos sistemas que permiten la composición de estos de manera gráfica, arrastrando y soltando representaciones gráficas de los mismos.

1.2 Objetivo de investigación

Desarrollar un marco de trabajo, utilizando las tecnologías e ideas de la Web de las Cosas, para la creación de aplicaciones a partir de la adquisición, composición y agregación de las fuentes de datos, servicios y dispositivos disponibles en los ambientes inteligentes e Internet de manera efectiva, sencilla y fácil.

1.2.1 Preguntas de investigación

Para lograr el objetivo de investigación, este trabajo de tesis busca responder las siguientes preguntas de investigación:

1. ¿Cómo integrar sensores, actuadores y otros aparatos en una infraestructura en común que tome en cuenta aspectos tales como la necesidad de notificación de eventos y el control de acceso?

2. ¿Cómo podría simplificarse la composición de fuentes de datos y servicios de manera que esta pueda ser descrita e implementada por desarrolladores no expertos?

3. ¿A qué características y requerimientos especiales de los ambientes inteligentes se les dará soporte en este trabajo?

4. ¿Qué tipo de escenarios de aplicación resultan más apropiados para el soporte basado en la Web y cuáles son más problemáticos?

5. ¿Qué protocolos, lenguajes de programación y otras tecnologías de la Web son adecuadas para desarrollar de un marco de trabajo para ambientes inteligentes?

6. ¿Cuál es el proceso que un desarrollador de aplicaciones de ambientes inteligentes debe seguir para construir este tipo de sistemas utilizando el marco de trabajo?

7. ¿Qué tan adecuada es la abstracción que ofrecen los *mashups* de servicios Web para los desarrolladores de aplicaciones en ambientes inteligentes?

8. ¿Qué modelo de interacción de usuario seguirían las personas que utilicen los ambientes inteligentes basados en *mashups*?

9. ¿Cómo podría permitirse la fácil extensión de las capacidades del marco de trabajo para permitir su aplicación en otros dominios de aplicación y la inclusión de nuevos tecnologías?

1.3 Metodología

Buscando responder las preguntas de investigación, y así completar el objetivo de investigación de este trabajo, se siguió la siguiente metodología:

Análisis de la literatura. Como fase inicial, se realizó una revisión exhaustiva de la literatura buscando analizar sistemas e infraestructuras con propuestas de cómo incorporar la consciencia de contexto en las aplicaciones. Se puso especial atención en los modelos y mecanismos de abstracción ofrecidos a los desarrolladores. También se siguió de cerca la evolución del trabajo de algunos investigadores y grupos de trabajo alrededor del concepto de la Web de las cosas con el objetivo de conocer sus propuestas y las limitaciones de sus trabajos.

Integración de sensores y actuadores. Se analizó y definió la estrategia de integración de sensores, actuadores y otros aparatos en una infaestructura común, tomando en cuenta la necesidad de notificación de eventos y el control de acceso, entre otros aspectos, buscando siempre la simplificación de procesos.

Simplificación de la composición. Se trabajó en la simplificación de la composición de fuentes de datos y servicios buscando requerir un mínimo conocimiento en lenguajes de programación y estrategias de comunicación.

Recopilación de requerimientos y características. Se recopilaron, acotaron y concretaron los requerimientos y características especiales de los ambientes inteligentes a los que se daría soporte en el presente trabajo de investigación. Dicha selección estuvo enfocada en aquellos escenarios de aplicación que presentaran retos y características para los cuales el soporte basado en la Web no fuera trivial y para los que su soporte representara la mejor relación costo/beneficio.

Definición/conformación del marco de trabajo. Tomando en cuenta los requerimientos obtenidos, escenarios, así como el resultado de la experimentación con los prototipos que definieron los principales elementos del marco de trabajo, así como también el proceso que un desarrollador de aplicaciones de ambientes inteligentes debe seguir para construir este tipo de sistemas utilizando el marco de trabajo.

Análisis de la abstracción y experimentación. Durante esta fase se evaluó el nivel de abstracción ofrecido por nuestras propuestas decidiéndose la creación de un editor de *mashups* para proveer una capa más de abstracción para los desarrolladores. También se experimentó con el desarrollo de aplicaciones en una serie de escenarios para ambientes inteligentes.

Evaluación. En esta fase se evalúan las contribuciones de este trabajo para corroborar el desempeño y su factibilidad, así como la facilidad de aprendizaje y usabilidad.

1.4 Contribuciones

Como se ha mencionado, existe una gran diversidad de plataformas, herramientas y sistemas para apoyar la creación de aplicaciones en ambientes inteligentes. En algunos de estos proyectos incluso se utiliza Internet como plataforma en común como es el caso de CoolTown (Caswell y Debaty, 2000) ó XWeb (Olsen *et al.*, 2001). Sin embargo, la mayoría de los proyectos tienen dificultades en cuanto a la forma en la que se provee la abstracción a los desarrolladores. En algunos, las aplicaciones se deben realizar usando el ó los lenguajes de programación en los que el proyecto se encuentra originalmente implementado, y en casos extremos, los desarrolladores deben aprender un nuevo lenguaje de programación que es introducido por el mismo proyecto.

El presente trabajo de investigación introduce un marco de trabajo en el que la construcción de aplicaciones consiste en la unión de componentes que ofrecen funcionalidad básica y que en conjunto componen la funcionalidad deseada. Estos componentes, gracias a los beneficios de los estándares de la Internet y de las arquitecturas basadas en recursos, pueden ser implementados en cualquier lenguaje y residir en cualquier servidor o dispositivo por lo que también se propone un lenguaje extensible para describir su funcionalidad. Este trabajo difiere del realizado en el grupo de trabajo UWA en el sentido de que se está proponiendo el uso de la tecnología de Internet en general, no sólo la Web. Es decir, el grupo UWA se dedica en mejorar las aplicaciones Web al proponer nuevos clientes Web (y sus estándares de marcado de hipertexto correspondientes) con características mejoradas para dichos ambientes. Este trabajo, propone a la Web como medio de comunicación para componentes de *software* independientes de plataforma y de lenguaje de programación, incluso, se posibilita el soporte para aplicaciones legadas. Otra de las características importantes en las que se contribuye es en la integración de tecnologías de adquisición de contexto como lo son las redes inalámbricas de sensores y de mecanismos de actuación en la forma de motores y otros artefactos físicos. Finalmente, el marco de trabajo propuesto,

si bien fue enfocado a ciertos requerimientos especiales, puede ser aplicado con éxito en otros dominios de aplicación o incluso en el desarrollo de aplicaciones tradicionales, siempre y cuando éstas se puedan expresar en una arquitectura basada en recursos.

En resumen, las contribuciones de este trabajo son:

- Un marco de trabajo con una arquitectura basada en recursos para simplificar las tareas de composición de servicios y que incorpora notificaciones de eventos y la integración de entidades físicas en una aplicación (como proveedores de contexto y de actuación).
- Un modelo de interacción de los usuarios con los ambientes inteligentes basado en *mashups* que facilita la creación de aplicaciones y el consumo de funcionalidad.

1.5 Contenido de la tesis

El resto de este documento se organiza de la siguiente manera:

Capítulo 2: La Web de las Cosas. En este capítulo se describe a más detalle el concepto sobre el cual se basa este trabajo. También incluye una introducción de cada una de las tecnologías que se utilizaron para diseñar y finalmente realizar la implementación de las propuestas lo cual es importante ya que permite entender muchas de las decisiones de diseño y las diferencias con trabajos anteriores.

Capítulo 3: Los Ambientes Inteligentes. En este capítulo se describe el escenario para el que se enfocó el soporte de aplicación. La descripción también incluye los requerimientos especiales para este tipo de ambientes y se habla de requerimientos adicionales. Una vez que se describen los requerimientos, se analiza el trabajo relacionado.

Capítulo 4: UbiSOA Framework. Este es el capítulo principal de la tesis, en él se describen los elementos arquitectónicos del marco de trabajo, se discute su implementación y

el desarrollo de herramientas y sistemas adicionales que permiten extender o mejorar las contribuciones.

Capítulo 5: Escenarios y Experimentación. En este capítulo se describen algunos escenarios para los cuales se desarrollaron aplicaciones buscando corroborar si efectivamente se da soporte a los requerimientos.

Capítulo 6: Evaluación. Incluye un análisis de cómo fueron atendidos cada uno de los objetivos específicos para lo que se detallan las actividades realizadas entre las que se encuentra una evaluación con un grupo de usuarios potenciales. Esto sirvió para corroborar la facilidad de aprendizaje y la usabilidad. Además, se analiza cómo fueron cubiertos los requerimientos de diseño.

Capítulo 7: Conclusiones. Finalmente, se concluye con una discusión acerca del soporte de aplicaciones para ambientes inteligentes con el enfoque de la Web de las cosas así como también se menciona el trabajo futuro.

Apéndices. Incluyen material de referencia que apoya la lectura de este documento.

Capítulo 2

La Web de las Cosas

Fue apenas hace un poco más de 23 años que Sir Tim Berners-Lee inventó la World Wide Web. Desde entonces a la fecha, lo que inicialmente sería una forma de compartir documentos por Internet se ha convertido en toda una fuente global de conocimientos y cultura humana que permite a las personas compartir sus ideas, proyectos e incluso su vida diaria. Todo comenzó en la navidad de 1989 con las primeras páginas de Internet: documentos simples conformados por texto etiquetado que hacen referencia a documentos relacionados con más información (a esto se le conoce como hipertexto). Para consultar las páginas, se introdujeron los navegadores de Internet y con el tiempo estos fueron agregando elementos adicionales, como por ejemplo, la posibilidad de incluir imágenes en las páginas. Durante la fase inicial de Internet, la única forma de conocer acerca de la existencia de una página era o porque se llegó a ella a través de una referencia encontrada en otra página o porque se le conocía de antemano. Fue por ello que surgieron una serie de sitios Web que contenían listas de referencias a páginas agrupadas por categorías. Yahoo! fue uno de esos sitios. La segunda fase de la Web vino de la mano de sitios Web especiales con el objetivo de indexar y categorizar automáticamente el contenido de toda la Web. Si bien la tarea parece imposible, sobre todo debido a la explosión de contenidos y a que mucha de esa información es de naturaleza dinámica, aún es posible generar un índice de contenidos lo suficientemente grande y organizado como para satisfacer las necesidades de búsqueda de la gran mayoría de los usuarios. Fue durante esta segunda fase que sitios como Google hicieron su aparición. La tercera fase fue un producto de la penetración de la Internet en segmentos no técnicos de la población en general y a la facilidad de acceso a la misma. La Web que hasta ahora ofrecía contenidos en mayoría estáticos y de sólo lectura pasó a ser

una auténtica plataforma de aplicaciones con contenidos generados y administrados por los propios usuarios. Por lo mismo, el contenido empezó a ser de otra naturaleza, mucho más relacionada a la vida privada de las personas. Wikipedia, YouTube, Facebook y Twitter son ejemplos de algunos sitios que aparecieron en esta fase conocida por algunos como la Web social y por otros como la Web 2.0. Actualmente podemos decir que estamos en la cuarta fase: la Web móvil. Es hasta ahora que finalmente se empieza a observar una tendencia hacia la ubicuidad del cómputo. Los nuevos teléfonos inteligentes y las redes de telefonía celular nos dan la posibilidad de poder consultar los servicios y recursos de la Web en prácticamente cualquier lugar en donde nos encontremos. Entre otros factores, esta capacidad fue la que provocó el surgimiento de servicios en los que el contexto de los usuarios influye directamente en el contenido que se ofrece. Por ejemplo, existen sitios que basados en la ubicación geográfica del usuario ofrece recomendaciones de lugares en dónde comer. En esta fase hicieron su aparición plataformas para aplicaciones móviles tales como iOS (2012) de Apple y Android (2012) de Google. Muchos han opinado que la quinta fase será la Web semántica. Una Web en la que las páginas también describen el contenido, el significado y la relación de la información que contienen, lo que ayudaría a que los buscadores sean mucho más inteligentes a la hora de indexar y de generar respuestas a las consultas de los usuarios. Sin embargo, una tendencia que está siendo cada vez más clara es que tal vez la quinta fase sea la Web de las cosas.

En la Web de las cosas, los objetos de la vida diaria están representados en la Web en forma de páginas que pueden ser utilizadas para conocer la información referente al objeto o incluso para controlarlo. Uno de los grupos pioneros en trabajar en esta visión es el Auto-ID Center del MIT. El grupo fue creado en 1999 para desarrollar el EPC (EPCglobal, 2011), una tecnología basada en RFID pensada como un reemplazo de los sistemas de códigos de barras. Con EPC a los productos se les etiquetaría con un pequeño chip que además de la información de identificación contendría datos dinámicos tales como el lugar y fecha de fabricación, la fecha de vencimiento (en caso de productos perecederos), las dimensiones,

el peso, entre otros. Además de esta información, la etiqueta también serviría para hacer un seguimiento activo de la línea de producción y abastecimiento. Por ejemplo, con EPC sería posible conocer la existencia exacta y las condiciones de un producto en una bodega con tan sólo presionar un botón. En el 2003, el grupo fue reemplazado por Auto-ID Labs (2012), una red de grupos de investigación liderados por la organización EPCglobal (2012). El objetivo de esta red de investigación es mucho más ambicioso: diseñar la arquitectura para la Internet de las Cosas. Y la idea es la siguiente: etiquetar a todos los objetos del mundo con un minúsculo dispositivo de identificación y comunicación, formar una red que los interconecte a todos y permitir que sean identificados e inventariados por los sistemas de Internet de nueva generación (utilizando para ello, IPv6). Otro de los grupos trabajando en esta visión es la IPSO Alliance (2012), sólo que en lugar de etiquetas, ellos utilizan el concepto de objeto inteligente (*smart-object*), el cual es una pequeña computadora con capacidades de comunicación y un sensor o actuador. El objetivo de la alianza es el formar un grupo abierto de compañías para el mercadeo e investigación acerca de cómo utilizar IP para los objetos inteligentes. En resumen, se puede decir que la Internet de las Cosas busca el proveer a todos los dispositivos y objetos con los que interactuamos la habilidad de tener su propia dirección IP.

La Web de las Cosas es una visión inspirada en la Internet de las Cosas en la que se busca que los objetos inteligentes, además de tener su dirección IP, expongan su funcionalidad utilizando protocolos, estándares y otras tecnologías bien conocidas y aceptadas de la Web (HTTP, URI, REST, RSS, Atom, etc.). La Web de las cosas puede resumirse en 3 principios fundamentales:

- La utilización de HTTP como protocolo de aplicación y no como protocolo de transporte, como es el caso de los servicios Web tradicionales (SOAP, WS-*, etc).

- Expone la funcionalidad de los objetos inteligentes a través de interfaces de programación de tipo REST (véase la Subsección 2.2), es decir, respetando una arquitectura orientada a recursos.

- Expone la funcionalidad asíncrona (eventos) de los objetos inteligentes a través de estándares de suscripción de la Web tales como RSS, Atom u otros mecanismos basados en *push*.

Con el establecimiento de estos principios se busca que los servicios proveídos por los objetos inteligentes mantengan un bajo acoplamiento y que al mismo tiempo ofrezcan una interfaz uniforme para acceder y exponer la funcionalidad de los mismos.

2.1 Aplicaciones

Una de las primeras aplicaciones en aplicar los conceptos de la Web de las cosas fue el proyecto Energie Visible (Guinard *et al.*, 2009). El proyecto tiene como objetivo ayudar a las personas que quieren ahorrar en el consumo de energía de sus casas al permitirles conocer el consumo individual de cada uno de los aparatos eléctricos. Por ejemplo, comparar el consumo de una lámpara de halógeno contra una normal ó el de una computadora que se deja conectada al enchufe. Para utilizar el proyecto todo lo que un usuario debe hacer es comprar unos dispositivos llamados Ploggs (2012), unos enchufes que se conectan a la toma de las paredes y en los que a su vez se conectan los aparatos eléctricos. El dispositivo mide el consumo de energía y transmite constantemente las lecturas. El sistema Energie Visible opera en cada uno de los dispositivos, generando fuentes de datos que posteriormente son utilizados en un *mashup* visual (más detalles en la Subsección 2.4) para mostrar una página Web con el consumo de energía de la casa en tiempo real. El usuario entonces puede observar una gráfica con el historial de consumo de cada aparato así como el consumo en tiempo real (véase la Figura 2.1).

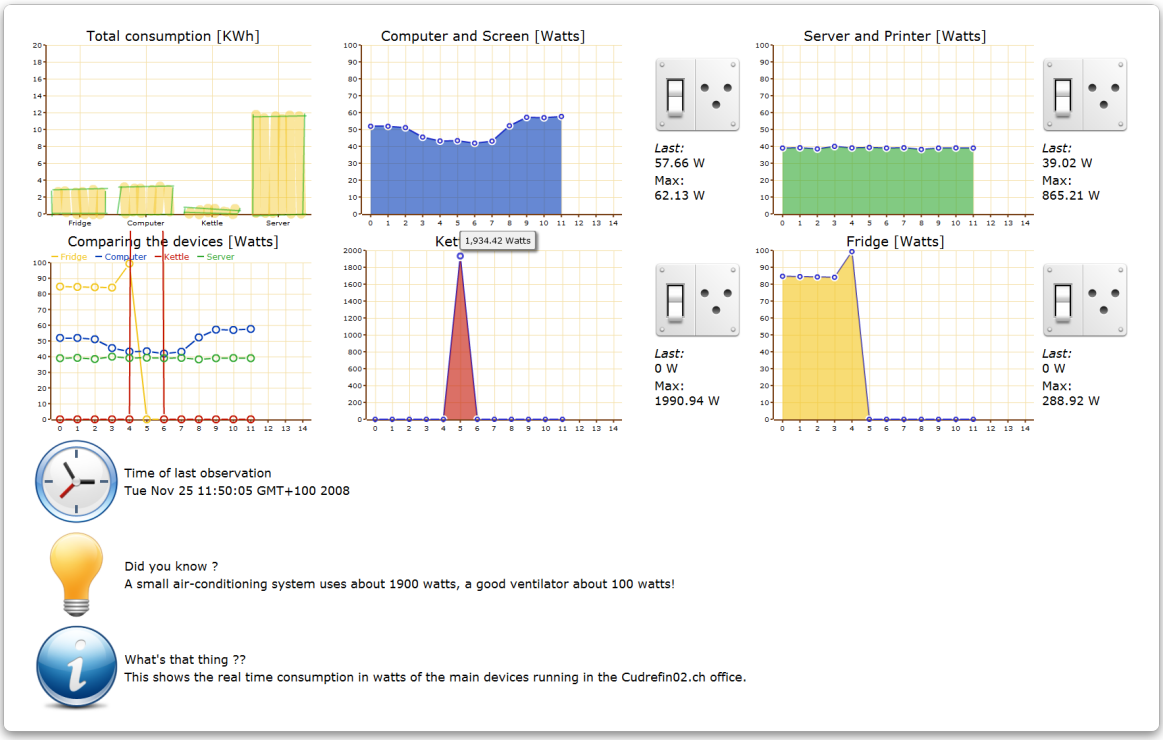


Figura 2.1: El sistema Energie Visible en funcionamiento. Por Guinard *et al.* (2009), recuperado de `http://www.webofthings.org/2009/02/11/energie-visible-a-sustainable-web-of-things-project` el 2 de diciembre de 2012.

Energie Visible no es el único proyecto trabajando con la visualización y administración del consumo de energía. Otros proyectos como Google PowerMeter (2012) y OpenEnergyMonitor (2012) también están enfocados en proveer la misma funcionalidad. Existen más proyectos enfocados en proveer otros tipos de sensores. Uno de los más característicos es Pachube/Cosm (2012), que tiene como objetivo el ser la plataforma de intercambio de datos para la Internet de las Cosas. En ella, individuos, organizaciones y compañías alrededor del mundo pueden almacenar, compartir y encontrar información sensorial de los objetos, dispositivos y edificios alrededor del mundo. La infraestructura de Pachube permite:

- *Administrar información ambiental y de sensado en tiempo real.* La plataforma de Pachube está conformada por miles de proveedores en donde las fuentes de datos

son almacenadas, convertidas y proveídas en múltiples formatos, lo que permite la interoperabilidad con los protocolos Web bien establecidos y los estándares industriales existentes. Las fuentes de datos también incluyen metadatos contextuales tales como la fecha y hora de actualización, la posición geográfica donde los sensores están ubicados, el tipo de unidades en la que los datos están almacenados, etiquetas que pueden ser utilizadas para categorizar las fuentes, entre otros.

- *Visualizar, monitorear y controlar remotamente a los ambientes.* Pachube permite incorporar gráficas y *widgets* (pequeños programas, en su mayoría de visualización) en las páginas Web de los usuarios. Las fuentes de datos pueden ser utilizadas para consultar información histórica de un sensor en específico o de varios en conjunto. Se pueden enviar alertas en tiempo real para que sea posible controlar los programas, dispositivos y ambientes de los usuarios. Entre las herramientas disponibles actualmente se encuentran visualizadores de gráficas, un *widget* para rastreo, un visor para realidad aumentada, alertas y aplicaciones para varios teléfonos inteligentes. En cuanto una fuente de datos es incorporada en Pachube, esta puede ser inmediatamente monitoreada y controlada por alguna herramienta o solución existente.

- *La construcción de aplicaciones móviles y sistemas Web.* Pachube ofrece una interfaz de programación físico-virtual que facilita y agiliza la construcción de aplicaciones que utilizan objetos inteligentes y ambientes en red. Además, la infraestructura de su plataforma hace posible la alta escalabilidad y disponibilidad necesarias para la administración de datos complejos. Adicionalmente, existen muchos ejemplos y librerías que facilitan aún más el desarrollo de aplicaciones y sistemas en la Internet de las Cosas.

- *Compartir datos y crear comunidades.* Pachube promueve la apertura de los sistemas, toda fuente de datos está disponible para cualquier usuario de la plataforma. Si bien, está planeada la incorporación de políticas de privacidad en versiones futuras de la

infraestructura, el ser un sistema abierto ha provocado la aparición de librerías de *software* y *hardware* creadas y mantenidas por la misma comunidad de usuarios.

En resumen, se puede ver a Pachube como una especie de YouTube en el que en lugar de compartir videos se comparten fuentes de datos de sensores que permiten a los usuarios monitorear y compartir información ambiental en tiempo real. El objetivo final de este proyecto es que la infraestructura se utilice no sólo para consumir información sino como una especie de puente entre ambientes inteligentes. Es decir, que una fuente de datos actúe como una entrada provocando una salida en otro ambiente (véase la Figura 2.2).

Pachube es un buen ejemplo de una aplicación de la Web de las cosas ya que las fuentes de datos son ofrecidas a través de una interfaz REST y en múltiples formatos de representación. Una fuente en Pachube puede ser consumida como un archivo CSV (*comma-separated values*, para ser usado en hojas de cálculo), JSON (un formato compacto para texto estructurado) y en XML. Para la representación en XML utilizan el lenguaje EEML (del inglés, Extended Environments Markup Language) (Haque Design + Research Ltd., 2008) que ha sido propuesto por la industria para representar información de sensado general.

Otro ejemplo de una aplicación que integra sensores y la Web en la vida diaria son los tenis Nike+ para correr (2012). El proyecto surgió como una colaboración entre Nike y Apple al darse cuenta de que muchas de las personas que suelen correr regularmente llevan consigo sus iPods para escuchar música. Nike lanzó una línea de tenis que cuentan con un compartimiento para alojar un sensor en forma de cápsula, este pequeño dispositivo comunica constantemente el número, la velocidad y el ritmo de los pasos de una persona de forma inalámbrica. Apple, por su parte, incorporó en sus últimos modelos de iPods y iPhones un receptor y entre ambos diseñaron una aplicación para estos dispositivos móviles que permite conocer el progreso de la carrera de los usuarios. Lo que es muy interesante del sistema Nike+ son las aplicaciones y servicios Web a los que el sensor en los tenis y las

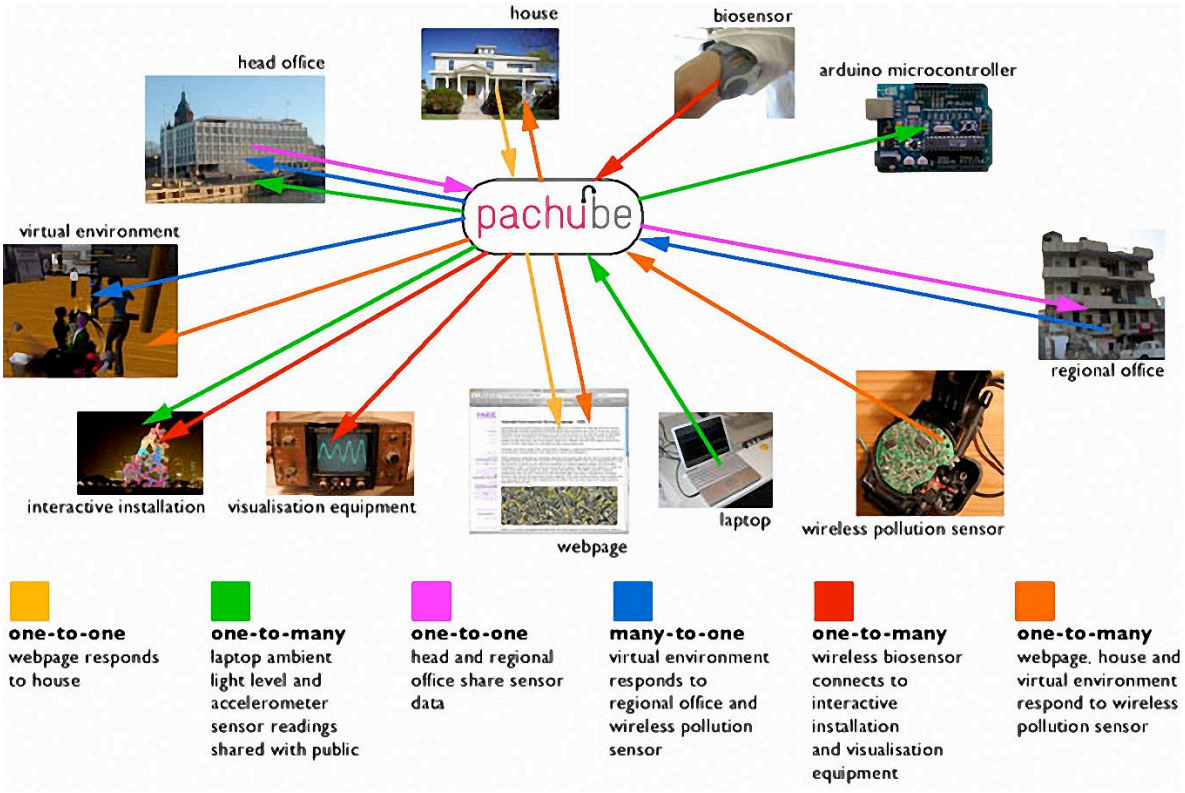


Figura 2.2: Posibles aplicaciones para la plataforma de Pachube. Por COSM Community (2012), recuperado de http://community.cosm.com/?q=node/1 el 2 de diciembre de 2012.

capturas de los dispositivos móviles se integran. La página Web de Nike+, permite a los usuarios planear un calendario con sus ejercicios, recibir consejos e incluso un programa de entrenamiento de acuerdo a su historial de ejercicios, competir con sus amigos para ver quién acumula más kilometraje o logra mejores tiempos de carreras, recibir medallas y estímulos en base a su progreso, entre otras muchas funciones. El sistema además no es cerrado, sino que se integra en las redes sociales de los usuarios. En las versiones recientes, al empezar a correr, el sistema envía una publicación a Facebook informando que el usuario acaba de empezar una sesión de ejercicio. Si uno de los amigos del corredor hace un comentario o marca a la publicación con un me gusta, el sistema informa inmediatamente al usuario reduciendo el volumen de la música, emitiendo el comentario en

Figura 2.3: La aplicación Nike+ GPS para iOS. Por Nike (2012), recuperado de `https://itunes.apple.com/mx/app/nike-gps/id387771637?mt=8` el 2 de diciembre de 2012.

audio, y subiendo el volumen de nuevo, como una forma de motivarlo a que continúe su ejercicio. Además de permitir la integración, la aplicación móvil puede ser utilizada para consultar el progreso actual o histórico de una carrera a través de gráficas, tablas e incluso con un mapa en el que se observa la ruta del ejercicio y en código de color la velocidad de la carrera (véase la Figura 2.3).

2.2 Arquitecturas orientadas a recursos

Las arquitecturas orientadas a recursos son aquellas en las que la funcionalidad no está representada como métodos o funciones de servicios individuales sino como recursos con direcciones únicas que son proveídos por las distintas entidades de un sistema. El concepto de orientación a recursos se origina en los principios implementados por la arquitectura REST. La transferencia de estado representacional (del inglés, Representational State Transfer) o simplemente REST, es un estilo arquitectónico para construir sistemas distribuidos Web a gran escala. Fue introducido por Roy Fielding (2000) en una tesis doctoral sobre la

Web y está muy ligado a la especificación del protocolo HTTP. REST difiere de otras tecnologías de servicios Web como SOAP, al utilizar directamente HTTP como protocolo de aplicación, es decir, la interacción con interfaces de servicios REST se basan en solicitudes HTTP simples y en la comunicación de documentos en formatos extensibles como XML. Servicios como SOAP, utilizan HTTP sólo para transportar documentos XML que contienen, además de la información solicitada, datos sobre abstracciones adicionales que deben ser posteriormente interpretados. Los principios de diseño de REST son cuatro: primero, cada recurso debe tener un identificador global único (URI); segundo, todos los servicios deben ser accesibles a través de un conjunto de operaciones bien definidas (las operaciones básicas de HTTP); tercero, el uso de representaciones de datos extensibles y estándares para compartir información (por lo general, HTML o XML); y cuarto, ni el cliente ni el servidor mantienen el estado de la aplicación, cada mensaje HTTP contiene toda la información necesaria para la operación descrita. A continuación se detallan cada uno de los principios:

2.2.1 Identificación global de recursos

REST utiliza URI (Mealling y Denenberg, 2002) para identificar a cada recurso. Los URIs son cadenas de caracteres utilizadas para referenciar una identidad o recurso en Internet. Un URI (Uniform Resource Identifier) puede ser clasificado como un localizador (haciendo referencia a la ubicación de un recurso) o como un identificador (haciendo referencia a la descripción del recurso), o a veces, como ambas. Cuando se utiliza como localizador, toma el nombre de URL (Uniform Resource Locator) y de URN (Uniform Resource Name) como identificador. En la mayoría de los sistemas REST, los URI utilizados son URLs, los cuales tienen la siguiente sintaxis:

```
<esquema> : <jerarquía> [? <consulta>] [# <fragmento>]
```

Esquema se refiere al protocolo utilizado para interpretar el URL, en este caso, HTTP. **Jerarquía** contiene la información para identificar el recurso que se solicita, la cual está

conformada por dos partes: **autoridad** y **ruta**. **Autoridad** contiene el usuario y la contraseña del recurso (si es que está protegido), la dirección y el puerto del servidor. **Autoridad** tiene la siguiente sintaxis:

```
[usuario:contraseña @] dirección [: puerto]
```

Ruta contiene la información necesaria para localizar el recurso que se solicita al servidor. Un ejemplo de un URL con un **esquema** y **jerarquía** sería el siguiente:

```
http://usuario:clave@numenor.cicese.mx:8324/alumnos
```

Consulta y **fragmento** son opcionales y no deben ser utilizados de ninguna manera para identificar recursos. Sin embargo, pueden ser utilizados para filtrar la información solicitada. Por ejemplo:

```
http://usuario:clave@numenor.cicese.mx:8324/alumnos?nombre=Juan#egresados
```

La identificación de recursos en REST es similar a la forma en la que los objetos son identificados en un sistema usando el paradigma de orientación a objetos. Es decir, el sistema está dividido en conceptos que son importantes para el sistema y que encapsulan toda la información necesaria. La responsabilidad sobre qué nombres utilizar recae en el diseñador del sistema por lo que es importante seleccionar nombres que sean descriptivos por sí mismos. Ejemplos:

- Una lista de libros: `http://servidor/libros`
- El libro con un ISBN específico: `http://servidor/libros/isbn/950-07-0320-3`
- Los libros en una categoría: `http://servidor/libros/categoria/drama`
- Una lista de todos los autores: `http://servidor/autores`
- Los libros de un autor: `http://servidor/autores/garcía-márquez`
- Los libros más comprados: `http://servidor/libros/más-comprados`

Tabla 2.1: Descripción de las operaciones CRUD.

CRUD	SQL	HTTP	Descripción
Create	INSERT	**POST**	Crea un nuevo recurso
Retrieve	SELECT	**GET**	Solicita un recurso
Update	UPDATE	**PUT**	Actualiza un recurso
Delete	DELETE	**DELETE**	Elimina un recurso

Tabla 2.2: La interfaz de un servicio REST para libros.

Recurso	Descripción	POST	GET	PUT	DELETE
`/libros`	Obt. una lista de libros o crea uno.	✓	✓	✗	✗
`/libros/isbn/#`	Obtiene, actualiza o borra un libro.	✗	✓	✓	✓
`/libros/categoria/#`	Obt. los libros de una categoría.	✗	✓	✗	✗
`/libros/autores`	Obt. una lista de libros por autor.	✗	✓	✗	✗
`/libros/autores/#`	Obt. los libros de un autor.	✗	✓	✗	✗

- Los autores más populares: `http://servidor/autores/más-populares`

2.2.2 Interfaz uniforme a los servicios

Los recursos en REST son manipulados a través de solicitudes de HTTP dirigidas al URI de los recursos con los que se desee interactuar y las acciones corresponden al método HTTP que se utilice en la solicitud. Las interfaces de los servicios REST siguen el modelo de las interfaces CRUD. Las operaciones Create, Retrieve, Update y Delete son las cuatro operaciones principales de cualquier sistema de almacenamiento persistente. Por ejemplo, en las bases de datos las operaciones básicas son INSERT, SELECT, UPDATE y DELETE, siguiendo el mismo modelo, en HTTP las operaciones básicas son POST, GET, PUT y DELETE. La Tabla 2.1 resume estas operaciones.

Cualquier recurso debe ser accesible a través de los cuatro métodos de HTTP. Sin embargo, la implementación de alguno de los métodos puede ser opcional. Por ejemplo, cuando un recurso sea de sólo lectura las operaciones DELETE y PUT no tendrían efecto.

La diferencia con los servicios SOAP aquí es más clara. Por ejemplo, un sistema que almacene libros, si estuviera implementado con SOAP, algunos métodos de ese sistema

Tabla 2.3: La interfaz REST de un servicio contra una tradicional.

Descripción	REST	Tradicional
Obtiene una lista de libros.	`GET /libros`	`obtLibros()`
Crea un libro.	`POST /libros`	`agrLibro(…)`
Actualiza un libro.	`PUT /libros/isbn/#`	`actLibroPorISBN(…)`
Obtiene un libro.	`GET /libros/isbn/#`	`obtLibroPorISBN(…)`
Borra un libro.	`DELETE /libros/isbn/#`	`borrLibroPorISBN(…)`
Obt. los libros de una cat.	`GET /libros/categoria/#`	`obtLibrosPorCategoria(…)`
Obt. los libros por autor.	`GET /libros/autores`	`obtLibrosPorAutor()`
Obt. los libros de un autor.	`GET /libros/autores/#`	`obtLibrosPorAutor(…)`

serían `agregarLibro()` y `eliminarLibro()` mientras que en REST, para agregar un libro se tendría que enviar solicitud POST a un URL `/libros` y para eliminarlo habría que enviar una solicitud DELETE al URL del libro `/libros/isbn/950-07-0320-3`. A menudo cuando se diseña el API de un servicio REST, se realiza una tabla CRUD de la interfaz. Por ejemplo, en la Tabla 2.2 se detalla la interfaz REST para el sistema de administración de libros y en la Tabla 2.3 se compara con la de una interfaz tradicional.

2.2.3 Múltiples representaciones por recurso

Así como se puede encontrar la misma película en formato DVD, Blu-ray o digital, así una arquitectura REST debería ofrecer sus recursos en múltiples formatos. Cuando se envía una solicitud al URI de un recurso, la solicitud HTTP contiene una serie de parámetros en su encabezado que especifica datos tales como la identificación del cliente que está enviando la solicitud. Uno de esos encabezados indica el formato en el que la representación del recurso deberá ser devuelta. A esto se le conoce como negociación de contenidos. Algunas implementaciones podrían utilizar otros modelos de negociación, por ejemplo, en lugar de utilizar el encabezado requerir que el URI contenga el formato de la representación como parte de la cadena de consulta. En la Subsección 2.3 se describen los formatos de representación más utilizados en la Web.

2.2.4 Cambio de estados por hipermedia

El término Hipermedia se refiere a un sistema de cómputo de obtención de información que permite a los usuarios obtener o proveer acceso a texto, audio, video, fotografías y otros medios relacionados a un tema en particular de manera interactiva (Nelson, 1965). Una característica primordial de este tipo de sistemas es que es un medio no lineal de información, es decir, el usuario a través de su interacción con el medio puede explorar los contenidos libremente al ir seleccionando vínculos entre ellos. El formato de hipermedia más utilizado en la Web es el hipertexto.

Uno de los principios más importantes de REST es el que dice que el flujo de la aplicación deberá ser especificado a través del seguimiento del hipermedia que se intercambia durante las solicitudes cliente/servidor. Es decir, ni el servidor ni el cliente deben llevar un seguimiento del estado de una aplicación, limitándose para estos propósitos el seguir las referencias que se regresen durante las solicitudes. Con esto se respeta la naturaleza *stateless* de la Web. Por ejemplo, un sistema REST que permita transferir el monto de la cuenta de un cliente a otra del mismo cliente en un mismo banco. Para iniciar la interacción, se haría una solicitud al URI que nos permite conocer una lista con las cuentas de un cliente:

```
http://www.banco.com/cuentas/cliente/7T676323A
```

El sistema debería responder con lo siguiente:

```
1 HTTP/1.1 200 OK
2 Content-Type: application/xml
3 Content-Length: 620
4
5 <?xml version=”1.0” encoding=”UTF-8”?>
6 <cuentas xmlns=”urn:org:banco:cuentas”>
7    <cuenta>
8       <id>AZA12093</id>
9       <cliente>7T676323A</cliente>
```

```
10        <monto>993.95</monto>
11        <link href="http://www.banco.com/cuentas/id/AZA12093" rel="self" />
12        <link href="http://www.banco.com/transferencias" rel="transferencias" />
13     </cuenta>
14     <cuenta>
15        <id>ADK31242</id>
16        <cliente>7T676323A</cliente>
17        <monto>534.62</monto>
18        <link href="http://www.banco.com/cuentas/id/ADK31242" rel="self" />
19        <link href="http://www.banco.com/transferencias" rel="transferencias" />
20     </cuenta>
21  </cuentas>
```

La primera línea, que contiene el código de respuesta HTTP, es lo más importante para un sistema REST. En este caso, el código es 200, que corresponde a OK, lo que nos hace saber que la solicitud fue exitosa y que en el cuerpo de la respuesta (líneas 5–21) viene incluida una representación del recurso solicitado (véase la Tabla 2.4). La línea 2 nos dice el formato de la representación. En este caso, XML.

Como se trata de una interfaz REST, la representación debería tener referencias a otros recursos que están disponibles a partir de la interacción inicial. En este caso, se tiene como respuesta información acerca de dos cuentas de banco para el cliente número 7T676323A (nodos XML en las lineas 7–13 y 14–20).

A su vez, el nodo XML de cada cuenta contiene 2 referencias. Por ejemplo, se tiene una referencia al URL `http://www.banco.com/cuentas/id/AZA12093` (línea 11) y se especifica una relación semántica tipo *self* para la misma. La relación semántica nos dice qué se puede hacer con el URL. En este caso, *self* indica que se puede encontrar una representación con más información sobre la cuenta #AZA12093 en esa dirección si quisiéramos saber por ejemplo el balance de la cuenta. La segunda referencia (línea 12), indica una relación transferencias, así que ése es el URL al que se le deben enviar solicitudes para realizar transferencias de una cuenta a otra. Como lo que se desea es transferir $100.00 de la cuenta #AZA12093 a la #ADK31242, se envía la siguiente solicitud:

```
POST /transferencias HTTP/1.1
Host: www.banco.com
Content-Type: application/xml
Content-Length: 187

<?xml version="1.0" encoding="UTF-8"?>
<transferencia xmlns="urn:org:banco:cuentas">
   <origen>AZA12093</origen>
   <destino>ADK31242</destino>
   <monto>100.00</monto>
</transferencia>
```

A lo que el servicio debería responder con el código 202 (Accepted), con lo que la transferencia se empezaría a procesar. Si existió algún problema, el código debería dar una idea de lo que pudo ser la causa. Por ejemplo, si el servicio regresara 404, alguno de los números de cuenta podría ser incorrecto, si regresa 503, se tendría que esperar un poco y volver a intentar la solicitud más adelante.

Como se puede observar, el seguimiento del cambio del estado de la aplicación es una combinación de los códigos de estado de respuesta a la solicitud enviada y del contenido de las representaciones devueltas (referencias semánticas a otros URIs). Como es de esperarse, los valores para el atributo *rel* que indica la semántica de la referencia, están estandarizados. La IANA (Internet Assigned Numbers Authority) mantiene un documento con la explicación acerca del significado de cada uno de los valores para dicho atributo (IANA, 2012). En el caso en que los valores recomendados no representen adecuadamente la semántica de la aplicación, es responsabilidad del diseñador de la interfaz del servicio escoger los valores que se utilizarán para denotar la funcionalidad (en el ejemplo anterior, el valor transferencias no es uno de los recomendados por la IANA).

Tabla 2.4: Códigos de estado para respuestas comunes.

Código	Descripción	Significado
200	OK	La solicitud fue exitosa. La repr. fue enviada al cliente.
201	Created	El URI del nuevo recurso va incluido en la respuesta.
202	Accepted	La solicitud fue exitosa, el servidor empezará a trabajar.
204	No Content	La solicitud fue exitosa, la respuesta está vacía.
300	Multiple Choices	El recurso solicitado tiene más de una opción válida.
301	Moved Permanently	El recurso se ha movido a un nuevo URI permanentemente.
302	Found	El recurso reside temporalmente en un URI diferente.
304	Not Modified	El recurso no ha cambiado desde la última solicitud.
307	Temporary Redirect	El recurso se ha movido a un nuevo URI temporalmente.
400	Bad Request	La solicitud es inválida.
401	Unauthorized	El cliente debe autentificarse.
403	Forbidden	El servidor entendió la solicitud pero no la puede realizar.
404	Not Found	El recurso no existe en el sistema.
405	Method Not Allowed	El método no está soportado.
410	Gone	El recurso ya no está disponible y no se conoce su nuevo URI.
500	Internal Server Error	El servidor encontró un error inesperado.
501	Not Implemented	El servidor no soporta la funcionalidad requerida.
503	Service Unavailable	El recurso no está disponible temporalmente.
550	Permission Denied	El cliente no tiene los privilegios necesarios.

2.3 Formatos de representación

Actualmente existe una gran diversidad de formatos de representación de recursos en la Web. Sin embargo, hay una clara preferencia en la utilización de sólo cuatro de ellos: HTML, XML, JSON y las fuentes de datos en Atom. En esta sección se introduce brevemente cada formato y para ayudar en la comparación, se utiliza como ejemplo la representación del mismo recurso en cada uno. El recurso representado es una lista de artículos que tienen como únicos atributos: título y contenido.

2.3.1 Hypertext Markup Language (HTML)

El propósito de este formato de representación en los servicios REST, es comúnmente para que funcione como una interfaz gráfica a la funcionalidad completa que ofrece el servicio o recurso al que representa. Los documentos HTML, personalizados, con hojas de

estilo CSS e interactividad implementada con JavaScript son desarrollados para ser utilizados por usuarios humanos (contrario a formatos de interoperabilidad diseñados para ser consumidos por otros sistemas de *software*). Por ejemplo, el siguiente documento HTML es el que es devuelto al enviar la solicitud GET a un servicio que mantiene un listado de artículos. En la Figura 2.4 se puede observar el mismo documento interpretado por un navegador Web. Como la página debe demostrar la funcionalidad del servicio, se incluye un formulario para enviar solicitudes POST y agregar nuevos artículos al servicio.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4     <meta charset="UTF-8" />
5     <title>UbiSOA Framework - Publisher Test</title>
6     <link rel="stylesheet" href="http://yui.yahooapis.com/3.3.0/build/cssreset/
          reset-min.css" />
7     <link rel="stylesheet" href="http://api.ubisoa.net/css/general.css" />
8  </head>
9  <body>
10    <header>
11       <a href="http://www.ubisoa.net/"></a>
12       <span>Services</span>
13    </header>
14    <div id="wrapper">
15       <div id="content-wrapper">
16          <h1>Publisher Test</h1>
17          <p class="subtitle">This is a test for the push protocol.</p>
18          <form class="column" method="POST">
19             <h2>Post New Item</h2>
20             <input type="text" name="title" placeholder="Title" required
                   autofocus /><textarea name="content" placeholder="Content"
                   required></textarea><input type="submit" value="Post Item" />
21          </form>
22          <div class="column">
23             <h2>Published Items</h2>
24             <ul>
25                <li><strong>Nan Vestibulum Arcu.</strong> Nam Vestibulum, Arcu
                      Solades Feugiat Consectetur, Nisl Orci Bibendum Elit.</li>
26                <li><strong>Praesent Eget Neque.</strong> Duis Aliquet Egestas
                      Purus In Blandit. Curabitur Vulputate, Ligula Lacinia.</li>
```

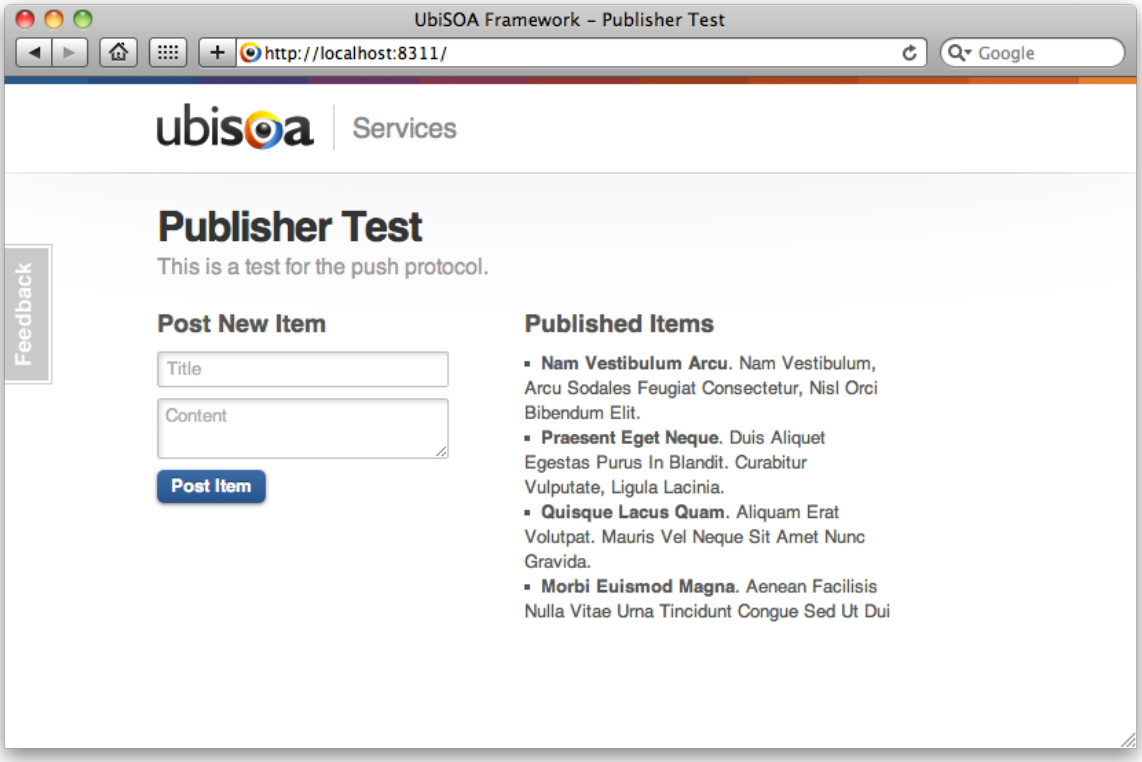


Figura 2.4: Ejemplo de representación HTML de un servicio REST.

```
27                <li><strong>Quisque Lacus Quam.</strong> Aliquam Erat Volutpat
                      Mauris Vel Neque Sit Amet Nunc Gravida.</li>
28                <li><strong>Morbi Euismod Magna.</strong> Aenean Facilisis Nulla
                      Vitae Urna Tincidunt Congu Sed Ut Dui.</li>
29             </ul>
30          </div>
31       </div>
32    </div>
33
34    <script src="http://s3.amazonaws.com/getsatisfaction.com/javascripts/
          feedback-v2.js"></script>
35    <script src="http://api.ubisoa.net//js/getsatisfaction.js"></script>
36    <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js
          "></script>
37 </body>
38 </html>
```

2.3.2 Extended Mark-Up Language (XML)

XML surgió como un lenguaje de etiquetado para codificar documentos en una manera fácil de interpretar por sistemas de *software*. Debido a su extensibilidad y maleabilidad, se ha utilizado para incluso definir nuevos lenguajes de programación, no sólo para describir información como era su propósito. La intención de que los servicios ofrezcan una representación en XML para sus recursos es principalmente la interoperabilidad con otros servicios. El siguiente ejemplo es un documento XML que contiene exactamente la misma información que muestra la Figura 2.4.

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <items>
3     <item>
4        <title>Nam Vestibulum Arcu</title>
5        <content>Nam Vestibulum, Arcu Solades Feugiat Consectetur, Nisl Orci
             Bibendum Elit.</content></item>
6     <item>
7        <title>Praesent Eget Neque</title>
8        <content>Duis Aliquet Egestas Purus In Blandit. Curabitur Vulputate,
             Ligula Lacinia.</content></item>
9     <item>
10       <title>Quisque Lacus Quam</title>
11       <content>Aliquam Erat Volutpat Mauris Vel Neque Sit Amet Nunc Gravida.</
             content></item>
12    <item>
13       <title>Morbi Euismod Magna</title>
14       <content>Aenean Facilisis Nulla Vitae Urna Tincidunt Congu Sed Ut Dui.</
             content></item>
15 </items>
```

2.3.3 JavaScript Object Notation (JSON)

JSON es un lenguaje para el intercambio de datos basados en la serialización de objetos en JavaScript. Es un lenguaje relativamente sencillo, el cuál sólo tiene dos estructuras: arreglos y listas de elementos llave-valor. Recientemente se ha utilizado como alternativa a

XML ya que los documentos codificados con JSON son mucho más pequeños que en XML; además, al ser una serialización de objetos, para interpretar un documento JSON los intérpretes sólo deben realizar una conversión mínima para convertir documentos en objetos. JSON se utiliza sobre todo en los sistemas Web modernos, en los que mucha de la funcionalidad está implementada en JavaScript y donde la interpretación de archivos XML representa una mayor desventaja. A continuación se muestra un documento JSON que contiene la misma información que en el ejemplo de XML anterior.

```
1  {
2      ”items”: [{
3          ”title”: ”Nam Vestibulum Arcu”,
4          ”content”: ”Nam Vestibulum, Arcu Solades Feugiat Consectetur, Nisl Orci
               Bibendum Elit.”
5      }, {
6          ”title”: ”Praesent Eget Neque”,
7          ”content”: ”Duis Aliquet Egestas Purus In Blandit. Curabitur Vulputate,
               Ligula Lacinia.”
8      }, {
9          ”title”: ”Quisque Lacus Quam”,
10         ”content”: ”Aliquam Erat Volutpat Mauris Vel Neque Sit Amet Nunc
              Gravida.”
11     }, {
12         ”title”: ”Morbi Euismod Magna”,
13         ”content”: ”Aenean Facilisis Nulla Vitae Urna Tincidunt Congu Sed Ut
              Dui.” }]
14 }
```

Otra de las ventajas de este formato es en el cómo se accede a la información una vez que se ha interpretado el documento. En el ejemplo anterior, para obtener el título del primer artículo sólo se tendría que escribir `items[0].title`. Se promueve el uso de JSON en plataformas con poca capacidad de cómputo, o de recursos limitados tales como las redes inalámbricas de sensores.

2.3.4 Atom Syndication Format (Atom)

Atom es uno de los formatos para codificar fuentes de datos Web. Una fuente de datos es una lista de elementos (comúnmente noticias) que los sitios Web utilizan para hacer un seguimiento de la publicación de contenidos nuevos. Los servicios que ofrecen esta representación además de XML, suelen introducir una copia del nodo XML del elemento que está descrito en Atom como una forma de preservar información. Retomando el ejemplo de la lista de artículos con título y contenido, las noticias de un archivo Atom están representadas por los artículos disponibles en el servicio y el contenido de la noticia es directamente el contenido del artículo. Adicionalmente a esto, también se incluye el nodo de la representación XML de cada elemento. Otra ventaja de esta representación es que existen muchos clientes de fuentes Web que permiten suscribirse a estas representaciones (véase la Figura 2.5). A continuación se puede ver un ejemplo de documento Atom.

```
1  <?xml version=”1.0” standalone=’yes’?>
2  <feed xmlns=”http://www.w3.org/2005/Atom”>
3     <entry>
4        <content type=”text/plain”>Nam Vestibulum, Arcu Solades Feugiat
             Consectetur, Nisl Orci Bibendum Elit.</content>
5        <id>urn:uuid:5d700446-ca3d-4301-89dc-22ea562670e0</id>
6        <title type=”text”>Nam Vestibulum Arcu</title>
7        <ubs:item>
8           <title>Nam Vestibulum Arcu</title>
9           <content>Nam Vestibulum, Arcu Solades Feugiat Consectetur, Nisl Orci
                Bibendum Elit.</content>
10       </ubs:item>
11    </entry>
12    <entry>
13       <content type=”text/plain”>Duis Aliquet Egestas Purus In Bla...
14       <id>urn:uuid:9f119758-d5ef-4d5d-b7a0-987c1e371403</id>
15       <title type=”text”>Duis Aliquet Egestas Purus In Blandit. Curabitur
             Vulputate, Ligula Lacinia.</title>
16       <ubs:item>
17          <title>Praesent Eget Neque</title>
18          <content>Duis Aliquet Egestas Purus In Blandit. Curabitur Vulputate,
                Ligula Lacinia.</content>
```

```
19      </ubs:item>
20   </entry>
21   <entry>
22      <content type="text/plain">Aliquam Erat Volutpat Mauris Vel Neque Sit
            Amet Nunc Gravida.</content>
23      <id>urn:uuid:1d085d29-4e54-494d-8e65-86d6f7fac73e</id>
24      <title type="text">Quisque Lacus Quam</title>
25      <ubs:item>
26         <title>Quisque Lacus Quam</title>
27         <content>Aliquam Erat Volutpat Mauris Vel Neque Sit Amet Nunc Gravida.
               </content>
28      </ubs:item>
29   </entry>
30   <entry>
31      <content type="text/plain">Aenean Facilisis Nulla Vitae Urna Tincidunt
            Congu Sed Ut Dui.</content>
32      <id>urn:uuid:a99812e9-b436-414e-a2b5-a420e3a7eaca</id>
33      <title type="text">Morbi Euismod Magna</title>
34      <ubs:item>
35         <title>Morbi Euismod Magna</title>
36         <content>Aenean Facilisis Nulla Vitae Urna Tincidunt Congu Sed Ut Dui.
               </content>
37      </ubs:item>
38   </entry>
39 </feed>
```

2.4 Sistemas *mashups*

Durante la Web 2.0 se observó una explosión en el desarrollo de sistemas en la Web. Sistemas, que además dejaron de estar diseñados exclusivamente para el consumo humano. Los sistemas Web 2.0 están diseñados para que sus contenidos puedan ser leídos y navegados tanto por humanos como por sistemas de cómputo especializados (muy a menudo referidos como robots). Para promover la interoperabilidad, los nuevos sitios Web han abierto sus sistemas y funcionalidad en la forma de servicios Web o fuentes de datos abiertas. Lo que ha provocado que la Web se haya convertido en una plataforma simple y flexible para la programación distribuida.

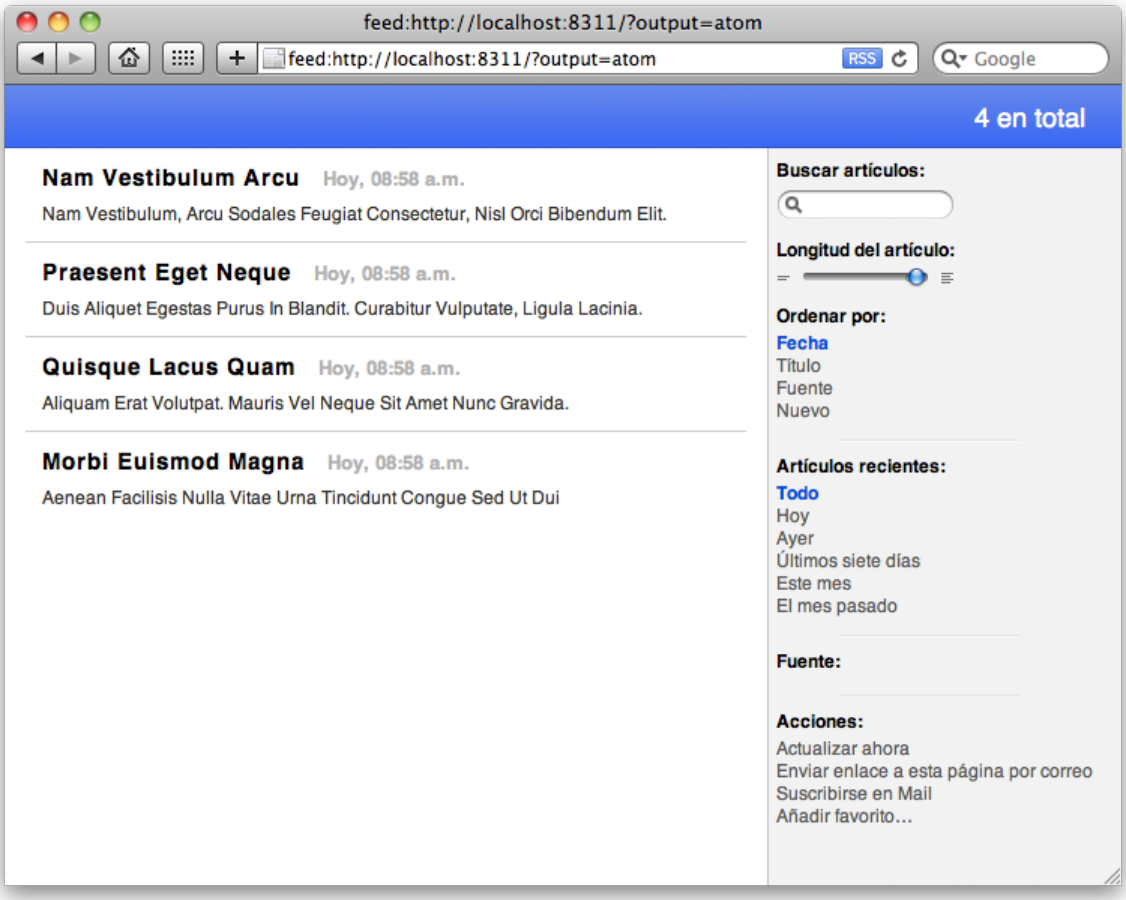


Figura 2.5: Ejemplo de representación Atom de un servicio REST.

En los últimos años se ha observado un interés creciente en aplicaciones Web integradas, popularmente conocidas como *mashups* (Zang *et al.*, 2008). Los *mashups* son la fusión de diferentes fuentes de datos e interfaces de programación (APIs) en una experiencia de usuario integrada. El concepto de *mashup* se originó en la música, donde el término denota la práctica de tomar dos o más canciones existentes y crear una nueva pieza al reordenar, intercalar y superponer partes de estas fuentes (Hartmann *et al.*, 2008). El término fue adoptado en las ciencias de la computación para referirse a aplicaciones creadas al programar usando una o más interfaces de programación abiertas, a los que también se les refiere como servicios de infraestructura (Brewer, 2001).

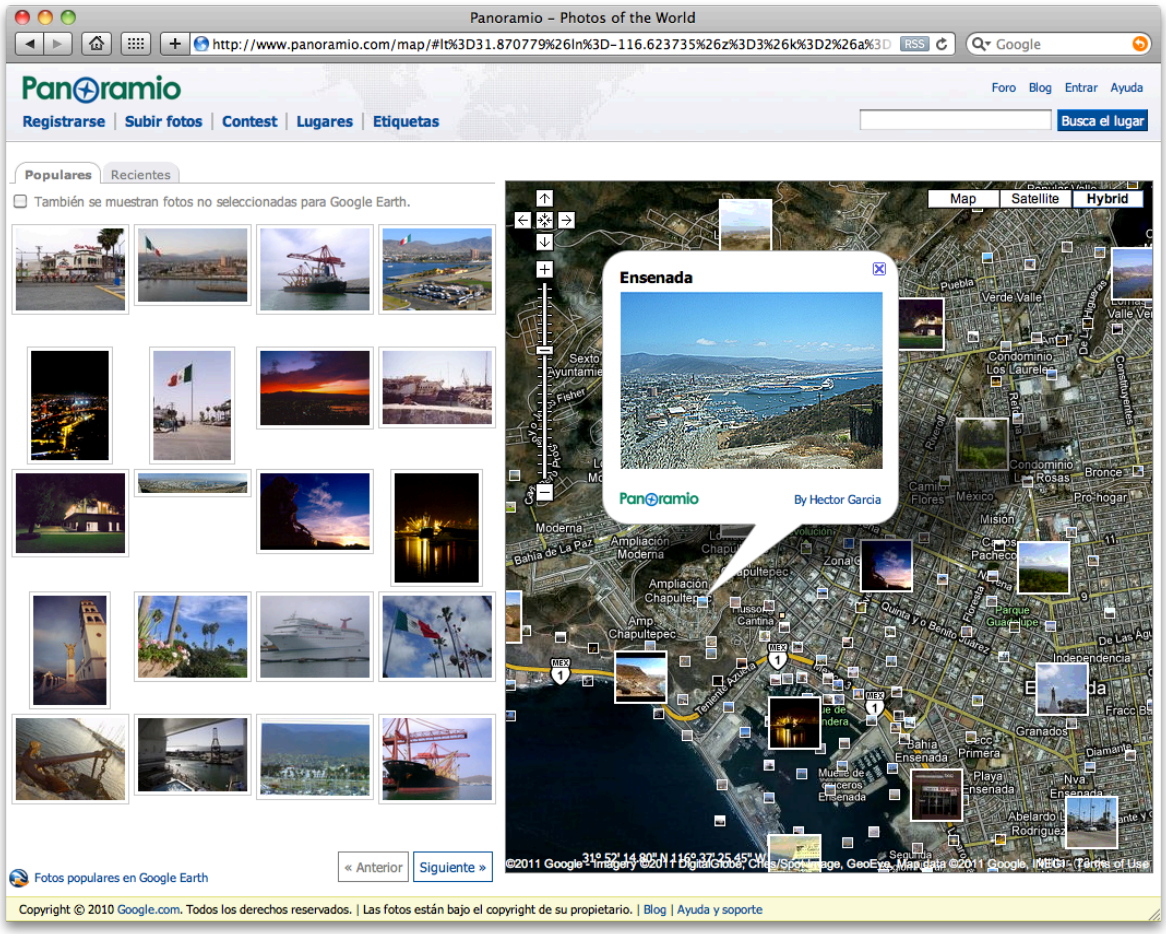


Figura 2.6: Panoramio, un ejemplo de uso muy común de un *mashup*.

Un ejemplo muy común de una aplicación *mashup* es la superposición de datos, fotos o videos en un mapa. Panoramio (2012) es uno de esos tantos servicios. Su funcionalidad está basada en un *mashup* de Google Maps y una fuente de fotos geolocalizadas. Los usuarios de este servicio comparten sus fotos indicando a qué lugar geográfico corresponden (si es que la foto no contiene ya, datos de geolocalización). De esta manera, es posible hacer búsquedas de fotos para una cierta región de un mapa (véase la Figura 2.6). Recientemente, Panoramio fue adquirido por Google que a su vez lo utiliza para mostrar fotos relacionadas a una vista de su servicio Street View.

La idea general de un *mashup* es que la integración de datos y funcionalidad sea lo más sencilla posible, incluso sin necesidad de habilidades de programación o de un esfuerzo significativo. Originalmente, la idea de los *mashups* era utilizada generalmente para combinar datos o componentes visuales pero recientemente ha sido aplicada en la composición de procesos o actividades. Esta evolución de los *mashups* tiene el objetivo de superar las limitaciones de las tecnologías actuales para composición de servicios Web. Por ejemplo, BPEL (siglas en inglés para Business Process Execution Language) (Andrews *et al.*, 2003) es uno de los lenguajes de composición más utilizados para servicios Web pero su uso requiere habilidades sustanciales de programación, el diseño de las composiciones debe estar pensado para ser ejecutadas sólo del lado del servidor y por lo mismo, no ofrecen soporte para procesos centrados en el usuario ya que son escenarios muy flexibles donde la personalización del lado del cliente es importante y necesaria.

Una de las características distintivas de los *mashups* es que a menudo están conformados por agregaciones muy sencillas cuya existencia está limitada a un usuario en particular, es decir, no se trata de aplicaciones permanentes. Por lo mismo, no necesitan la definición de procesos complejos, como los descritos en los lenguajes de composición tradicionales. El usuario debería poder realizar composiciones simples y ligeras que soporten el ensamblado transitorio y sencillo de fuentes e interfaces de programación Web.

Para dar una mejor idea del panorama de los sistemas *mashups*, Albinola *et al.* (2009) hacen una clasificación ortogonal de tres dimensiones (véase la Figura 2.7):

1. La primera dimensión es la naturaleza de los *mashups* que son generados por el marco de trabajo o herramienta. Se puede distinguir entre *mashups* de **datos**, **lógicos** y de **presentación**.

    - Los ***mashups* de datos** están principalmente centrados en la combinación de datos de diferentes fuentes. Ejemplos de marcos de trabajo para *mashups* de

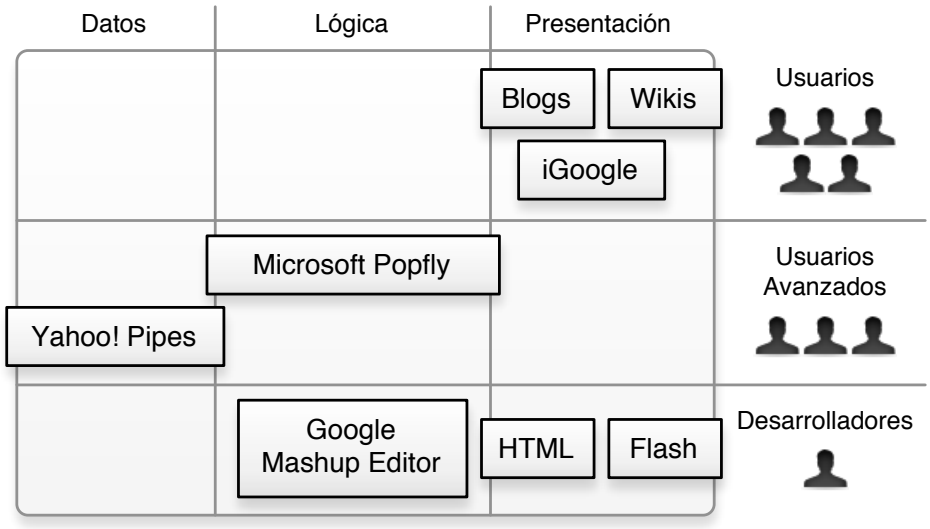


Figura 2.7: Las dimensiones de un sistema *mashup* según Albinola *et al.* (2009).

datos son Yahoo! Pipes (2.4.1) y Google Mashup Editor (2.4.2), que combinan y generan fuentes RSS y JSON.

- Los ***mashups* lógicos** proveen los medios para permitir que diferentes servicios colaboren con su lógica hacia un objetivo en común. El ejemplo más claro de este tipo de marcos de trabajo es Microsoft Popfly (2.4.2).
- Los ***mashups* de presentación** no combinan ni datos ni lógica sino que solamente presentan datos o servicios tomados de diferentes fuentes usando una interfaz común. El mejor ejemplo de este tipo de *mashups* es iGoogle (2.4.3).

2. La segunda dimensión es el tipo de los usuarios capaces de utilizar el marco de trabajo para resolver un tipo de problema creando un *mashup*. Se puede distinguir entre **Desarrolladores**, **Usuarios Avanzados** y, simplemente, **Usuarios**.

   - Los **Desarrolladores** son las personas que tienen conocimientos profundos en las tecnologías de la Web, que cuentan con habilidades de programación y que

pueden resolver al problema al nivel más bajo de abstracción. Este tipo de usuarios utilizan los *mashups* como herramientas para programar más rápido.

- Los **Usuarios Avanzados** no tienen habilidades avanzadas de programación, pero entienden el problema tecnológico y están cómodos con un nivel de abstracción intermedio. Además, están conscientes de aspectos tales como las diferencias entre formatos de representación, las ventajas y desventajas de un cierto protocolo, problemáticas no funcionales que podrían ocurrir al ejecutar las aplicaciones, entre otros.
- Los **Usuarios** son personas que sólo tienen conocimiento del problema a un nivel aplicativo y que no tienen ningún conocimiento tecnológico.

3. La tercera dimensión se refiere al cómo y al dónde se ejecuta un *mashup*. Se puede distinguir entre *mashups* de **cliente**, **servidor** e **híbridos**.

    - Los *mashups* del lado del **cliente** utiliza los recursos del navegador Web del usuario. El ambiente de ejecución está basado en JavaScript, XML, JSON e idealmente en HTML 5.
    - Los *mashups* del lado del **servidor** se ejecutan en servidores especializados en donde pueden hacer uso de tecnologías más avanzadas y de muchos más recursos que los que hay disponibles del lado del cliente. En este caso, debido a que el flujo de ejecución de los *mashups* se mantiene por el servidor, el diseño del marco de trabajo debe seguir de cerca los principios del diseño de servicios, por ejemplo, no mantener el estado.
    - Los *mashups* **híbridos** combinan las ventajas y desventajas de clientes y servidores y ofrecen un ambiente de ejecución más robusto. Generalmente, los *mashups* son combinados por los servidores y descargados para su ejecución en los clientes, de donde podrían surgir más solicitudes al servidor para mantener el flujo de datos.

2.4.1 Mashups de datos

Yahoo! Pipes (2012) es una aplicación Web de Yahoo! que provee una interfaz gráfica para construir *mashups* de datos que agregan fuentes, páginas y otros servicios basados en la Web. Además de permitir la creación, también funciona como servidor de las aplicaciones creadas por los usuarios. Para crear un *mashup* con Yahoo! Pipes, el desarrollador debe diseñar una tubería (*pipe*) de datos y especificar reglas acerca de cómo se debe modificar el contenido en cada paso.

En la Figura 2.8 se muestra un ejemplo de tubería. Se utilizan dos puntos de salida, una búsqueda Web en Yahoo! con los términos *Noticias Mocorito* y una búsqueda de fotos en Flickr con el término *Mocorito*. En realidad, estas fuentes son representaciones de datos devueltas por servicios Web que son actualizadas cada vez que se solicita la salida de la tubería. Ambos puntos de entrada, se conectan con una operación unir que generará una sola fuente RSS con la agregación de todas las publicaciones que encuentre en las fuentes de entrada. Finalmente, la salida de esta operación se conecta a la salida de la tubería. La aplicación final (la tubería) se almacena en los servidores de Yahoo! y entonces está disponible para cualquier cliente que la solicite (incluso otras tuberías).

2.4.2 Mashups de lógica

Google Mashup Editor (2012) era un servicio de creación de *mashups* de Google. A diferencia de otras herramientas, este editor estaba diseñado para su uso por desarrolladores con experiencia en programación de aplicaciones Web. Por lo mismo, el sistema carecía de editores visuales que permitan, como en el caso de Yahoo! Pipes, arrastrar y soltar representaciones visuales para indicar cómo se llevará a cabo la composición (véase la Figura 2.9). En Google Mashup Editor, el desarrollador debía utilizar JavaScript, HTML, CSS y algunas etiquetas especiales diseñadas para indicar operaciones y otras tareas específicas para el editor. Los *mashups* desarrollados con este editor tomaban la forma de *gadgets* (una es-

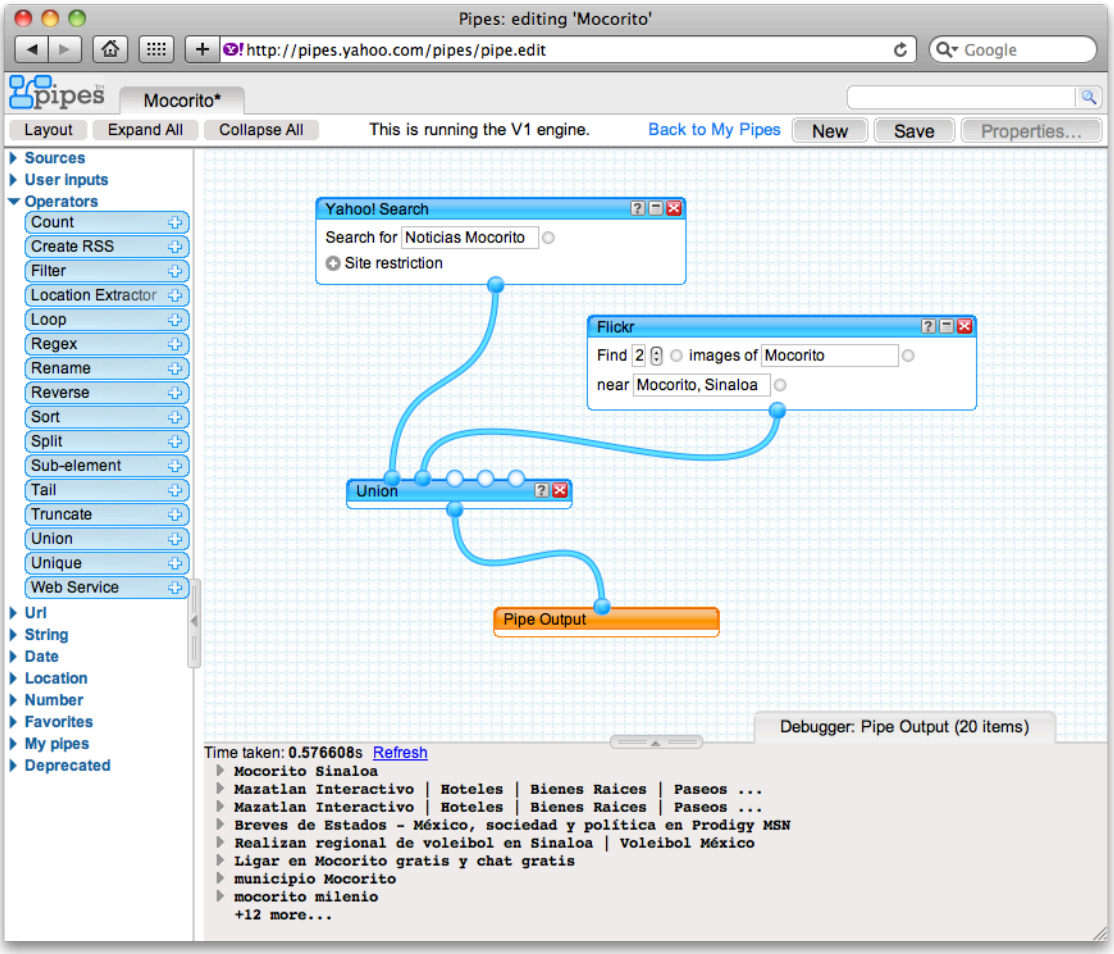


Figura 2.8: Ejemplo de un *mashup* en Yahoo! Pipes.

pecie de *widgets* definidos en un documento XML) los cuales podían ser incluidos en otras páginas Web al copiar y pegar un código especial que se generaba cuando se almacenaba un *mashup*. Adicionalmente, los *gadgets* podían ser utilizados en iGoogle, otro servicio de Google.

A principios de 2009, el Google Mashup Editor fue cancelado por motivos financieros y la funcionalidad del marco de trabajo fue incorporada en Google Apps Engine, un servidor de aplicaciones en el que se pueden desarrollar no sólo *mashups*, sino aplicaciones Web de propósito general en Java o Python.

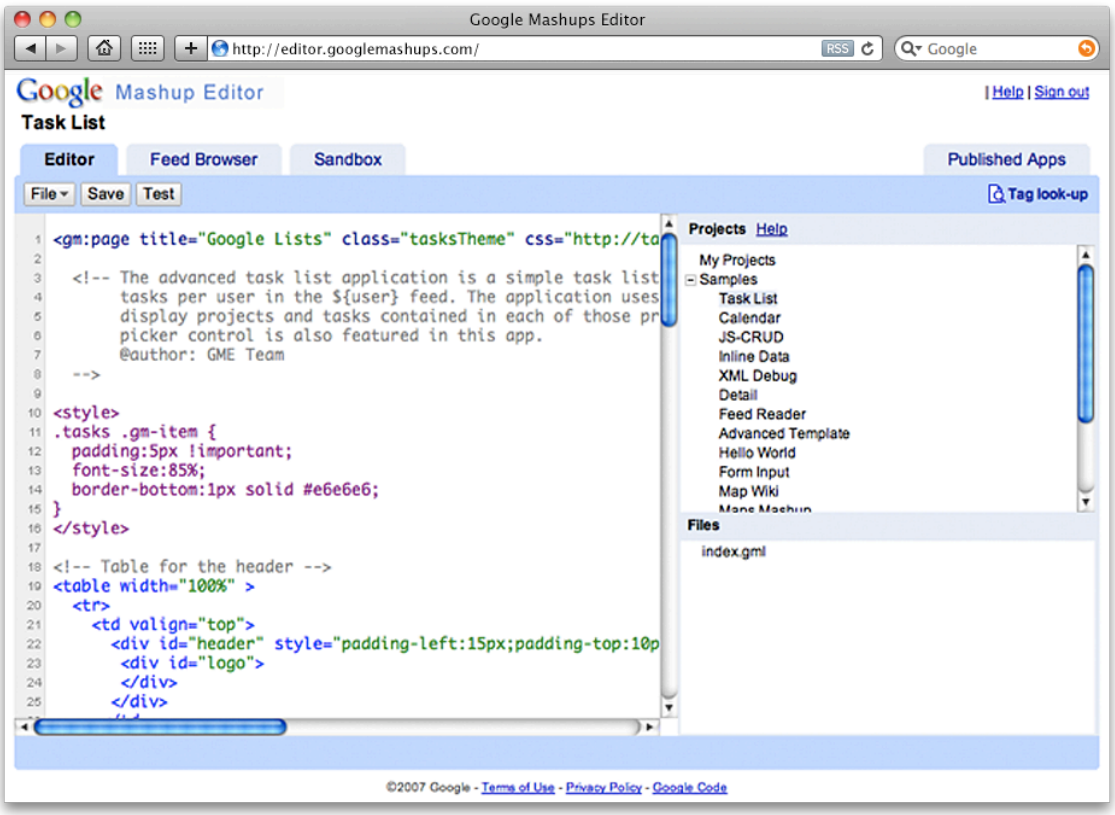


Figura 2.9: Google Mashup Editor, el servicio de creación de *mashups* de Google.

Microsoft Popfly (2012) fue un sitio Web que permitía a los usuarios crear páginas Web, porciones de sistemas Web y *mashups* lógicos y visuales utilizado la plataforma Silverlight. El editor de mashups de Popfly era una herramienta totalmente visual en donde bloques prefabricados de funcionalidad podían ser combinados en diferentes servicios Web y herramientas de visualización. Por ejemplo, en la Figura 2.10 se pueden observar dos puntos de partida, un bloque que representa fotos de una cuenta de Flickr y un bloque que representa la extracción de las fotos en una página Web. De estos dos bloques, la salida está conectada a un bloque combinador que genera una nueva fuente de datos con la agregación de las entradas, esta nueva fuente es entonces enviada a un bloque carrusel que representa a un *widget* visual que muestra las fotos en la fuente en una visualización 3D, con las fotos girando una a una. Adicionalmente, los usuarios podían utilizar un panel de

Figura 2.10: El editor de *mashups* de Microsoft Popfly.

edición HTML y JavaScript para especificar funcionalidad mucho más compleja que la que se podía representar gráficamente. Este servicio sufrió una suerte similar a la de Google Mashup Editor y fue cancelado en Julio de 2009. A pesar de ello, aún sigue siendo una referencia importante cuando se trata acerca de los *mashups* visuales gracias al nivel de abstracción tan alto que permitía.

2.4.3 Mashups de presentación

iGoogle (2012), un servicio de Google, es una página de inicio personalizable basada en Ajax. Permite a los usuarios agregar gadgets y fuentes Web en una visualización organizada

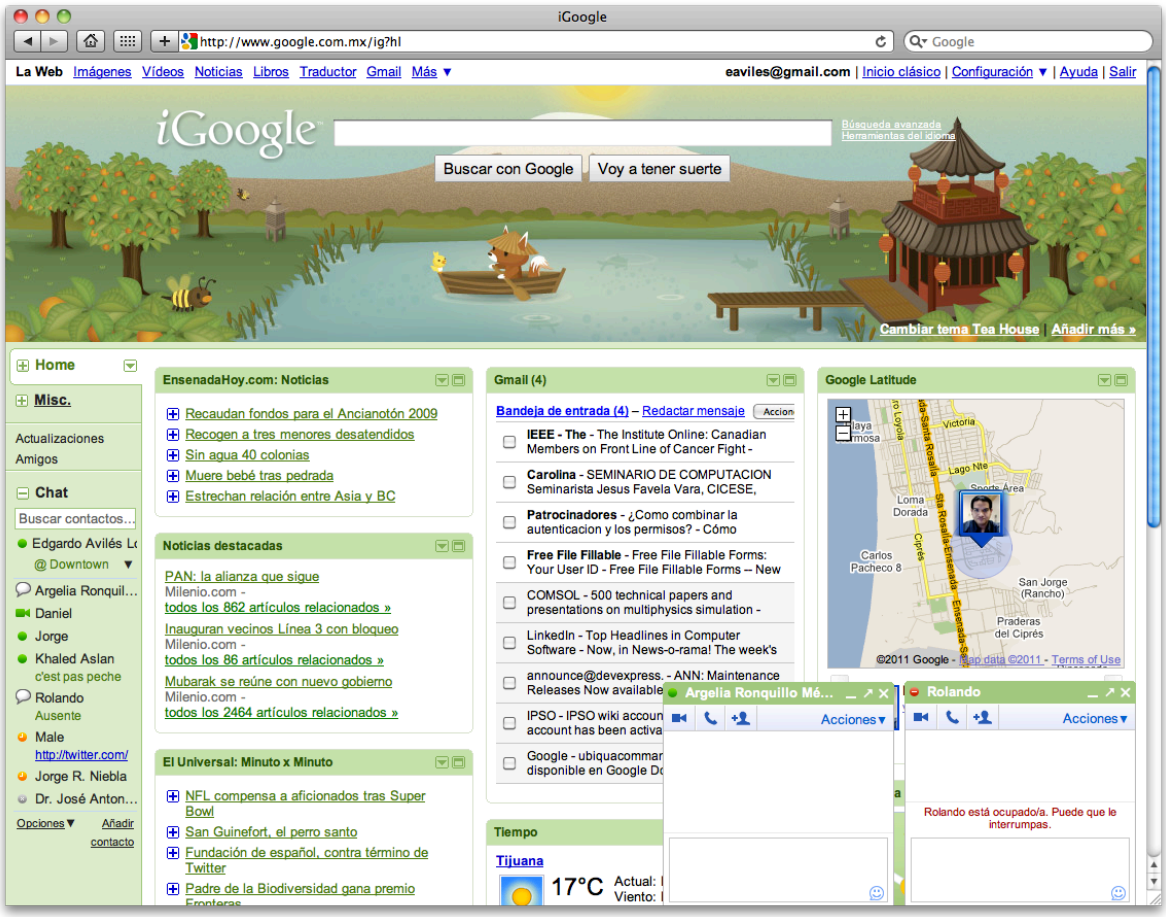


Figura 2.11: Ejemplo de configuración de iGoogle.

por pestañas y columnas. Para lo que primero se debe programar un *gadget* utilizando un API especial (basado en HTML y JavaScript que se ejecuta en servidores) o se puede utilizar uno de los *gadgets* existente en la biblioteca de iGoogle. Un *gadget* puede ser desde una lista con las publicaciones recientes de una fuente Web hasta una miniaplicación de correo electrónico.

2.5 Conclusiones

Los conceptos y tecnologías introducidos en este capítulo son la base de las aportaciones del presente trabajo de tesis, su comprensión es importante ya que dan una idea de qué tan básicas y qué tan limitantes son las herramientas que se utilizaron en la resolución de los problemas. El siguiente capítulo trata acerca de las características y requerimientos especiales de los ambientes inteligentes, a los que se les debía dar soporte con las herramientas descritas en este capítulo.

Capítulo 3

Los ambientes inteligentes

La visión del cómputo ubicuo define un mundo aumentado lleno de dispositivos inteligentes que conforman una red distribuida de alta granularidad (Weiser y Brown, 1997). La inteligencia ambiental (o AmI por sus siglas en inglés, Ambient Intelligence) va más allá al estudiar cómo hacer que los ambientes de cómputo sean sensitivos, adaptables y que respondan a la presencia de personas, con el objetivo de ayudar en el proveer soporte a sus experiencias de la vida diaria (Aarts *et al.*, 2002).

El surgimiento de nuevos tipos de dispositivos de cómputo móvil, los recientes avances en comunicación inalámbrica, sensores inteligentes y otros, nos proveen de las herramientas y métodos para proveer nuevas alternativas y herramientas innovadoras para mejorar la asistencia de los usuarios, y por lo mismo, mejorar su estilo de vida al ayudarles a mantener un cierto grado de independencia (Kleinberger *et al.*, 2007).

3.1 Características y requerimientos

El marco de trabajo propuesto en el presente trabajo de tesis, así como cualquier infraestructura que busque ayudar en el desarrollo de aplicaciones para el dominio AmI debe proveer un soporte efectivo a las características comunes exhibidas por sus aplicaciones (Weber *et al.*, 2005) y también a sus requerimientos no funcionales (Nehmer *et al.*, 2006). Dichos requerimientos se pueden resumir de la manera siguiente:

- *Invisibilidad.* Cualquier dispositivo o servicio que sea utilizado para aumentar el ambiente debe estar totalmente escondido a los usuarios. Es decir, dar la impresión de que son invisibles. Por ejemplo, un escenario en el que se necesite rastrear e identi-

ficar automáticamente a los usuarios. Para resolverlo, podría existir la necesidad de equipar de alguna manera al usuario con algún dispositivo que ayude a proveer el mecanismo de rastreo e identificación. En este caso, el dispositivo que se les entregue a los usuarios podría tomar la forma de un reloj, una pulsera, un par de lentes, botones en sus ropas, etc.

- *Movilidad.* En los escenarios de AmI los usuarios no están limitados a un solo ambiente o área física. El usuario transita constantemente entre ambientes aumentados, a veces incluso, llevando consigo parte del procesamiento requerido para su asistencia. Para resolver esto, se podría aprovisionar al usuario con una computadora portátil, un celular u otro dispositivo especializado que debe ser portado todo el tiempo. El propósito principal es hacer que el dispositivo represente al usuario proveyendo o consumiendo servicios en la infraestructura según como sea instruido por las necesidades de las tareas activas.

- *Consciencia de contexto.* Todo espacio aumentado debe estar equipado con sensores y de una infraestructura de comunicación inalámbrica con la que sea posible monitorear y estar al tanto de información de sensado útil para la tarea que se está llevando a cabo. Una vez que la información de contexto cruda ha sido adquirida, el ambiente debe proveer los medios para que se pueda modelar el contexto con el propósito de proveer contexto derivado, más complejo, que deberá ser entregado y compartido a las entidades interesadas en el sistema.

- *Anticipatorio.* Los sistemas de vida asistida deben ser capaces de actuar en representación de sus usuarios sin requerirles que hagan una solicitud implícita de acción. Lo contrario a requerir que el usuario utilice un teclado, un ratón u otro medio para indicar una acción.

- *Comunicación natural.* Los usuarios deben comunicarse e interactuar con el ambiente inteligente por medio de los dispositivos y objetos a los que están acostumbrados

en su vida diaria. Además, se debe recordar que los sistemas AAL son utilizados en su mayoría por tres grupos de personas: las personas a las que se está asistiendo, el personal médico (o de asistencia) y el personal de mantenimiento (técnicos). Cada grupo de personas tiene sus propios requerimientos de interacción humano-computadora. El sistema debe estar consciente de esto y proveer un modelo de interacción apropiado para cada usuario.

- *Adaptables.* Los sistemas deben estar diseñados de una manera tal que puedan resolver anormalidades o situaciones excepcionales al adaptarse a sí mismos en tiempo de ejecución tan autónomamente como sea posible Oppermann *et al.* (1997). Para apoyar esto, los sistemas deben efectuar un monitoreo continuo buscando condiciones críticas para que puedan realizar una autooptimización, una autoconfiguración o un automantenimiento, según sea necesario. Adicionalmente, la arquitectura de dichos sistemas deben estar diseñadas de tal manera que nuevos escenarios de aplicación, dispositivos y tecnologías puedan ser incorporados o actualizados según vayan cambiando con el tiempo.

- *Robustez y Disponibilidad.* El sistema debe continuar su ejecución y continuar proveyendo su funcionalidad a pesar de cualquier mal uso, error o incluso con *hardware* problemático o recursos de red inestables.

- *Extensibilidad.* El diseño de los sistemas debe permitir la incorporación de nuevos componentes o recursos en tiempo de ejecución. Es decir, se debe permitir que un sistema en ejecución soporte un nuevo tipo de recurso sin tener la necesidad de dejar de proveer la funcionalidad que el sistema está ofreciendo.

- *Seguridad.* Si se encuentra una excepción en tiempo de ejecución, esta no debería provocar que se entregue una respuesta con un resultado incorrecto ni tampoco provocar una falla considerable que detenga la ejecución del sistema entero. Es decir, el sistema debe hacer exactamente el trabajo para el cual fue diseñado.

- *Protección.* Si el sistema utiliza información personal como un recurso, este debe garantizar un buen grado definido de privacidad para las personas bajo observación.

- *Oportunidad.* A pesar de que los sistemas AAL no pueden ser considerados sistemas en tiempo real de manera estricta, existen casos en los que retrasos o demoras en la entrega de información deben ser evitados o al menos, mantenidos a un mínimo.

- *Eficientes.* Los sistemas deben administrar y utilizar los recursos de manera inteligente. Recursos tales como el ancho de banda, procesamiento, memoria, etc.

Adicionalmente, este trabajo está enfocado de manera importante en proveer abstracciones de alto nivel y herramientas fáciles de utilizar por parte de los desarrolladores, su motivación principal es facilitar la creación de prototipos transitorios. Por lo mismo, se ha seguido el siguiente conjunto de principios de diseño inspirados por Greenberg (2009) y Albinola *et al.* (2009).

- Ser agnóstico al lenguaje de programación, delegando la responsabilidad de la selección de un lenguaje al desarrollador. De esta manera, se pueden utilizar el mismo conocimiento, las mismas habilidades y las mismas herramientas a las que el desarrollador está habituado.

- Eliminar barreras de implementación de bajo nivel al proveer alternativas sencillas de alto nivel para realizar lo mismo.

- Minimizar las tareas de mantenimiento y tareas no esenciales que podrían incluso no estar relacionadas a la aplicación.

- Encapsular conceptos de diseño y patrones que han comprobado su efectividad en objetos que puedan ser incluidos en los sistemas con muy poco esfuerzo de implementación.

- Presentarse a sí mismos a través de una interfaz de programación simple y concisa que promueva y siga la forma en las que las personas piensan acerca del dominio de aplicación a mano, permitiendo que se enfoquen en la aplicación en lugar de las problemáticas de bajo nivel relacionadas en la realización de las mismas.
- Hacer que las cosas simples se puedan realizar con pocas líneas de código.
- Dar soporte a la creación de agregaciones de servicios simples cuya existencia esté limitada a un conjunto particular de usuarios por un periodo limitado de tiempo (ensamblajes transitorios).
- Los resultados deben ser utilizables inmediatamente para que los desarrolladores puedan experimentar con diferentes alternativas para sus objetivos.

Finalmente, en los ambientes inteligentes existe la necesidad siempre constante de construir prototipos. Los prototipos a menudo están enfocados en escenarios muy específicos que tratan de manipulaciones de datos muy simples que pueden, por lo mismo, ser modelados en *mashups*. Con el objetivo de promover la investigación, los desarrolladores podrían estar interesados en compartir los *mashups* para que los resultados puedan ser replicados y mejorados a su vez. El término aplicaciones de situación (Blau *et al.*, 2009) hace referencia a esta necesidad. Las aplicaciones de situación son aplicaciones que nunca alcanzan un estado final y que son utilizadas solamente para probar una idea y compartir soluciones. Estos requerimientos adicionales también están considerados.

3.2 Trabajos relacionados

Los trabajos analizados a continuación presentan algunas similitudes a las contribuciones de este trabajo de tesis. Sin embargo, difieren en cuanto a los requerimientos para el soporte de ambientes inteligentes y del tipo de aplicación al que están enfocados.

3.2.1 Sentient Graffiti

López-de-Ipiña *et al.* (2007) presenta Sentient Graffiti (SG), una plataforma siguiendo el concepto de la Internet de las Cosas en las que los usuarios llevan consigo dispositivos móviles con los que pueden navegar, descubrir, buscar, anotar y filtrar los objetos inteligentes a su alrededor en la forma de recursos Web. En esta infraestructura, los usuarios utilizan un cliente que pueden utilizar para agregar anotaciones a objetos en regiones espaciales.

Mientras que este trabajo incorpora requerimientos de cómputo ubicuo, especialmente, los de los sistemas de ambientes inteligentes, considera a los servicios como entidades pasivas, requiriendo que el usuario inicialice explícitamente la interacción con los objetos inteligentes.

3.2.2 Social Devices

En Vazquez y López-de-Ipiña (2008), los autores presentan la implementación de un par de prototipos para evaluar el potencial de su marco de trabajo para ofrecer el soporte a la integración de los objetos inteligentes en la Web. Introducen el concepto de dispositivo social (social device) y proponen una infraestructura para lograr que las acciones realizadas con los objetos sean integradas en redes sociales en la Web. Nuestro trabajo difiere en que en nuestro marco de trabajo los objetos pueden interactuar entre sí; además, en que integra no sólo objetos, sino también usuarios con el modelo de interacción basado en *mashups* como se explica en capítulos posteriores.

3.2.3 Mashlight

Albinola *et al.* (2009) presentan Mashlight, un marco de trabajo para *mashups* que asemejan a aplicaciones Web en forma de *widgets*. Mashlight permite la creación de *mashups* de datos, lógica y presentación a través de los conceptos de bloques y procesos. Los bloques consisten de unidades independientes de funcionalidad y están inspiradas en los

widgets presentes en los sistemas operativos modernos (Mac OS X y Windows 7). Internamente, un bloque representa a un servicio Web SOAP o partes de una interfaz gráfica de usuario. Los bloques están conectados a través de parámetros de entrada, salida y de la ejecución de una aplicación. Una vez que los bloques están configurados y listos para ser ejecutados, estos son administrados a través de procesos que controlan el flujo de una aplicación *mashup*. Para ayudar al desarrollador, el marco de trabajo incluye un constructor de bloques, una librería de bloques y un editor de *mashups*.

Mientras que esta propuesta tiene éxito en proveer una abstracción de alto nivel para el desarrollador, esta no está diseñada para dar soporte a ambientes inteligentes o ubicuos. Primero, los bloques en una aplicación *mashup* deben conocerse en tiempo de diseño. Los ambientes ubicuos son muy dinámicos así que es muy difícil conocer a priori qué servicios estarán disponibles en el ambiente puesto que los servicios van a aparecer o desaparecer conforme el usuario se mueve físicamente en el ambiente. Segundo, los bloques en un flujo no pueden comunicarse de manera asíncrona para, por ejemplo, reportar notificaciones cuando existan nuevos datos. Tercero, las aplicaciones de Mashlight sólo pueden correr en navegadores basados en Webkit así que las aplicaciones que corren en un ambiente aumentado y que no están en un dispositivo administrado por el usuario no son posibles.

3.2.4 Mashups físicos

Guinard y Trifa (2009) analizan la integración de dispositivos del mundo real en la Web al transformarlos en recursos REST. Los autores proponen dos métodos de integración. En el primero, describen cómo un servidor Web actual puede ser implementado en dispositivos de gama baja; en el segundo, cuando los recursos son muy limitados, proponen el uso de una entidad intermediaria que pueda ofrecer un API unificado basado en REST a los dispositivos al esconder los protocolos de comunicación que realmente se utilizan para comunicarse con ellos. Adicionalmente, los autores ejemplifican cómo los servicios pueden ser utilizados luego en aplicaciones *mashups*.

Nuestras contribuciones van más allá de sólo proveer acceso a las entidades físicas o de facilitar el desarrollo de mashups a partir de recursos REST. El diseño de nuestro marco de trabajo toma en cuenta requerimientos especiales para el soporte de ambientes inteligentes. Entre alguno de esos requerimientos están: la necesidad de comunicar notificaciones cuando un recurso o contexto cambia, la posibilidad de ejecutar una aplicación en el fondo de un ambiente, la incorporación de servicios genéricos en las composiciones para las cuales el servicio a ser utilizado se desconoce, la posibilidad de involucrar la entrada explícita de un usuario en el flujo de aplicación, entre otras que se analizan con más detalle en el siguiente capítulo.

3.2.5 Advanced Internet of Things

Zhang y Mitton (2011) presentan el paradigma denominado Advanced Internet of Things (AIoT) basado en el lenguaje unificado de descripción de objetos o UODL (*unified object description language*) que es también propuesto por los mismos autores. UODL permite identificar e interconectar objetos inteligentes y eventos en un formato estandarizado poniendo disponible toda la información y eventos relacionados a los objetos, además, permite integrar sistemas de control de terceros fácilmente.

El paradigma AIoT y nuestro marco de trabajo ofrecen ambos mecanismos para identificar objetos, hacer búsqueda de servicios y emplazamiento pero la diferencia principal es en la forma en la que todos los objetos están modelados. Nuestro marco de trabajo está basado en una arquitectura REST y cada uno de sus elementos está representado por un servicio que provee recursos a través de solicitudes HTTP simples. Debido a esto, cualquier servicio de terceros existente que use HTTP como protocolo de aplicación puede ser fácilmente integrado. De la misma manera, nuestro marco de trabajo puede utilizar directamente las mismas tecnologías para seguridad y privacidad de datos que están disponibles para la Web tradicional.

Finalmente, AIoT no involucra al usuario en las aplicaciones, nuestro marco de trabajo está diseñado para permitir que los usuarios creen o modifiquen aplicaciones siguiendo el concepto de *mashups*.

3.2.6 Wisdom Web of Things

Esfuerzos anteriores como los de la Internet de las Cosas y la Web de las cosas se han enfocado en la integración de los dispositivos físicos de la vida diaria con la Internet como la plataforma en común para adquisición e integración de datos. Recientemente, la pregunta de qué hacer con los datos que tales dispositivos generan y de cómo administrarla para que generen datos más ricos está empezando a ganar importancia. Los servicios de siguiente generación en la Internet están formando las bases para un nuevo "hipermundo" en el que las cosas en el mundo físico, las computadoras y los datos del mundo cibernético y los humanos en el mundo social todos están convergiendo unos con otros. Hay muchos retos en el "hipermundo", retos enfocados en cómo asegurar la simbiosis armoniosa de humanos, computadoras y cosas. El término Wisdom Web of Things (W2T) es utilizado para denotar esta problemática. El término está inspirado por el ciclo material en el mundo físico, W2T se enfoca en el ciclo "*de las cosas a los datos, información, conocimiento, sabiduría, servicios, humanos y entonces de regreso a las cosas*". Un sistema W2T está diseñado para implementar dicho ciclo (Zhong *et al.*, 2010).

Aunque nuestras contribuciones no estuvieron enfocadas en incorporar el concepto de los sistemas W2T, estas pueden ser utilizadas para dar soporte en el desarrollo de sistemas W2T de la forma siguiente: la parte básica del ciclo (datos, información y conocimiento) es implementada gracias al diseño de servicios individuales que son descubiertos e identificados en términos de qué información proveen y cómo lo hacen; la conversión de conocimiento a sabiduría y de ahí a servicios es cubierta al haber identificado a los servicios necesarios para una composición (que es como se aplicaría la sabiduría), y esta al ser ejecutada se transforma a su vez en llamadas a los servicios involucrados; finalmente,

la parte de servicios a los humanos y de ahí de vuelta a las cosas se implementa con el uso del editor de *mashups*, en donde los usuarios interactúan directamente con los servicios generando solicitudes que finalmente influyen en el comportamiento de las cosas.

3.3 Conclusiones

En este capítulo se introdujo la visión de los ambientes inteligentes, así como los requerimientos y las características especiales de sus aplicaciones. Estos conceptos son importantes para el presente trabajo de tesis, ya que los ambientes inteligentes es el escenario de aplicación que se ha utilizado durante la fase de diseño y experimentación. En el siguiente capítulo se describe a detalle el diseño y funcionamiento del marco de trabajo que este trabajo propone para dar soporte a las características especiales de las aplicaciones en los ambientes inteligentes. De igual manera, en este capítulo se analizaron trabajos relacionados que presentan algunas similitudes a nuestras contribuciones.

Capítulo 4

UbiSOA Framework

Como se menciona en el primer capítulo, a través de los años se ha propuesto una gran cantidad de proyectos en plataformas y marcos de trabajo para cómputo ubicuo. Los modelos en los que estos proyectos están basados son muchos también: ontologías, predicados, espacios de datos, interpretación de contexto, entre otros. Como algunos autores mencionan (Yang *et al.*, 2006), dos de los modelos han demostrado ser más apropiados para el diseño de sistemas ubicuos: la orientación a servicios y el modelo guiado por contexto. En este trabajo de tesis se sigue un modelo híbrido. El marco de trabajo propuesto está basado principalmente en la orientación a servicios ya que las entidades básicas están representadas por servicios que proveen interfaces REST y debido al uso del mecanismo de descubrimiento de servicios como un directorio para encontrar y consumir los recursos necesarios. Pero, también se emplea el modelo guiado por contexto con la integración de capacidades de adquisición de contexto en la forma de notificaciones de eventos vía *push* para servir y administrar información de contexto desde y hacia consumidores y proveedores de servicios.

La plataforma propuesta se llama UbiSOA (en referencia a Ubiquitous Service-Oriented Architecture) y como cualquier plataforma que permite la composición y ejecución de aplicaciones ad hoc, esta provee tres mecanismos básicos: el descubrimiento de servicios, para la detección e identificación de componentes en tiempo de ejecución; un mecanismo de mensajería en común para todos los participantes, para permitir el intercambio de información y la cooperación entre servicios; y un mecanismo de notificación de eventos, para permitir que una aplicación pueda responder a cambios en el ambiente.

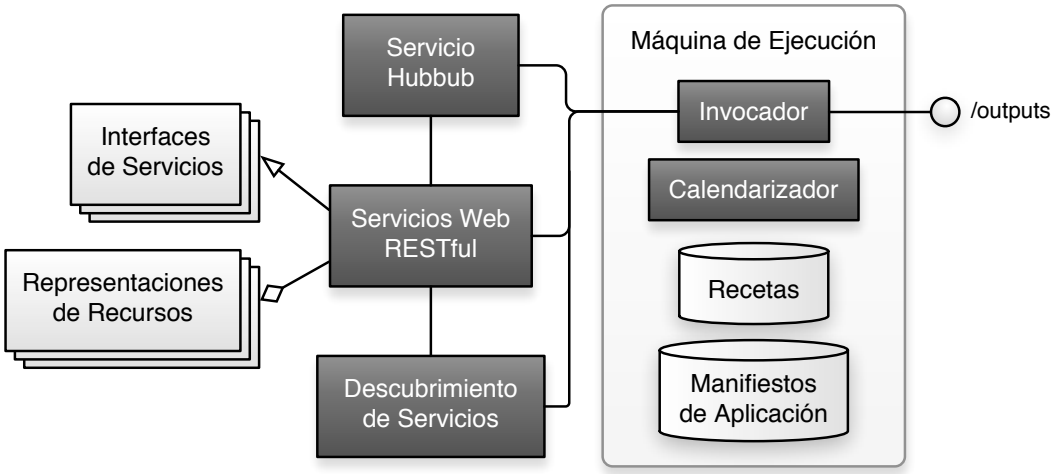


Figura 4.1: Vista general de la arquitectura de UbiSOA.

4.1 Vista general

UbiSOA tiene 4 componentes principales: los servicios, las máquinas de ejecución, el mecanismo de descubrimiento y las notificaciones (véase la Fig. 4.1). Un servicio es la unidad mínima de lógica o datos y está representado por un servicio Web RESTful que implementa una o más interfaces genéricas y que provee recursos utilizando las representaciones recomendadas por UbiSOA. El descubrimiento de servicios permite a los consumidores de servicios encontrar a sus potenciales proveedores en un ambiente inteligente. Las notificaciones las proveen servicios Hubbub que avisan a los consumidores suscritos cuando algún recurso fue actualizado en el sistema. Finalmente, las máquinas de ejecución son las responsables de consumir, extraer y combinar los distintos recursos según como sean instruidas a través del uso de manifiestos de aplicación y de la aplicación de recetas. En las siguientes secciones se detalla cada uno de estos componentes o elementos principales.

4.2 Servicios en UbiSOA

Cada componente en la plataforma está modelado como un servicio, incluso las entidades que se encargan de hacer el seguimiento de las composiciones de servicios (las máquinas de ejecución). Definimos a un servicio en UbiSOA como un servicio Web que expone su funcionalidad siguiendo el modelo REST (Fielding, 2000). La razón del soporte para este tipo de servicios en lugar de soluciones más tradicionales (tales como las del WS-*) es debido a que los servicios REST son más susceptibles a la integración ad hoc (Pautasso, 2009). Los servicios REST no necesitan de objetos *proxy* especiales para la administración de llamadas entre servicios ya que son utilizados a través de llamadas HTTP simples, una característica disponible en la mayoría de los lenguajes de programación modernos.

Cualquier servicio Web puede ser parte de la plataforma siempre y cuando cumpla con las siguientes características (véase la Figura 4.2):

- Sigue el modelo RESTful.
- Implementa al menos una de las interfaces de servicios de la plataforma.
- El servicio por sí mismo o a través de otra entidad en su representación registra su descripción en la plataforma mediante el uso del mecanismo de descubrimiento de servicios.
- Provee representaciones de recursos en al menos uno de los formatos de datos recomendados.

Hay una decisión importante de diseño en la descripción de servicios UbiSOA. Los sistemas RESTful carecen de un lenguaje de descripción estándar debido a sus propias restricciones arquitectónicas. Un API basado en REST no puede definir nombres fijos para los recursos, así que la interfaz completa de un servicio no puede ser determinada a priori. Específicamente, la restricción arquitectónica establece que *el hipermedia debe ser la máquina de estado de aplicación*. Es decir, después de la solicitud inicial, junto con la respuesta

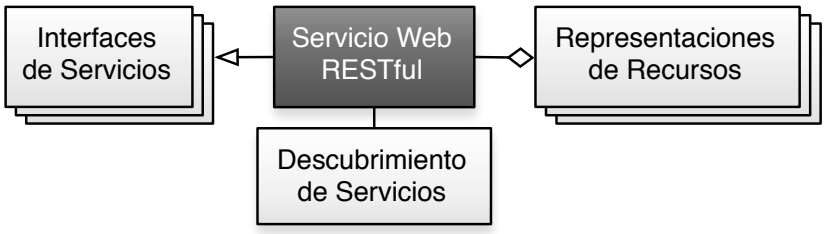


Figura 4.2: La estructura de un servicio UbiSOA.

se obtiene una relación para URIs relacionados que se pueden utilizar para continuar solicitando datos según sea necesario para conformar la funcionalidad deseada. Mientras que la limitación arquitectónica desacopla efectivamente al servidor de los clientes, es difícil establecer una manera formal y estandarizada para describir los APIs basados en REST. Lenguajes tales como WADL (Handley, 2009) se han utilizado para estos propósitos. En UbiSOA, un servicio no está completamente descrito por un solo documento WADL. En su lugar, un servicio puede implementar múltiples especificaciones de interfaces, las cuales están almacenadas en documentos individuales. La intención es simplificar la especificación de composiciones de servicios y proveer flexibilidad al permitir que un servicio implemente tareas compuestas mucho más ricas.

Se han diseñado interfaces de servicios básicas en las siguientes categorías:

- *Adquisición de contexto y acceso a datos.* Servicios que proveen la adquisición de contexto primario. Por ejemplo, redes inalámbricas de sensores, lectores RFID, entre otros. Los servicios de datos permiten el acceso a documentos y fuentes de datos Web. La mayoría de los servicios en esta categoría proveen notificaciones de eventos.

- *Servicios lógicos.* Son servicios utilizados para proveer procesamiento adicional de entrada/salida. Por ejemplo: un servicio para derivar información de contexto compleja desde datos de sensado crudos y un servicio de geologalización que calcula la ubicación de un objeto a partir de una colección de lecturas de fuerza de señal Wi-Fi.

- *Control y automatización.* Servicios que proveen una interfaz de control para entidades físicas tales como luces, motores, puertas, abanicos y otros.

- *Visualización.* Utilizados para mostrar datos al usuario. Los servicios en esta categoría cubren reproductores, lectores o *widgets* Web. La plataforma incluye un tipo especial de *widgets* Web que pueden ser insertados en páginas Web y que están listos para actualizarse a sí mismos con el arribo de notificaciones de nuevo contenido.

- *Servicios básicos.* Describen los servicios de los mecanismos básicos de la plataforma. Por ejemplo, se podría incorporar una nueva tecnología de descubrimiento de servicios tan sólo diseñando un servicio que implemente su interfaz.

Este conjunto puede ser fácilmente extendido al tomar como base alguna de las interfaces disponibles. Por otra parte, con el objetivo de conformar un sistema de mensajería común, todos los recursos proveídos por servicios en la plataforma deben ser entregados en al menos uno de los siguientes formatos:

- *HTML*. Una página Web diseñada para su uso por usuarios humanos. Esta representación debe proveer acceso a toda la funcionalidad ofrecida por el servicio.
- *XML*. Una representación válida en un esquema XML genérico relacionado a la funcionalidad proveída por el servicio. Así como se provee un conjunto básico de interfaces, también se provee un conjunto de esquemas listos para ser utilizados.
- *JSON*. Un archivo JSON contiene los mismos datos que una representación XML equivalente. Además implementan el modelo JSONP.
- *ATOM*. Una fuente Atom cuyos elementos están aumentados con etiquetas XML del esquema XML relacionado.

La decisión acerca de qué tipos de formato deben ser proveídos durante la implementación de un escenario personalizado depende de la máquina de ejecución que será utilizada para ejecutar la composición de servicios (como se discute en la Sección 4.5.2).

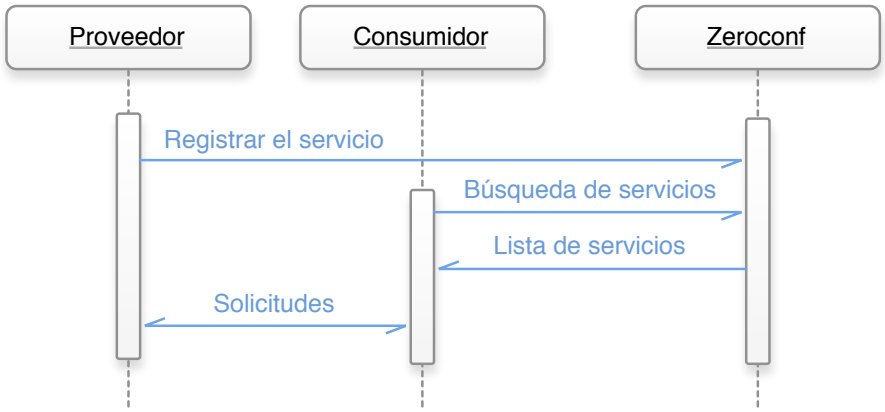


Figura 4.3: Diagrama de secuencia del descubrimiento de servicios.

4.3 Descubrimiento de servicios

El mecanismo de descubrimiento de servicios está principalmente basado en el protocolo Zero Configuration Networking (IETF Zeroconf Working Group, 2012). Este protocolo fue seleccionado de entre otros por su facilidad de uso e implementación así como a su baja complejidad. El protocolo se originó en un grupo de trabajo de la IETF para detectar servicios en una red local de forma automática. Se utilizan las capacidades de Zeroconf para anunciar servicios en UbiSOA. Para hacerlo, una vez que un servicio está listo para recibir solicitudes, debe ser registrado en el ambiente al transmitir su descripción a través de la implementación Zeroconf que se utilice[1] (véase el diagrama de secuencia en la Fig. 4.3). La descripción de un servicio consiste de los siguientes atributos: el nombre del servicio, una descripción humanamente legible, el nombre del servidor y el número de puerto del recurso de red, un URI base en el que los *endpoints* REST están contenidos, el URL del servicio *hub* utilizado para transmitir notificaciones de eventos, y finalmente, las interfaces que el servicio implementa (URLs a las especificaciones WADL).

[1] La implementación de UbiSOA utilizada para conducir los casos de estudio están basados en Bonjour, una implementación de Zeroconf desarrollada por Apple. `http://developer.apple.com/bonjour`

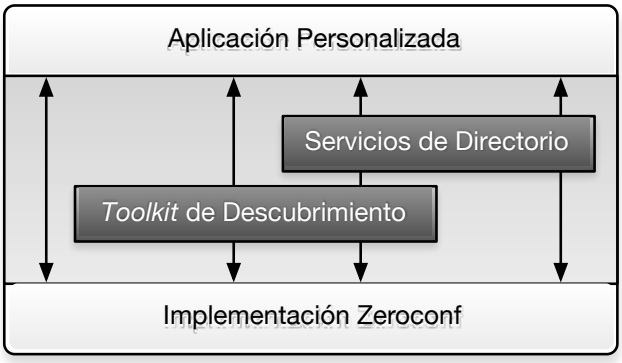


Figura 4.4: Alternativas de interacción con el mecanismo de descubrimiento de servicios.

Existen tres maneras de registrar un servicio (véase la Figura 4.4):

- A través de interacción directa con Zeroconf, utilizando una de las muchas bibliotecas disponibles para diferentes lenguajes de programación. Este método es el más complejo y requiere de conocimientos previos acerca de cómo utilizar el protocolo así como de las especificaciones correctas utilizadas por UbiSOA.
- A través del uso del *Toolkit* de descubrimiento. El *toolkit* es un conjunto de utilerías de programación escritas en Java y en C/C++ para proveer un nivel de abstracción más alto a las librerías de Zeroconf. También incluye otros métodos tales como la habilidad para obtener notificaciones cuando cambie la disponibilidad de servicios.
- A través de un servicio de directorio. Este tipo especial de servicios es utilizado cuando la interacción directa con la implementación Zeroconf no está disponible. También proveen notificaciones de eventos cuando un servicio registrado cambia de estado de disponibilidad.

Con estas alternativas, los servicios en un ambiente pueden ser identificados fácilmente para ser utilizados después. Debido a que sólo se pueden buscar servicios en la red local, se han incluido otro tipo de servicios llamados los servicios *gateway* que pueden ser utilizados como puentes entre dos o más ambientes. De esta manera, los servicios disponibles

remotamente pueden ser utilizados en composiciones locales o viceversa. Una vez que el proveedor de un servicio se ha registrado en Zeroconf, cualquier consumidor interesado lo podrá encontrar cuando este último realice una búsqueda por algún servicio que implemente una interfaz dada.

4.4 Notificación de eventos

En las arquitecturas REST un evento usualmente representa una notificación de la actualización de un recurso enviada a las entidades suscritas a un servicio. Esto es lo que pasa en UbiSOA, el mecanismo de notificación de eventos permite el intercambio de información de contexto y mensajes de control al alertar a los servicios ejecutándose en la plataforma cuando un recurso conteniendo dichos datos es actualizado. El diseño del mecanismo está basado en el protocolo PubSubHubbub (2012), también conocido como Hubbub. Este protocolo introduce una entidad especial llamada *hub* en el modelo tradicional de comunicación publicar/suscribir. En Hubbub, una vez que un consumidor realiza su primera solicitud del recurso de un proveedor, la representación de respuesta incluye un vínculo apuntando al servicio *hub* utilizado por el proveedor para anunciar actualizaciones de recursos; los consumidores interesados en recibir notificaciones para ése recurso en particular puede entonces enviar una solicitud de suscripción al *hub* vinculado (véase el diagrama de secuencia en la Fig. 4.5). El *hub* almacena una copia local de la representación que será actualizada después de que recibe una solicitud de *ping* del proveedor notificándole que existen nuevos datos. Si esto no pasa dentro de una cierta ventana de tiempo, el *hub* solicita el recurso por sí mismo y actualiza su copia local; cuando sea que el recurso es actualizado, el *hub*notifica a cada uno de sus suscriptores que pueden entonces solicitar la representación actualizada al *hub*.

Hasta el momento, Hubbub sólo ofrece soporte a representaciones de recursos en los formatos Atom y RSS ya que permiten que los elementos contengan vínculos a recursos

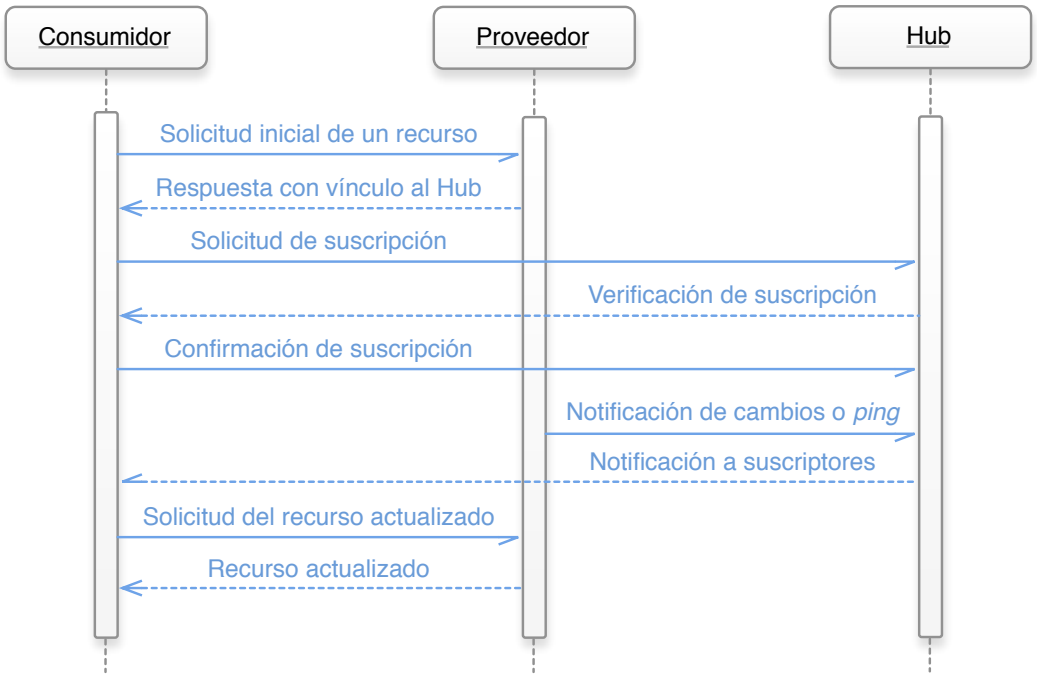


Figura 4.5: Diagrama de secuencia de la notificación de eventos.

relacionados, en este caso, a servicios *hub*. Sin embargo, UbiSOA extiende estas capacidades al agregar el soporte a cualquier representación posible. Las referencias a servicios *hubs* son esperadas como vínculos dentro de los documentos o como un atributo en las descripciones de los servicios. De esta manera, los servicios que no pueden incluir vínculos en las representaciones debido a las limitaciones de los formatos de datos, pueden apuntar a un *hub* en la descripción del servicio. adicionalmente, si un servicio declara implementar el protocolo Hubbub pero no apunta a un *hub* en ninguna manera en particular, una máquina de ejecución buscará un servicio *hub* disponible en el ambiente para ser utilizado para notificaciones de eventos durante el flujo de las composiciones.

4.5 Lenguaje y máquinas de ejecución

En los sistemas *mashups*, una aplicación es en realidad una composición de servicios mezclados para proveer la lógica necesaria para una aplicación. Para poder construir tales

aplicaciones se necesita primero describir la composición de servicios utilizando el lenguaje que esté disponible en la plataforma que correrá los *mashups*. Como se indica anteriormente, la definición de una descripción formal para servicios REST aún necesita ser estandarizada; lenguajes de composición general tales como WS-BPEL (OASIS, 2012) o EMML (OMA, 2012) que son altamente dependientes de las descripciones completas de los servicios tienen dificultades para ofrecer una solución fácil de usar para desarrolladores sin experiencia. Al ser lenguajes de composición de propósito general, son sistemas de granularidad muy fina así que los desarrolladores deben estar conscientes de cómo las solicitudes, descripciones y cómo la lógica individual se mezclará. Para confrontar estos problemas, se propone un lenguaje simple de composición en el cual la lógica de aplicación está expresada por la descripción de las dependencias entre componentes o servicios con baja granularidad. Cada paso de composición es realizado de acuerdo a las interfaces que implementan los servicios involucrados, así que la lógica de aplicación completa puede ser expresada como el grafo de una tubería lógica directa. Las aplicaciones de UbiSOA están descritas por documentos llamados manifiestos de aplicación, que definen:

- Los servicios involucrados en la composición y su descripción.
- El flujo entre servicios.
- Las salidas del *mashup* donde los datos o funcionalidad finales están disponibles.
- Datos de control de aplicación tales como el calendario de ejecución, credenciales de autentificación y otros metadatos.

El escenario de uso de los sistemas tradicionales de *mashups* es muy simple: una vez que la composición está definida, esta es enviada para su ejecución a un servidor; el servidor ejecuta la composición y expone cualquier servicio generado a través del flujo de datos. Los sistemas que proveen *mashups* ubicuos, por otra parte, deben considerar características especiales adicionales.

UbiSOA presenta las siguientes características:

- Un manifiesto de aplicación no sólo es ejecutado en servidores públicamente disponibles, estos pueden ejecutarse en ambientes locales cerrados, dispositivos móviles personales, etc.
- El usuario puede ser parte del flujo de ejecución (para por ejemplo escoger entre alternativas).
- La composición incluye además de servicios específicos, servicios genéricos cuya entidad de quien los implementa se desconoce de antemano.
- Se involucran eventos o datos en tiempo real proveniente de dispositivos físicos.
- La ejecución de una aplicación puede ser calendarizada.
- Los usuarios pueden compartir y colaborativamente diseñar manifiestos de aplicación utilizando las máquinas de ejecución disponibles en el ambiente (siempre y cuando tengan los permisos apropiados).
- Los servicios en representación de objetos en el mundo real son administrados a través de permisos y control de acceso.

Los manifiestos de ejecución son ejecutados por una de las entidades especiales en la plataforma: los servicios de máquinas de ejecución. Una máquina de ejecución está a cargo de administrar el flujo de datos descrito al identificar proveedores, solicitar recursos, recibir eventos relacionados y proveer recursos de salida para finalmente completar la funcionalidad objetivo. Su arquitectura está compuesta de 5 componentes principales (véase la Figura 4.6) los cuales son en realidad implementaciones de una o más de las interfaces de servicios básicas. Para discutir la funcionalidad de cada componente y el escenario de uso general consideremos como ejemplo una aplicación que genera un *widget* con un mapa que muestra las últimas lecturas de sensado de 2 redes inalámbricas de sensores. La Figura 4.7 muestra la representación gráfica de los servicios involucrados y el flujo de datos que terminará en la generación de un recurso de salida conteniendo la representación de un *widget* que recibe notificaciones de eventos y que puede ser insertado en páginas Web.

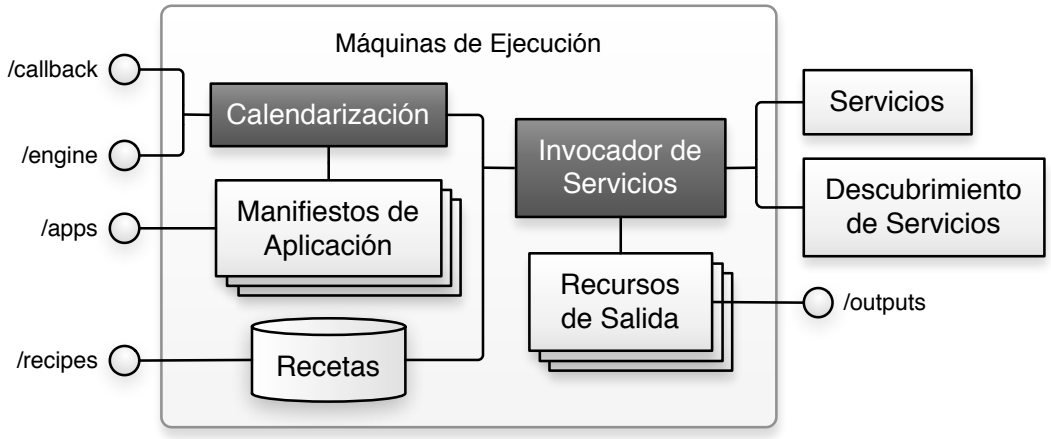


Figura 4.6: Arquitectura de una máquina de ejecución.

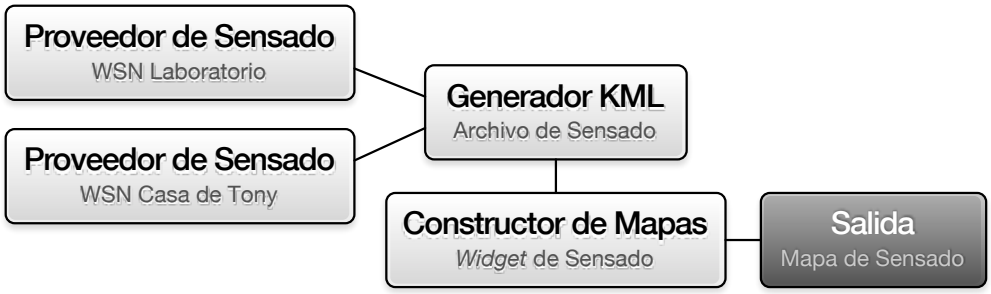


Figura 4.7: La representación gráfica de un manifiesto de aplicación.

4.5.1 Emplazando manifiestos

El escenario inicia con el envío del manifiesto de aplicación de la aplicación de ejemplo (véase la Figura 4.8) a una máquina de ejecución. Los archivos de manifiestos emplazados en una máquina son administrados a través del *endpoint* */apps*, que es en donde este tipo de recursos deben ser controlados de acuerdo a la interfaz *app-storer*.

4.5.2 Ejecutando aplicaciones

Una vez que una aplicación ha sido emplazada, el componente Calendarización iniciará su ejecución en la fecha y hora indicados en el manifiesto. Las aplicaciones pueden ser

```
{
   "interfaces": [
      { "id": "i1", "url": "http://…/push-publisher.wadl" },
      { "id": "i2", "url": "http://…/push-subscriber.wadl" },
      { "id": "i3", "url": "http://…/sensing-provider.wadl" },
      { "id": "i4", "url": "http://…/kml-generator.wadl" },
      { "id": "i5", "url": "http://…/widget-generator.wadl" },
      { "id": "i6", "url": "http://…/map-builder.wadl" } ],
   "services": [
      { "id": "s1", "name": "WSN Research Lab",
         "implements": [ "i3", "i1" ], … },
      { "id": "s2", "name": "WSN Tony's Home",
         "implements": [ "i3", "i1" ], … },
      { "id": "s3", "name": "Sensing Map File",
         "implements": [ "i4", "i2", "i1" ], … },
      { "id": "s4", "name": "Sensing Widget",
         "implements": [ "i6", "i5", "i2", "i1" ], … } ],
   "outputs": [
      { "id": "o1", "name": "Sensing Map", … } ],
   "flow": [
      { "from": [ "s1", "s2" ], "to": [ "s3" ], … },
      { "from": [ "s3" ], "to": [ "s4" ], … },
      { "from": [ "s5" ], "to": [ "o1" ], … } ],
   …
}
```

Figura 4.8: Un manifiesto de aplicación en formato JSON.

iniciadas inmediatamente, a un cierto tiempo o sólo puestas en espera. Adicionalmente, para flujos simples, donde no están involucradas notificaciones de eventos, el manifiesto puede declarar tiempos de inicio adicionales y un esquema de repetición. Ya sea que la aplicación esté corriendo o no, estas pueden ser controladas mediante la interacción directa con el componente Calendarización utilizando el *endpoint* /apps, una funcionalidad obligada por la interfaz *app-engine*.

4.5.3 Recetas lógicas

Para realmente ejecutar aplicaciones, el flujo de datos como está descrito en los manifiestos es primero transformado en un árbol de dependencias lógicas. Entonces, el compo-

nente Invocador de Servicios examina cada nodo y coteja sus condiciones actuales contra los requerimientos de la receta lógica que es utilizada para llevar a cabo el procesamiento actual requerido por ese paso en el proceso del *mashup*. Las condiciones de un paso se refieren a: la descripción del servicio representado por el nodo actual, la presencia o disponibilidad de dicho servicio y los servicios utilizados por este nodo. Los requerimientos de una receta declaran: las interfaces que el servicio objetivo debe implementar y qué tan reciente un recurso debe ser para que pueda ser utilizado en el procesamiento de la lógica proveída por la receta. El concepto de recetas está basado en el hecho de que muchas de las tareas de composición presentan situaciones muy similares. En la aplicación de ejemplo, se puede esperar que la funcionalidad expresada por la conexión de cada instancia del Generador KML a una interfaz Constructor de Mapas sea para generar un *widget* con un mapa resaltando algunos puntos del archivo KML. Si este no es el caso, fácilmente se pueden agregar más recetas a una máquina de ejecución publicando la representación de una receta (que incluye los requerimientos y el código *script* para realizar el paso de composición) al *endpoint* `/recipes` (parte de la interfaz *app-cookbook*).

4.5.4 Administrando solicitudes y eventos

Para completar el paso de la composición de acuerdo a la receta lógica, el Invocador de Servicio realiza todas las solicitudes de recursos que estén involucradas. Al arribar las respuestas, estas son mezcladas (por el *script* de la receta) en un nuevo recurso que será almacenado en el nodo actual del árbol lógico completo. Para manejar los eventos, el Invocador de Servicios se suscribe a sí mismo a los recursos para los cuales requiere notificaciones utilizando para ello un servicio *hub* intermediario. Es aquí donde el descubrimiento de servicios es utilizado para encontrar un *hub* en el caso en que uno de los servicios declare implementar la interfaz *push-subscriber* pero no incluya un vínculo a un servicio *hub*. Después de la suscripción, las notificaciones de eventos son entregadas al *endpoint* `/callback` de la máquina de ejecución, donde el componente de Calendarización identifica el mani-

fiesto de aplicación objetivo y activa la ejecución del nodo en donde la suscripción fue hecha. Cuando el flujo de datos de una aplicación es completado, el Invocador de Servicios es activado en cada nodo hijo para posibilitar la actualización de cualquier recurso derivado. El Invocador de Servicios no provee directamente ninguna funcionalidad externa así que no implementa ninguna interfaz de servicios.

4.5.5 Interacción con usuarios

Es llevada a cabo por servicios cuyos recursos representan interacciones de usuario. Este tipo de servicios implementa la interfaz *push-publisher* principalmente para detener el flujo de datos de una aplicación y permitir que el usuario cause una notificación de eventos con su interacción.

4.5.6 *Endpoints* de salida

Los recursos generados en cada paso de la composición son almacenados en el árbol de dependencia de servicios. Algunas veces, la completa funcionalidad de una aplicación es llevada a cabo a través de la acumulación de cada solicitud de recursos individual, en otras ocasiones, el objetivo final de un *mashup* es proveer nuevos recursos. Para ofrecer soporte a esto, el manifiesto de aplicación incluye declaraciones de salidas en el flujo de datos (véase la Figura 4.8), todos los servicios conectados terminan en una salida en la cual se genera código fuente que los usuarios pueden copiar y pegar en las páginas Web. Para interactuar con las salidas, cada máquina de ejecución expone las salidas de las aplicaciones en el *endpoint* `/outputs` (como establece la interfaz *app-started*). Los recursos de salida no son documentos planos y simples, una aplicación puede seguir siendo ejecutada y esta actualizando sus salidas. En el ejemplo, las lecturas de sensado serán constantemente notificadas. los *widgets* de UbiSOA están diseñados para recibir actualizaciones PubSubHubbub del lado del cliente, así que las salidas son documentos vivos, siempre actualizados. Adicionalmente, un recurso de salida puede declarar la implementación de interfaces,

como si se tratara de un servicio normal, así que el mecanismo de descubrimiento puede identificarlos e incluirlos en composiciones.

4.5.7 Compartiendo aplicaciones

Las máquinas de ejecución pueden ser utilizadas simultáneamente por múltiples usuarios. Para diseñar colaborativamente aplicaciones, los usuarios pueden editar los mismos manifiestos interactuando con el *endpoint* */apps*. Existe otra forma de compartir manifiestos: la exportación de manifiestos de aplicación. Los manifiestos exportados contienen junto con el documento original, la definición de todas las recetas encontradas apropiadas por la máquina de ejecución utilizada. Así, si un desarrollador quiere llevar una aplicación a otra máquina de ejecución esta será ejecutada de la misma forma (siempre y cuando ambas máquinas de ejecución utilicen la misma tecnología de *scripts*).

4.6 Conclusiones

A lo largo de este capítulo se detallan las características, componentes y mecanismos del marco de trabajo propuesto por el presente trabajo de tesis. El siguiente capítulo describe los resultados de la fase de experimentación en la que se trabajó en el soporte a diferentes escenarios de aplicación con los objetivos de mejorar iterativamente el diseño del marco de trabajo y de ejemplificar su uso.

Capítulo 5

Prototipos y escenarios de aplicación

Durante la fase de diseño de UbiSOA se experimentó con el desarrollo de una serie de prototipos buscando dirigir ideas, enfocar esfuerzos y concretar decisiones a través de las muchas iteraciones de diseño. Los prototipos ayudaron también a identificar soluciones en común a los distintos problemas presentados por los escenarios de aplicación. Tales soluciones, en la forma de servicios y herramientas de *hardware* y *software*, constituyen la infraestructura base de UbiSOA. En este capítulo se describe dicha infraestructura así como su utilización en la implementación de 3 escenarios de aplicación en el contexto de los ambientes inteligentes.

5.1 Servicios

Como se menciona en el capítulo 4, un servicio en UbiSOA consiste básicamente de un servicio Web RESTful que expone su funcionalidad al describir las interfaces que implementa y que notifica su disponibilidad dentro de un ambiente al registrarse asimismo en el mecanismo de descubrimiento de la plataforma. Se ha diseñado un conjunto de interfaces genéricas para estos servicios incluyendo: capacidades para adquisición de contexto, análisis de comportamiento, geolocalización bajo techo, administración de metadatos, entre otros. Una vez que los servicios son registrados, un desarrollador puede entonces crear composiciones definiendo qué servicios están involucrados en una composición y describiendo el flujo de datos entre ellos en un manifiesto de aplicación. Los manifiestos son enviados a una de las máquinas de ejecución disponibles en el ambiente, que son finalmente las encargadas de realizar solicitudes, administrar los eventos que se generen y mantener el flujo de datos con el objetivo de proveer la funcionalidad de la aplicación. Si así lo decide,

el desarrollador puede optar por realizar las solicitudes y recibir los eventos manualmente, es decir, sin el uso de las máquinas de ejecución. Para ello, deberá utilizar los mecanismos de la plataforma directamente.

De igual manera, es importante recordar que algunos servicios en la plataforma tienen la necesidad de notificar datos según se generen, es decir, siguen un modelo de comunicación basado en eventos. UbiSOA ofrece soporte para este tipo de servicios al implementar un modelo de comunicación *publish/subscribe*. Un ejemplo sería el siguiente: supongamos que un servicio necesita recibir notificaciones cada vez que haya una nueva lectura en un lector de RFID. El servicio debe suscribirse al servicio que provee las lecturas de RFID indicando parámetros tales como el cómo y cuándo se hará la notificación. Ya que la mayoría de los componentes de la plataforma son en realidad servicios RESTful, una notificación de datos consiste usualmente de una solicitud POST a un URI objetivo. Adicionalmente, la notificación puede ser configurada para utilizar operaciones de agregación para por ejemplo, en lugar de recibir múltiples cambios de temperatura cada par de segundos, recibir la temperatura promedio por hora.

Se ha desarrollado un prototipo funcional de UbiSOA que incluye versiones de servidor, escritorio y móvil de la máquina de ejecución. Para conducir los casos de estudio utilizados en la fase de experimentación, se han implementado parcial o completamente los siguientes servicios básicos:

5.1.1 Geolocalización bajo techo

El objetivo de este servicio es estimar la ubicación actual de un usuario basándose en una muestra de la intensidad de señal de un dispositivo con respecto a puntos de acceso Wi-Fi cercanos. Este esquema ha sido muy utilizado en otros trabajos que requieren el conocimiento de la ubicación de usuarios u objetos en ambientes bajo techo (Krumm y Horvitz, 2004). El servicio requiere ser entrenado antes de que una ubicación pueda ser esti-

mada, este se realiza midiendo la intensidad de la señal Wi-Fi en diferentes puntos del área a ser cubierta y asociando la intensidad de la señal medida a una ubicación específica. El proceso está asociado directamente a la plataforma que se utiliza durante el entrenamiento, requiriendo entrenamiento adicional para cada nueva plataforma que se desee utilizar. Cuando se solicita una estimación de ubicación, el servicio busca en las medidas almacenadas durante el entrenamiento y estima una nueva posición utilizando una heurística personalizada basada en la similitud de las intensidades de señal. Durante pruebas experimentales y utilizando entre 3 y 10 puntos de acceso Wi-Fi, se pudo alcanzar un error promedio cercano a los 4 metros que resultó ser suficiente para las necesidades de geolocalización de los casos de estudio.

5.1.2 Redes inalámbricas de sensores

Este servicio provee una abstracción de alto nivel a diferentes plataformas de sensores inalámbricos permitiendo la adquisición de contexto primario de manera sencilla y práctica. El tipo de contexto que se adquiere depende de la capacidad de sensado de los nodos en la red. Durante el desarrollo del prototipo de UbiSOA, se utilizaron 8 motes MicaZ y 10 motes Telos B, ambos de Crossbow (2012), que proveen contexto ambiental (luminosidad, temperatura, humedad, ruido) y de usuario (captura de audio y de movimiento mediante el uso de un acelerómetro). En algunos casos especiales, se proveyó a cada usuario con un nodo de sensado para monitorear su comportamiento al rastrear los cambios en los datos del acelerómetro. La implementación actual de este servicio está basado en un trabajo previo (Avilés-López y García-Macías, 2009). El servicio provee notificaciones asíncronas siguiendo el modelo *publish/subscribe*. Una entidad debe primero suscribirse describiendo el tipo de sensor deseado, el área de sensado de interés, la ventana de tiempo y el operador de agregación a ser aplicado a las lecturas de sensado disponibles dentro de esa ventana.

5.1.3 Lectores RFID

El servicio provee notificaciones a partir de lecturas de etiquetas RFID. La intención de este servicio es ofrecer una forma de identificar automáticamente objetos y usuarios. Este servicio se utiliza en el escenario de toma de medicinas para identificar las botellas de pastillas (véase la Subsección 5.3.1).

5.1.4 Fuentes de video

La implementación actual de este servicio toma la forma de un *widget* Web, un componente visual que puede ser fácilmente insertado en otras páginas Web o sistemas. Adicionalmente, su diseño permite modificar las fuentes de video para que en conjunto con técnicas de procesamiento de imágenes pueda ofrecer un análisis de comportamiento más rico y complejo.

5.1.5 Comandos de voz

Los comandos de voz son un modo de interacción muy natural en donde el esfuerzo del usuario es minimizado. Con este servicio, se facilitan comandos de voz en un ambiente al proveerlo con una red inalámbrica de sensores cuyos nodos están equipados por un micrófono abierto permanentemente. La confiabilidad de la red de reconocimiento de voz es extremadamente importante, más sobre todo ya que los nodos de sensado son altamente susceptibles a fallas. Por ello, durante las pruebas se instalaron los nodos redundantemente, a menudo con áreas de cobertura superpuestas, para que de esta manera si un nodo falla otro pudiera retomar su funcionalidad sin irrumpir en el funcionamiento general de la red. La funcionalidad principal que el servicio expone es la notificación de un nuevo comando identificado por la red. Para detectarlo, el nodo primero captura sonido, si está dentro del rango de voz humana es entonces enviado a una estación base en donde el sonido es analizado para determinar si es un comando. La estación base es además, el

proveedor del servicio. Para detalles más específicos véase el trabajo publicado por Palafox y García-Macias (2009).

5.1.6 Síntesis de voz

La sintetización de voz es otra forma muy natural de comunicación con los usuarios ya que no los distrae de sus otras actividades. Proveedores de este servicio tienen directamente asociados una bocina inalámbrica en la que se reproducen mensajes de texto sintetizados que son recibidos a través de solicitudes REST.

5.1.7 Actuadores

Servicios de este tipo permiten a los desarrolladores llevar a cabo una reacción física en el ambiente que puede tomar la forma de luces, motores, sonidos entre otras acciones. Se han implementado servicios para encender servomotores sencillos que pueden ser utilizados para abrir puertas o ventanas. Siguiendo el diseño de la infraestructura, los servicios de actuación incluidos en la implementación funcional de UbiSOA son genéricos, es decir, pueden ser fácilmente adaptados para controlar cualquier dispositivo de *hardware* que dependa de parámetros para ejecutar algún movimiento (como rotación o traslación).

5.1.8 Dispositivos de entrada

Se han implementado servicios para permitir entradas de usuario simples tales como botones, barras de deslizamiento, eventos *touch*, rotadores y otros utilizando Phidgets (2012). Como se menciona en el capítulo 3, una de las características comunes de los sistemas Aml es el proveer interacción natural con los usuarios a través de los medios a los que están acostumbrados. Los servicios de entrada se han diseñado con un enfoque general para que puedan ser extendidos para incluir otro tipo de dispositivos, por ejemplo, sensores infrarrojos, balanzas eléctricas, entre otros. Este tipo de servicios están basados

en eventos así que cualquier entidad interesada en recibir el evento de una nueva entrada de usuario debe primero suscribirse a ella.

5.1.9 Otros servicios

Existen otros servicios implementados para permitir acciones especializadas. Un ejemplo de dichos servicios es el procesador de contexto que recibe lecturas de sensado enviadas por un servicio de sensores inalámbricos y entonces, después de aplicar una cierta lógica (a través de *scripts*)), genera nuevos eventos. Dicho servicio se ha utilizado en el escenario de aplicación de monitoreo inteligente.

UbiSOA en general está diseñado para ser fácilmente extensible. Cualquier desarrollador que desee implementar nuevos tipos de servicios debe seguir los siguientes pasos:

- Implementar el servicio en cualquier lenguaje de programación siempre y cuando este exponga su funcionalidad a través de un servicio RESTful.
- Describirlas en un archivo de descripción las interfaces genéricas que el servicio implementa.
- Registrar el servicio en el mecanismo de descubrimiento de servicios utilizando para ello una de las herramientas proveídas por la plataforma.

5.2 Herramientas

Junto con el diseño e implementación de servicios en la plataforma, se ha desarrollado un par de herramientas construidas a partir de la infraestructura de UbiSOA para mejorar el soporte para el desarrollo de aplicaciones de vida asistida. La más importante es un *toolkit* llamado *sentient visor* que puede ser utilizado para desarrollar aplicaciones que permitan la interacción de los usuarios con objetos aumentados en una forma muy natural.

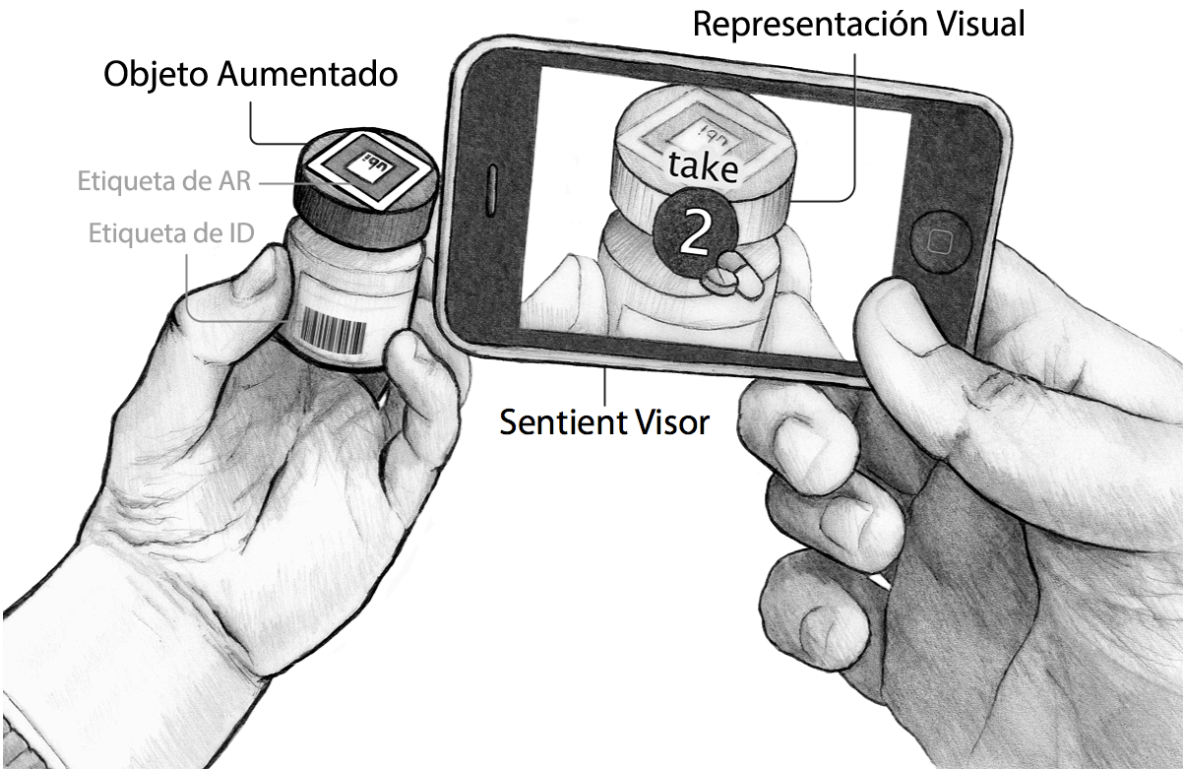


Figura 5.1: El escenario de uso de un *sentient visor*.

5.2.1 *Sentient visors*

La interacción de usuario con ambientes inteligentes tiene el objetivo de ser lo más natural posible. Idealmente, sin incluso requerir el uso de medios tradicionales tales como pantallas, teclados, etc. Cuando se diseña la interacción de usuarios en dichos ambientes, sobre todo si están enfocados a adultos mayores, se debe pensar en nuevas formas de interacción que ofrezcan a los usuarios alternativas que no requieran de conocimientos previos ni de entrenamiento especial para utilizarse. Por ejemplo, los adultos mayores necesitan un modelo de interacción humano-computadora simple y claro que les presente información filtrada, condensada y resumida basada en sus perfiles y preferencias personales. Siguiendo la idea introducida por el concepto de dispositivos ambientales (Wisneski

et al., 1998) en este trabajo de tesis se desarrolló lo que llamamos *sentient visors*. Un *sentient visor* es un dispositivo llevado por el usuario que básicamente presenta tres características: una pantalla, una cámara (apuntando hacia el frente) y capacidades de sensado. Una computadora *tablet* o un *smartphone* moderno pueden ser utilizados como *sentient visor* (Wright, 2009).

El escenario de uso de un *sentient visor* (véase la Figura 5.1 es como sigue):

- Para iniciar la interacción, el usuario apunta con la cámara (el dispositivo) al objeto aumentado de interés o al área en el ambiente de acuerdo a lo que el escenario de aplicación involucra.
- El dispositivo entonces automáticamente identifica a lo que está siendo enfocado, captura cualquier información de contexto adicional que sea requerida por la aplicación y entonces envía una solicitud pidiendo un escenario de realidad aumentada a un servicio en la plataforma que será mostrado en la pantalla del dispositivo.
- El escenario de realidad aumentada podría incluir algún tipo de interacción básica tal como tocar uno de los objetos en la pantalla. En dicho, caso, el dispositivo enviaría solicitudes adicionales a los proveedores de información según el flujo de aplicación lo necesite.

El escenario final es que el usuario observa una versión aumentada de la realidad física. De esta manera, el usuario puede interactuar con los objetos que él utiliza en su vida diaria con tan solo apuntar e dispositivo hacia el mismo. Se utiliza esta herramienta en el escenario de toma de medicamentos (véase la Subsección 5.3.1).

Se puede encontrar más información acerca del *sentient-visor* en las siguientes publicaciones: Avilés-López *et al.* (2009, 2010); García-Macías *et al.* (2011).

5.2.2 Editor de *mashups*

En UbiSOA, un *mashup* ubicuo se puede codificar en la forma de un *script* (el manifiesto de aplicación) que especifica los servicios involucrados, las interconexiones entre ellos y el flujo de datos a través de la aplicación. O bien, una aplicación puede ser ensamblada utilizando el Editor de *Mashups* de UbiSOA, un sistema para crear *mashups* a través de actividades tan simples como arrastrar y soltar representaciones gráficas de los servicios involucrados para dar soporte a un escenario determinado.

En el escenario de uso del editor se espera que el usuario lleve consigo un dispositivo que provea sus servicios personales (tales como localización, contactos, notificación, entre otros), ejecute aplicaciones *mashup* y permita el acceso a la infraestructura a través del editor. Internamente, el editor es en realidad una combinación de un servicio de directorio y uno de ejecución. Ambos servicios conforman lo que es la funcionalidad local del dispositivo. Es decir, proveen un directorio de servicios locales y una máquina que ejecutará *mashups* de forma local.

El editor de *mashups* UbiSOA está conformado por 8 paneles de interacción (véase la Figura 5.2). En el lado izquierdo del editor, se encuentran los recursos directamente relacionados con el usuario (paneles 1, 2 y 3). En el panel 1 se listan todos los *mashups* que están almacenados en la máquina de ejecución local. En este panel se puede detener o iniciar el estado de los *mashups* así como también conocer cuál es el problema en el caso en que el flujo de datos de uno de ellos se haya detenido. En el panel 2 se listan todos los servicios almacenados en el directorio local. Normalmente, los servicios en este panel corresponden a proveedores que están ejecutándose en el mismo dispositivo del usuario en donde se está ejecutando el editor. Al panel 2 se le nombra como el panel de los servicios personales. En el panel 3 es un listado de servicios adicionales que el usuario ha registrado en el directorio. En este panel los usuarios pueden agregar más servicios personales que no estén contemplados en el panel 2. Por ejemplo, sus cuentas de Facebook o Twitter.

Figura 5.2: Vista general del editor de *mahups* UbiSOA.

Del lado derecho del editor, se encuentran los recursos que se han encontrado automáticamente en el ambiente del dispositivo del usuario (paneles 4, 5, 6 y 7). Los paneles 4 y 5 podrían aparecer de forma múltiple. El editor busca todos los servicios de máquinas de ejecución y de directorio que haya disponibles. Por cada máquina de ejecución, el editor muestra una instancia del panel 4, por cada directorio, una instancia del panel 5. El panel 4 es un listado de las aplicaciones registradas en la máquina de ejecución y el panel 5 un listado de los servicios en el directorio. El editor permite utilizar servicios de ejecución o de directorio que no se han detectado. El usuario puede agregar el URL de estos servicios estos se mostrarán como una instancia del panel 6. En el panel 7 se listan todos los servicios para los que existe una receta en todas las máquinas de ejecución encontradas. Los servicios listados no tienen un URL de alguien que los provea, el objetivo es que el usuario los pueda utilizar en los flujos de datos para especificar la presencia de un servicio genérico como se explica más adelante.

En la parte central del editor se encuentra el área de instanciación (panel 8). En esta área el usuario arrastra las representaciones de servicios a la izquierda (recursos locales) o derecha (recursos remotos) del editor y los interconecta al dibujar las relaciones entre las cajas. Una vez que el usuario ha completado la implementación del *mashup*, puede proceder a enviarlo a una de las máquinas de ejecución ya sea local o remota.

El capítulo 6 contiene más detalles acerca del editor de *mashups*. Para más información acerca de su diseño, consulte la publicación: Avilés-López y García-Macías (2009).

5.3 Escenarios de aplicación

A continuación se presentan algunos escenarios apoyados con desarrollos de este trabajo de tesis que utilizan tecnologías de vida asistida con el objetivo de promover estilos de vida más seguros y sanos para los adultos mayores. Estos escenarios han sido muy útiles para demostrar los beneficios de UbiSOA para desarrollar aplicaciones AmI.

Figura 5.3: La aplicación de toma de medicamentos mostrando los dos posibles resultados.

5.3.1 Toma de medicamentos

El Sr. Hernández es una persona de edad avanzada que necesita tomar muchos medicamentos diferentes diariamente. Algunas veces, olvida tomar alguna medicina y lo que es aún peor, toma dos o tres veces la misma dosis porque simplemente quiere asegurarse que la tomó. Esta situación que es muy conocida mundialmente (Routledge *et al.*, 2004) puede llevar a problemas adversos con consecuencias muy serias. El Sr. Hernández está físicamente discapacitado y le es imposible visitar a su médico excepto en ciertas situaciones muy especiales. Sin visitas frecuentes, las revisiones médicas son conducidas a través de llamadas por teléfono y algunas veces el médico cambia la dosis al alterar el número de pastillas que debe tomar o al imponer una calendarización diferente lo que dificulta aún más el problema porque el Sr. Hernández debe recordar la dosis apropiada para cada una de las diferentes botellas de pastillas que toma durante el día.

Para ofrecer soporte a este escenario, se deben identificar primero qué objetos en la vida diaria del Sr. Hernández se pueden aumentar para permitirle utilizar el sistema de una manera que le parezca natural. Se decidió por las botellas de pastillas a las que se les anexó una etiqueta RFID para poder identificarlas individualmente. A continuación, se analizó qué servicios disponibles en la plataforma se podrían utilizar. Se seleccionaron los servicios de lector RFID (para la identificación de botellas) y el servicio de sintetización de voz (para alertar al usuario). Después, se identificó qué servicios personalizados eran necesarios para completar la funcionalidad necesaria. Se implementaron dos servicios más: uno para proveer una página Web que permitiera modificar la especificación de las dosis al médico y un servicio que llevara un registro de la toma de las dosis y que le enviara alertas al usuario cuando la calendarización de una dosis se cumpliera. Finalmente, para tomar todos los usuarios e implementar la aplicación final, se utilizó el *sentient visor* para crear un escenario de realidad aumentada para mostrar si la dosis de una botella debe ser ingerida o si el usuario debe esperar.

El nuevo escenario con el soporte de UbiSOA es como sigue: El Sr. Hernández toma muchos medicamentos diferentes en el transcurso del día. Cuando él quiere asegurarse de que está cumpliendo el calendario de las dosis toma su dispositivo móvil (un *smartphone*) y lo apunta a cada una de las botellas de pastillas. En la pantalla, la botella aparenta brillar de verde o rojo indicando si debe tomar la dosis o esperar por ella; si debe tomarla, el número de pastillas aparece en pantalla, si debe esperar, un contador animado aparece mostrando cuántos minutos y segundos debe esperar hasta la dosis siguiente (véase la Figura 5.3). Cuando toma la dosis, un botón aparece que debe ser presionado una vez que la tome. Cuando el Sr. Hernández no ha usado el teléfono y el calendario no se cumple, el servicio de alerta notifica al servicio de sintetización de voz para enviarle una notificación de usuario diciéndole el número de pastillas que debe tomar inmediatamente. Algunas veces, el médico del Sr. Hernández ingresa a la página Web para cambiar el calendario de las dosis o para consultar el progreso del paciente, pero todo esto es transparente para el

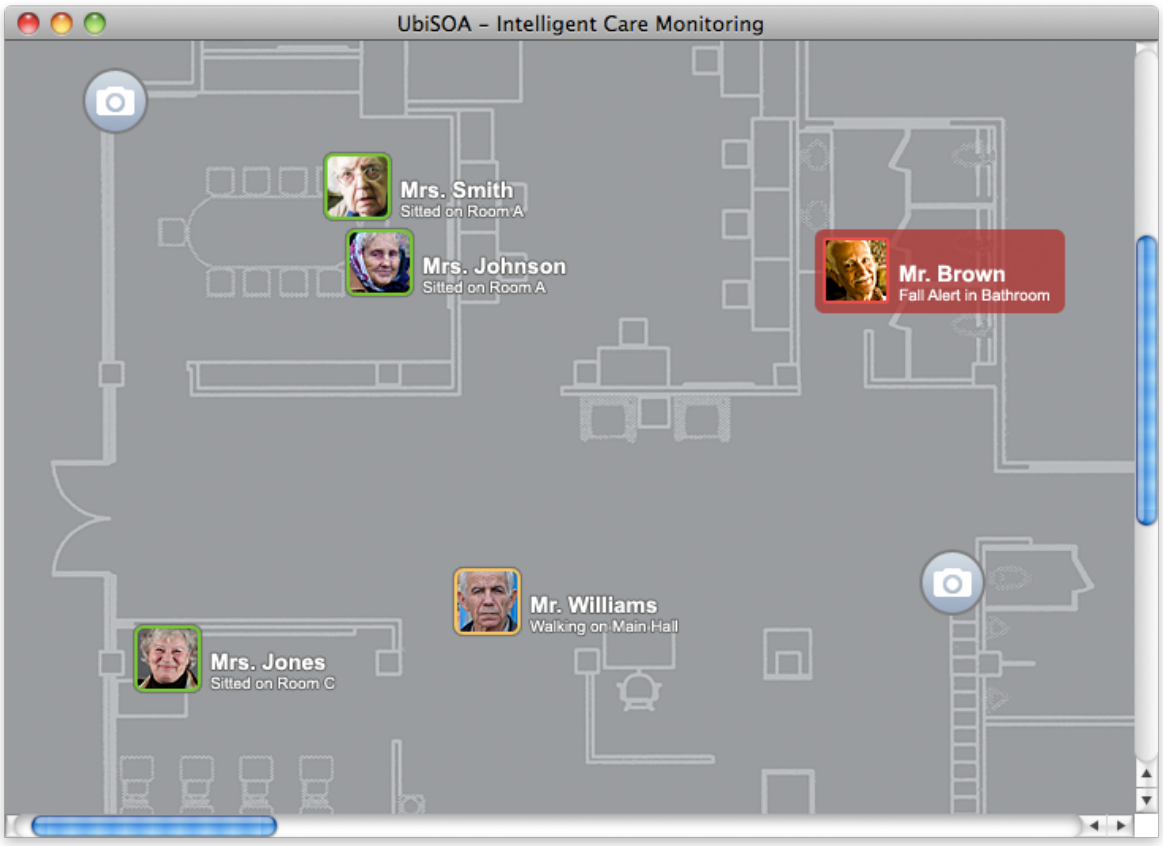


Figura 5.4: Versión de escritorio de la aplicación de monitoreo inteligente.

Sr. Hernández ya que es el sistema el que se encarga de las actualizaciones y de los otros detalles sin requerir su participación explícita.

5.3.2 Monitoreo inteligente

La Sra. García es una cuidadora en un hogar de asistencia. Como parte de sus actividades, debe revisar regularmente a los residentes para ver si necesitan asistencia. Para evitar que ella camine a través de los diferentes cuartos en el edificio, le instalaron un sistema de video de circuito cerrado. Ahora, cuando está en la oficina puede echar un vistazo a los residentes al observar un monitor que muestra todas las fuentes de video en el edificio. Algunas veces, cuando uno de los residentes cae, si ella no es cuidadosa con la inspección,

el adulto mayor podría pasar un largo tiempo en el suelo hasta que se de cuenta del accidente. Debido al hecho de que durante el proceso de envejecimiento trae consigo una declinación considerable de las habilidades motoras, los adultos mayores se vuelven cada vez más propensos a las caídas. Mientas que la Sra. García no es la única cuidadora, esta tiene muchas otras responsabilidades, y después de un tiempo, el monitor con las fuentes de video ya no se usa más para monitoreo sino para tener un registro histórico de lo que pasa dentro del edificio.

Para implementar este escenario, se proveyó a cada uno de los residentes con nodos de sensado portátiles, de esta manera se pudo conocer su ubicación utilizando las intensidades de señal Wi-Fi e inferir su comportamiento al analizar las lecturas de un acelerómetro. Se obtiene la ubicación de los usuarios gracias al servicio de geolocalización el cual tuvo que ser entrenado antes de usarse. Se tomaron lecturas en 2,500 ubicaciones diferentes cada una con 10 lecturas de intensidad de señal. Se equipó al hogar de asistencia con 5 puntos de acceso Wi-Fi extra para mejorar la estimación de ubicación. Cada uno de los nodos interactúa directamente con el servicio de geolocalización para estimar su ubicación individual y entonces la reporta a un servicio centralizado. El análisis de comportamiento es proveído por lecturas de acelerómetro, cada sensor reporta segmentos de datos cada 10 segundos a un servicio centralizado, los datos son analizados ahí para concluir si el usuario está caminando, sentado o si ha caído. La implementación del servicio centralizado en ambos casos es una especialización del procesador de contexto. Se utilizan las fuentes de video de cada cámara para proveer una forma para que los cuidadores confirmen que las alertas son eventos reales, evitando falsos positivos. Finalmente, se implementó una aplicación en versiones de escritorio y móviles que utiliza la funcionalidad proveída por los servicios de ubicación y comportamiento centralizados.

El nuevo escenario es como sigue: La Sra. García debe revisar a los residentes regularmente. Cuando está en la oficina, en su computadora de escritorio observa un mapa del

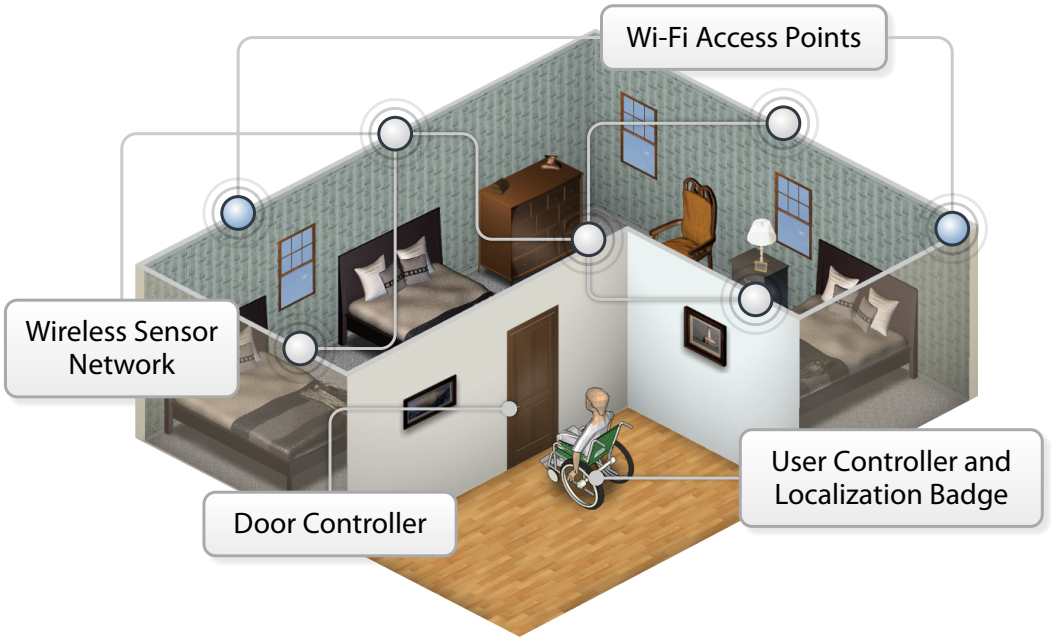


Figura 5.5: Infraestructura implementada para el escenario de asistencia basada en voz.

edificio mostrando la actividad y ubicación de cada residente. Cuando hace *click* en uno de los íconos de los residentes, el sistema muestra una fuente de video de la cámara más cercana para que pueda asegurarse de que no es un falso positivo y entonces ir directamente a asistir al residente (véase la Figura 5.4). Cuando ella y otros cuidadores no están en la oficina, estos reciben una alerta de caída conteniendo la ubicación y una imagen capturada desde la cámara más cercana. Los residentes también pueden utilizar un botón en el nodo de sensado para activar una alerta.

5.3.3 Asistencia basada en voz

El Sr. Martínez es un paciente que utiliza una silla de ruedas eléctrica para moverse dentro del edificio. Muy a menudo, requiere asistencia para abrir puertas o ir al baño. Cuando nadie está cerca, él tiene que esperar hasta que un cuidador u otra persona lo ayuda.

Para dar soporte a este escenario, se utilizan los servicios de geolocalización, comandos de voz, dispositivos de entrada y actuadores presentados anteriormente. Se diseñó un dispositivo de entrada personalizado (y su servicio) que presenta dos botones para uso del adulto mayor. La aplicación (una composición de servicios) es ejecutada en el ambiente. Cuando el usuario presiona el botón de comando, la aplicación identifica su ubicación y espera para que el servicio de comandos de voz detecten un comando, cuando lo recibe, actúa de acuerdo a la solicitud. Cuando el usuario presiona el botón ayuda, el sistema envía una notificación a un cuidador en la forma de un mensaje conteniendo la ubicación del paciente (véase la Figura 5.5).

El escenario asistido por voz es como sigue: El Sr. Martínez es un adulto mayor que presenta discapacidades físicas que lo hacen usar una silla de ruedas. Cuando necesita abrir una puerta, presiona el botón comando y dice abrir, entonces la puerta enfrente de él se abre y lo deja entrar. Una vez dentro, presiona el botón nuevamente y dice cerrar, entonces la puerta se cierra tras él. Cuando el Sr. Martínez requiere asistencia de un cuidador, presiona el botón ayuda, entonces un cuidador recibe la solicitud y su ubicación para que pueda ser encontrado rápidamente.

5.4 Lecciones aprendidas

Trabajar con sistemas para ambientes de vida asistida centrados en los adultos mayores es una tarea interesante y con muchos retos; mientras que la investigación ofrece un buen grado de complejidad, los resultados tienen un impacto social importante. Cuando se diseñaron los escenarios de aplicación se encontraron problemas que requieren una administración de privacidad más cuidadosa, por ejemplo, el escenario de monitoreo inteligente, en donde las actividades de un hogar de asistencia son constantemente rastreadas. Se cree que, siempre y cuando la aplicación sea mantenida localmente, la privacidad de los usuarios en dichos casos no es tan crucial. Por otra parte, existen escenarios que están más

orientados hacia el cuidado de la salud, en donde el procesamiento y transmisión de datos confidenciales de los pacientes requieren de mayores consideraciones de privacidad y seguridad (Ruan y Yeo, 2009). Se encontró otro problema con la precisión de la geolocalización. Se necesitaba un mecanismo para obtener geolocalización al menor costo posible (en términos de cómputo, infraestructura y precio) lo que llevó a basarlo en lecturas de intensidad de señal Wi-Fi. Las técnicas basadas en dichas lecturas logran obtener estimaciones dentro de un error aproximado de 20 metros, pero se necesitaba una ventada de error de a lo más 5 metros. Se logró que la precisión fuera lo suficientemente satisfactoria pero esto resultó ser particularmente difícil en el escenario de asistencia basado en voz. Afortunadamente, las puertas en el hogar de asistencia en el que se condujo la experimentación estaban lo suficientemente apartadas una de otra. En el capítulo 6.4 se incluye una discusión más detallada acerca del soporte para ambientes inteligentes utilizando el enfoque de orientación a servicios y creación de aplicaciones a través de composición.

Capítulo 6

Evaluación

Para evaluar nuestras contribuciones se siguió una estrategia consistente en 3 fases. La primera fase comprendió de un experimento de usabilidad con los usuarios potenciales de la plataforma para evaluar qué tan fácil de entender el marco de trabajo es y si existían problemas de usabilidad que impidieran lograr los objetivos planteados a los participantes. La segunda fase trata de la verificación del cumplimiento de los requerimientos de diseño que fueron identificados desde la etapa de análisis del trabajo de tesis. Finalmente, la tercera fase de evaluación consistió de, utilizando los resultados de las 2 fases anteriores, tomar cada uno de los objetivos específicos de la tesis y evaluar si estos fueron alcanzados o no. En este capítulo se detalla cada una de dichas fases de evaluación concluyendo con una discusión acerca del cumplimiento del objetivo general del trabajo de tesis.

6.1 Experimento de usabilidad

La primera fase de la evaluación de UbiSOA fue la conducción de un experimento de usabilidad con el objetivo de evaluar qué tan fácil de entender y utilizar es el marco de trabajo. La facilidad de comprensión significa que los usuarios pueden identificar claramente los distintos elementos de una plataforma, entender cómo interactúan unos con otros, qué acciones puede llevar a cabo para describir la funcionalidad que ellos necesitan y finalmente, qué forma toma dicha funcionalidad. La facilidad de uso significa que los usuarios no tienen obstáculos o impedimentos en lograr realizar las tareas que necesitan llevar a cabo para lograr sus objetivos.

Después de analizar distintas alternativas, se decidió utilizar el marco de trabajo de las dimensiones cognitivas (CDs) (Green, 1989, 1991) como la base para el estudio de usabili-

dad. Una de las razones principales es que es una metodología muy utilizada para conocer la usabilidad de entornos de desarrollo y otras herramientas principalmente visuales. Las CDs son muy útiles también cuando se dificulta la comparación con otras propuestas y sistemas similares ya sea porque no estén disponibles o por la falta de parámetros claros de comparación. El marco de trabajo de las CDs se origina en el análisis de una serie de conceptos que surgen en estudios psicológicos del cómo piensa un desarrollador mientras este realiza las tareas necesarias para crear un artefacto de información. En las siguientes subsecciones se describe la metodología basada en las CDs, la realización del experimento y sus resultados.

6.1.1 Objetivo general del estudio

La realización del estudio tuvo los objetivos siguientes:

- Evaluar la efectividad total de UbiSOA en el desarrollo de aplicaciones en ambientes inteligentes por parte de un grupo de usuarios potenciales.
- Revisar el estado de cada una de las dimensiones del marco de trabajo de las CDs en el diseño de la plataforma, lo que finalmente nos permitió evaluar su facilidad de aprendizaje y uso.
- Identificar obstáculos de usabilidad que impidiera a los usuarios lograr sus objetivos.

6.1.2 Participantes

Dada la naturaleza del estudio, las características de UbiSOA, los recursos disponibles y las tareas que los usuarios llevarían a cabo, se optó por contar con la participación de 6 desarrolladores. Uno de ellos realizaría una prueba piloto para corregir problemas antes de la participación de los otros 5 usuarios. En el caso de no encontrar problemas graves, se utilizarían los resultados de los 6 participantes, como fue finalmente el caso. La decisión de contar con 5 participantes está basada en el trabajo por Nielsen y Landauer (1993). La idea principal es que entre más usuarios participen en una prueba más observaciones en

común se obtendrán, siendo 5 un número apropiado considerando factores tales como el costo/beneficio. Los participantes debían tener experiencia con lenguajes de programación, de preferencia, con aquellos lenguajes diseñados específicamente para el desarrollo de aplicaciones en la Web. Adicionalmente, debían tener experiencia con el protocolo HTTP, o al menos, tener una idea de cómo funciona. En un principio se buscó contar con participantes con poca y mucha experiencia para poder hacer una comparación de los resultados entre los dos grupos. Finalmente se decidió preferir participantes noveles ya que estos ofrecerían mejores observaciones dado que uno de los objetivos de UbiSOA es que pudiera ser utilizado por usuarios no expertos.

6.1.3 Metodología

La metodología del estudio estuvo enteramente basada en el marco de trabajo de las dimensiones cognitivas. Las CDs son una herramienta para ayudar a evaluar la usabilidad de artefactos basados en información ya sean interactivos o estáticos. Es también un compendio de estrategias de diseño bien conocidas en diferentes dominios que llevan a contrarrestar problemas de usabilidad. Históricamente, las CDs fueron introducidas por Green (1989, 1991) quien enfatizó la generalidad del método ya que es aplicable en todos los dominios en donde la representación de información está presente. Más adelante, Green y Patre (1996) hicieron un análisis más exhaustivo en cuanto a su aplicación al dominio de los lenguajes de programación visuales.

El marco de trabajo de las CDs se basa en la lexicalización de conceptos, es decir, en la nomenclatura de conceptos referentes a problemas de usabilidad muy bien conocidos y comunes. Una vez que se les ha dado nombre a los problemas, los que discuten no tienen que explicar cada idea cada vez que la utilicen. Las CDs no proveen una herramienta matemática sobre la que pueda hacerse un análisis profundo, en su lugar, un artefacto de información es evaluado a través de 13 dimensiones diferentes, cada una de las cuales es cognitivamente relevante, la evaluación genera un perfil y el perfil dictamina la conve-

niencia del artefacto para varias tareas. Las dimensiones no son intrínsecamente buenas o malas, diferentes tipos de actividades son mejor identificadas por diferentes perfiles.

Diseñar un jardín no es igual a diseñar un puente. El diseño de un jardín es muy estético y el trabajo a realizar es explorar las opciones, cambiar de idea, y encontrar una nueva visión. El diseño de puentes es menos inspirador y mucho más crítico ya que hay que considerar la seguridad. Ambos son dos tipos diferentes de actividad de usuario y ambos representan diferentes tipos de *software* y herramientas a utilizar.

El marco de trabajo considera 4 tipos de actividades diferentes:

- *Incrementación.* Agregar una nueva ficha a un fichero. Agregar una fórmula a una hoja de cálculo.
- *Transcripción.* Copiar detalles de un libro a una ficha bibliográfica. Convertir una fórmula en términos de una expresión lógica.
- *Modificación.* Cambiar los términos de un catálogo. Cambiar la organización de un documento. Modificar un programa para resolver otro problema.
- *Diseño exploratorio.* Diseño tipográfico. Esquematización. Programación al vuelo.

Cada actividad está mejor representada por su propio perfil de dimensiones cognitivas. Antes de definir las dimensiones, es necesario introducir los conceptos de notación, ambiente y medio. La **notación** comprende las marcas o símbolos percibidos por el desarrollador que son combinados para construir una estructura de información. El **ambiente** contiene las operaciones o herramientas para manipular dichas marcas. La notación es impuesta en un **medio**, que puede ser persistente como el papel, o temporal como el sonido. Las dimensiones cognitivas valoran el proceso de construir o modificar estructuras de información. Ya que el proceso es afectado por los tres conceptos, la notación, el ambiente y el medio, es necesario tomarlos en cuenta durante la evaluación.

Existen otros dos conceptos: capas y subdispositivos. Ambos conceptos son utilizados para descomponer un sistema complejo en partes independientes cada una con su propia notación, ambiente y medio.

Las 13 dimensiones del marco de trabajo son las siguientes:

- *Abstracción*, tipos y disponibilidad de mecanismos de abstracción.
- *Dependencias ocultas*, vínculos importantes entre entidades que no son visibles.
- *Compromiso prematuro*, limitaciones en el orden de hacer las cosas.
- *Notación secundaria*, información adicional en medios diferentes a la sintaxis formal.
- *Viscosidad*, resistencia al cambio.
- *Visibilidad*, habilidad de ver los componentes fácilmente.
- *Cercanía del mapeo*, cercanía de la representación al dominio.
- *Consistencia*, semántica similar es expresada en formas sintácticas similares.
- *Difusidad*, expresividad del lenguaje.
- *Propensión al error*, la notación facilita errores.
- *Operaciones mentalmente difíciles*, alta demanda de recursos cognitivos.
- *Evaluación progresiva*, el progreso puede ser verificado a cualquier hora.
- *Provisionalidad*, grado de compromiso a las acciones o marcas de la notación.
- *Expresividad de rol*, el propósito de un componente es rápidamente inferido.

Tradicionalmente, la aplicación de las CDs consiste en argumentar la aplicación de cada una de las dimensiones en un sistema evaluado. Algunos autores han optado por realizar mediciones por ejemplo, el número de líneas de código, para argumentar si la viscosidad de su sistema es buena o mala. Sin embargo, la selección de dichos argumentos es complicada, sobre todo a la hora de defender la validez de la evaluación. Incluso es difícil argumentar qué dimensiones son aplicables para el tipo de actividad del sistema. Por lo mismo, Blackwell y Green (2000) introdujeron un cuestionario genérico para evaluar artefactos de información que permite conocer, a partir de las observaciones de un grupo de usuarios, el

estado de cada dimensión. La metodología de la presente evaluación consistió en la aplicación de una versión simplificada y aumentada de dicho cuestionario. Simplificada, debido a que se omitieron las partes del cuestionario que corresponden al concepto de subdispositivos ya que el presente trabajo de tesis no cuenta con dichas características. Aumentada, ya que se agregaron preguntas a la parte introductoria con el objetivo de corroborar los perfiles de los participantes en cuanto a la experiencia de programación.

6.1.4 Esquema y tiempos de la sesión

Las sesiones se condujeron en una cafetería con una duración de no más de 2 horas cada una. La aplicación de la prueba se realizó individualmente. A continuación se describen las actividades realizadas por cada uno de los participantes.

Introducción a la sesión (10 minutos) – Se discutió con el participante:

- Su experiencia y participación en estudios de usabilidad anteriores.
- La importancia de su participación en este estudio.
- El rol que desempeñó el moderador durante la prueba.
- La organización de las herramientas que se emplearon.
- El protocolo para el resto de la sesión.
- Se le solicitó que piense en voz alta y que narre cada una de sus acciones.

Introducción a la plataforma (20 minutos) – Se dio la misma introducción breve a cada uno de los participantes acerca del cómo está organizada la plataforma, en qué consiste cada uno de los servicios, cómo están conformadas las aplicaciones y cómo son ejecutadas. También se les hizo una demostración del uso del editor de *mashups* en la creación de una aplicación que encendía un LED siempre y cuando una tarjeta RFID estuviera cerca de un lector. Posteriormente se le entregó la documentación impresa (véase el Apéndice A.1).

Tarea de implementación (30 minutos como máximo) – En este punto el participante debió implementar la aplicación de prueba utilizando el editor de *mashups* de la plataforma. La implementación consiste en dos partes, en la primera desarrolló una aplicación base y en la segunda se le pidió realizar una modificación a la misma. Al completarse los 30 minutos se le pidió que se detuviera independientemente del estado en el que se encontrara la implementación.

Después de la prueba (60 minutos) – Se le aplicó el cuestionario de las dimensiones cognitivas (véase el Apéndice A.2) y se le cuestionó acerca de su experiencia del uso de la plataforma, si encontró dificultades o problemas al desarrollar la aplicación y si deseaba realizar algún comentario adicional.

6.1.5 Aplicación a desarrollar

Una aplicación en UbiSOA es en realidad una composición de servicios. Para desarrollar una aplicación primero se debe especificar qué servicios forman parte de una composición y cómo están interconectados. La definición de la composición se describe en un archivo especial denominado Manifiesto de Aplicación. Este archivo es enviado posteriormente a una Máquina de Ejecución que se encarga de realizar las solicitudes a las entidades involucradas y de llevar el flujo de la aplicación. Para facilitar estas tareas, la plataforma ofrece algunas herramientas entre las que se encuentra el editor de *mashups*. Un desarrollador puede alternativamente optar por utilizar los servicios de manera directa, es decir, utilizar los mecanismos que provee UbiSOA para identificar y consumir la funcionalidad de los servicios disponibles en la plataforma. Sin embargo, ya que el editor es en realidad una herramienta espejo de la funcionalidad de bajo nivel, es decir, toda actividad realizable a un bajo nivel se puede replicar en el editor, en este estudio los participantes utilizaron el editor exclusivamente.

El uso del cuestionario de las CDs requiere que los participantes experimenten la extensión de la funcionalidad de una aplicación que previamente desarrollaron, esto para que puedan contestar preguntas acerca de la dificultad de introducir modificaciones en el sistema. Por lo mismo, la tarea de implementación de este estudio se condujo en dos partes. En la primera, la aplicación debía presentar la siguiente funcionalidad:

1. Al iniciar, la aplicación necesita encontrar al menos 1 instancia disponible de 2 servicios diferentes: sensado a través de una red inalámbrica de sensores y un servicio de control de LEDs (con colores rojo, amarillo, verde y azul).

2. La aplicación se deberá suscribir al servicio de sensado para obtener la lectura de luminosidad más reciente cada 2 segundos. El rango de este valor va de 0 a 1000.

3. Inmediatamente después de recibir una notificación con el valor de luminosidad más nuevo y usando el servicio de LEDs se deberán encender las luces de la manera siguiente: si el valor más nuevo cae dentro de 0 y 250, sólo el LED azul se enciende; si el valor cae dentro de 250 y 500, sólo los LEDs azul y verde se encienden; si el valor cae dentro de 500 y 750, sólo los LEDs azul, verde y amarillo se encienden; si el valor cae dentro de 750 y 1000, todos los LEDs se encienden.

En la segunda etapa la aplicación debía ser modificada para agregar lo siguiente:

1. Encontrar al menos 1 instancia de 2 servicios adicionales: un lector de tarjetas RFID y un servicio de publicación a Twitter (configurado para enviar *tweets* a una cuenta predeterminada).

2. Publicar un mensaje en Twitter siempre y cuando se cumplan dos condiciones. La primera es que el LED color de rojo esté encendido según lo establecido en la primera etapa y la segunda es que exista una tarjeta al alcance del lector RFID.

La aplicación a desarrollar permite a los participantes experimentar las características más importantes de UbiSOA. Desde el descubrimiento de servicios, la subscripción y recepción de eventos, hasta la creación de nuevas recetas cuya funcionalidad conforma finalmente la aplicación solicitada.

6.1.6 Ubicación y organización

Se utilizó un entorno semicontrolado para conducir las sesiones. El estudio tuvo lugar en una cafetería equipada con una instalación de *hardware* de la plataforma que consiste en: una red inalámbrica de sensores, un lector RFID con algunas tarjetas y un controlador con 4 LEDs de color azul, verde, amarillo y rojo. Los participantes utilizaron una misma *laptop* con acceso a Internet y en la que estaban instalados los servicios y herramientas de *software* de UbiSOA.

6.1.7 Mediciones

La aplicación de las CDs no requieren de la toma de mediciones otras que las observaciones que los mismos usuarios proveen por medio del cuestionario. Sin embargo, se tomaron algunas medidas tales como el tiempo que tomó a los participantes completar las 2 tareas de implementación, el número de líneas de código que tuvieron que escribir y el número y naturaleza de los errores cometidos. Estas medidas sirvieron para apoyar el análisis de los resultados.

6.1.8 Ejemplo de aplicación

A continuación se ejemplifican las actividades que llevó a cabo cada participante. Primero, durante la introducción a la plataforma, se le hizo una demostración del uso del editor de *mashups*. Aquí es donde se le explicó al participante cómo interactuar con el editor, cuáles son las áreas que lo componen, dónde se listan los servicios que están disponibles

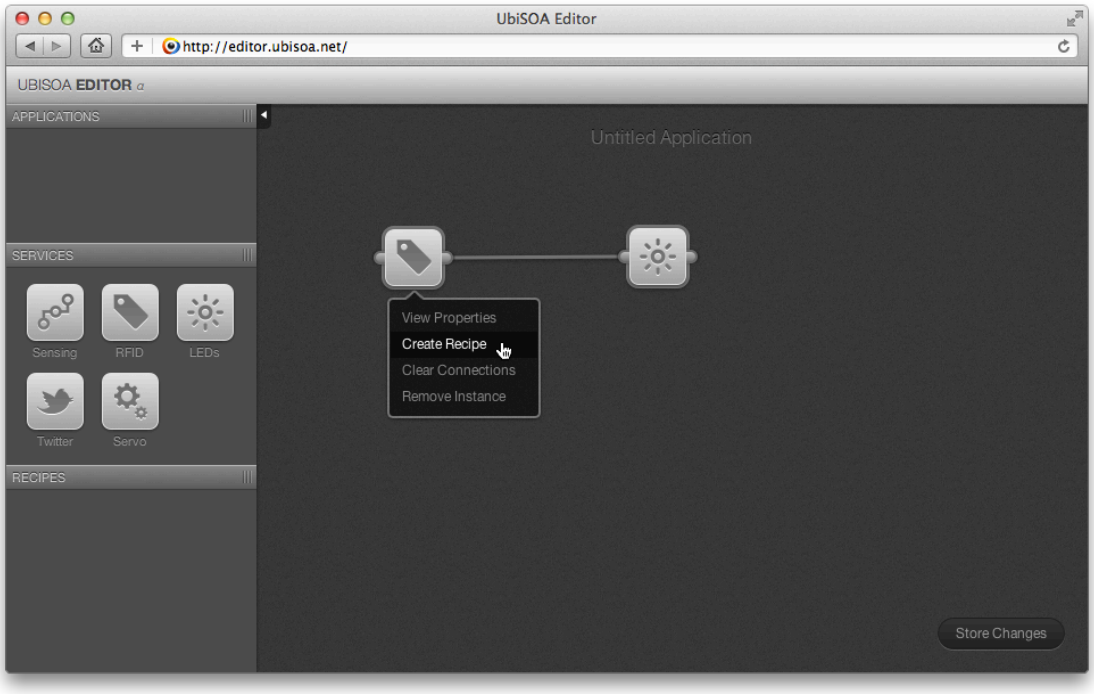


Figura 6.1: El editor, después de agregar dos instancias y una interconexión.

(descubiertos automáticamente en el ambiente), y cómo funciona la interfaz al arrastrar y soltar las representaciones gráficas.

La demostración continuó mostrando cómo crear una aplicación sencilla; Una aplicación que enciende un LED mientras que haya una tarjeta encima de un lector RFID. Para ello, se arrastró y soltó una instancia del servicio con el lector RFID y otra del control de LEDs. Luego, se arrastró el conector del servicio de lectura de RFID con el del control de LEDs. La figura 6.1 muestra cómo se despliegan las instancias y la conexión entre ellas.

En este punto es donde se le explicó al participante el concepto de receta. La forma en la que UbiSOA ejecuta aplicaciones es recorriendo el árbol del flujo de datos (descrito en un manifiesto de ejecución) y revisando en cada nodo si existe una receta que dadas las condiciones hasta ese punto pueda ser ejecutada para ir poco a poco propagando funcionalidad y datos a través del árbol entero.

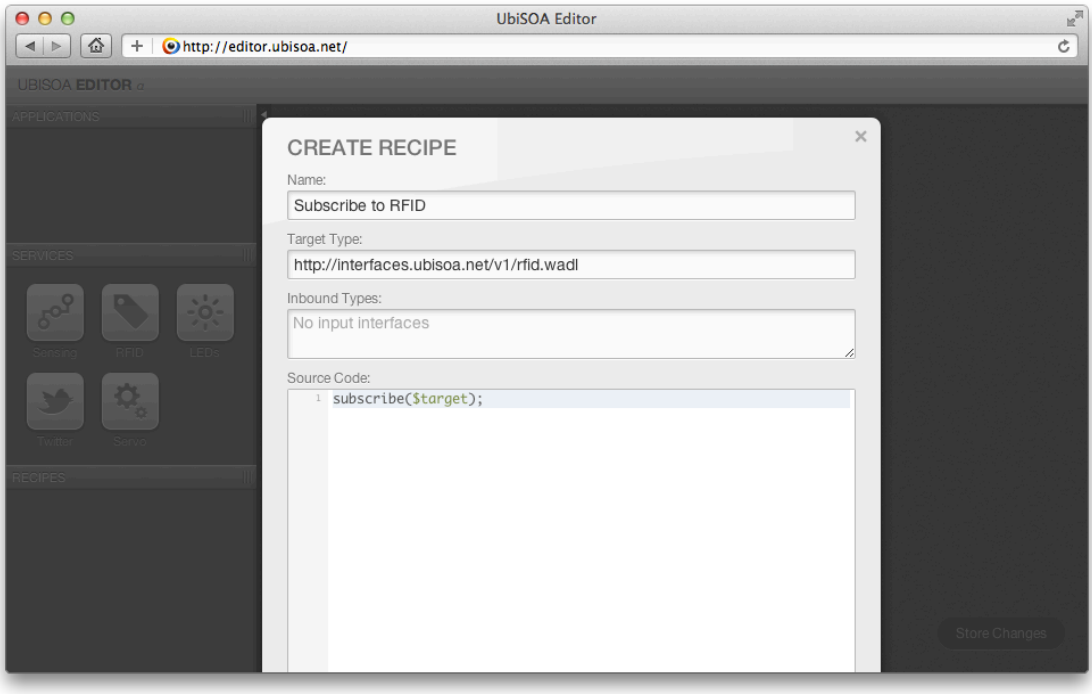


Figura 6.2: La creación de una receta de subscripción a RFID.

Como se puede ver en la figura 6.1 el editor no encuentra ninguna receta en el ambiente (véase la sección abajo a la izquierda de la pantalla). Cuando existe una receta válida que aplica uno de los elementos colocados en el área de instanciación (ya sea instancia o conexión) el borde del mismo se ilumina de color verde. Una vez explicada la situación, se procedió a crear una receta para que UbiSOA sepa qué hacer cuando se encuentre a un servicio de RFID que no tiene conexiones de entrada. En dado caso, la funcionalidad que necesitamos es que la máquina de ejecución se suscriba al servicio.

Aquí se le explica al usuario la naturaleza genérica que se busca en la introducción de nuevas recetas. En el prototipo de UbiSOA, las recetas están implementadas en PHP el cuál se ha extendido con la inclusión de métodos y tipos de datos para simplificar las tareas relacionadas con la extracción de datos e interacción HTTP con los servicios.

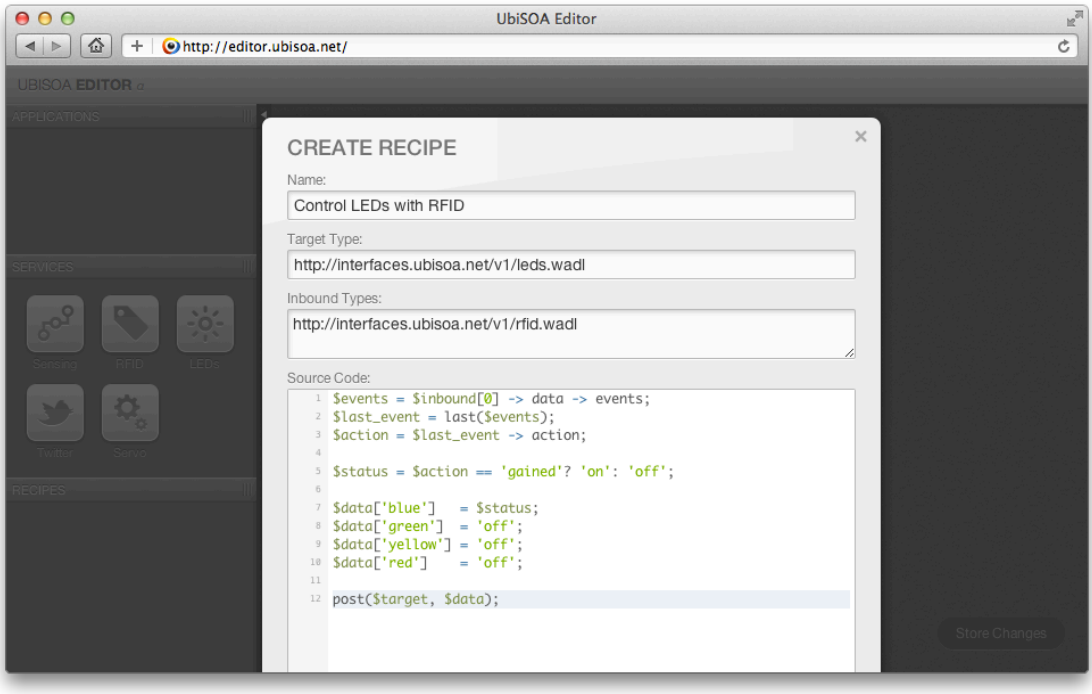


Figura 6.3: La creación de una receta para controlar LEDs usando eventos RFIDs.

Por ejemplo, `$target` representa a la instancia en la que está detenida la máquina de ejecución al recorrer el árbol. En este caso tal instancia es el lector de RFID ya que como se ve en la Figura 6.2, el objetivo debe ser una instancia que implemente la interfaz genérica RFID. Para enviar una subscripción todo lo que el desarrollador debe hacer es llamar el método `suscribe($target)`, UbiSOA se encarga de los detalles de bajo nivel. En el Apéndice A.1 se habla más acerca de estas facilidades.

Una nueva receta más es necesaria para especificar la funcionalidad que necesitamos cuando la ejecución llegue a preguntarse qué hacer con una instancia de un servicio controlador de LEDs que recibe como entrada notificaciones de lecturas RFID asíncronas. En la figura 6.3 se muestra el código necesario para crear esta otra receta. En resumen lo que se hace es obtener la última lectura RFID de la instancia de entrada, si la última acción fue *gained*, es decir, que una tarjeta está al alcance del lector, el LED azul debe encenderse, debe

apagarse en caso contrario. Los parámetros para encender o apagar las luces se envían con método `post($target, $data)`.

Finalmente, la aplicación está lista para ser enviada a una de las máquinas de ejecución disponibles en el ambiente. Una vez registrada, esta aparece en la lista de aplicaciones arriba a la izquierda de la pantalla. El botón junto al nombre de la misma permite iniciar o detener su ejecución.

6.1.9 Resultados

A continuación se presentan y discuten los resultados del experimento de evaluación. Los resultados están organizados de acuerdo a las secciones del cuestionario del experimento. Cada sección corresponde a una dimensión del marco de trabajo de las CDs con tres secciones más. Dos al inicio (Información Básica y Partes del Sistema) y una al final (Preguntas Adicionales).

Información básica

Como información básica el cuestionario pregunta a los participantes sus nombres y el del sistema que evaluaron, el tiempo que lo han utilizado, si se consideran competentes en su uso, y si han utilizado sistemas similares. Adicionalmente, se agregaron preguntas extra para extraer información acerca de la experiencia del participante en cuanto a la programación Web, al uso del lenguaje PHP, su familiaridad con las arquitecturas REST y su experiencia con el protocolo HTTP.

El tiempo de uso es la métrica principal que se tomó para apoyar el análisis de las respuestas al cuestionario. La Figura 6.4 muestra los tiempos para cada participante. El área en azul corresponde al tiempo que les tomó desarrollar la primera parte de la aplicación,

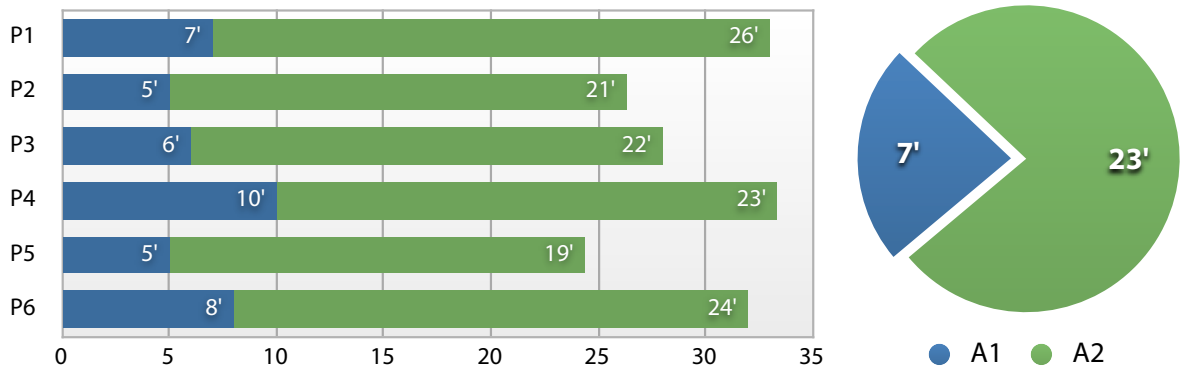


Figura 6.4: Tiempo que tomó a cada participante (P#) completar las aplicaciones (A#).

mientras que el área en verde corresponde a la segunda. En promedio la primera prueba tomó siete minutos y la segunda veintitrés, dando en promedio total los treinta minutos asignados para la tarea de implementación. Tres de los seis participantes tardaron un poco más en completar las dos aplicaciones. Sin embargo, ya que el retraso fue en mayor parte por problemáticas mínimas del funcionamiento del prototipo, se les permitió exceder los treinta minutos máximos asignados.

Posteriormente se les preguntó a los participantes acerca de si se sentían competentes en utilizar el sistema, tomando en cuenta la introducción que se les dio al mismo y su experiencia al utilizarlo después de las tareas de implementación. Tres de cinco participantes contestaron afirmativamente, dos se manifestaron negativamente y uno de ellos mencionó que se sentía muy poco competente en su uso pero que de seguir utilizándolo no tendría problemas en aprenderlo por completo. Precisamente este es uno de los participantes que no tenía experiencia en el uso de PHP.

En cuanto a si habían utilizado sistemas similares, cuatro de los seis participantes mencionan no haberlo hecho. Sólo uno de ellos manifestó haber utilizado un sistema similar aunque sólo en la forma de interacción con el mismo, es decir, a través de una interfaz en la que se arrastran, sueltan y conectan representaciones gráficas de componentes independientes (el sistema en cuestión es LabVIEW (2012). Otro de los participantes citó a Visual

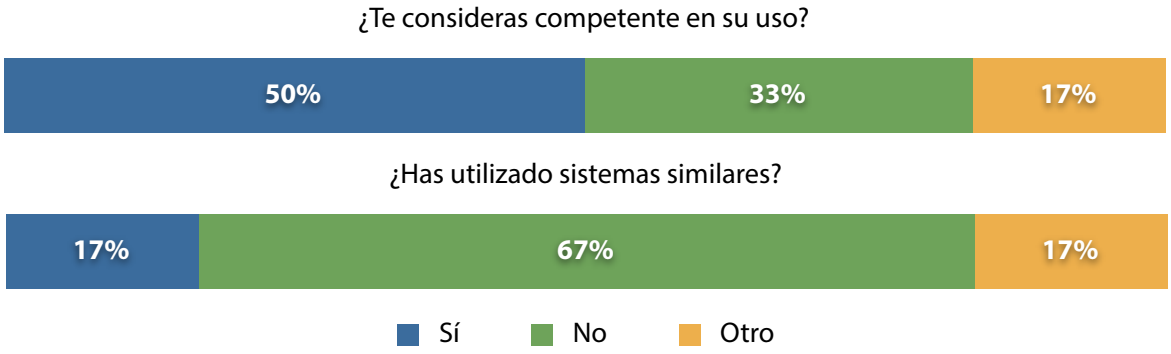


Figura 6.5: Acerca de la competencia en el uso del sistema y el uso de sistemas similares.

Studio como sistema similar pero su respuesta fue categorizada dentro de "otro" ya que, si bien es cierto que Visual Studio tiene un editor gráfico en el que se interconectan objetos, estos son para modelar datos en la forma de XML o diagramas UML y no para definir un flujo de datos o de lógica como en el caso de nuestras contribuciones. La Figura 6.5 muestra las respuestas a esta y a la pregunta anterior.

Para las preguntas restantes relacionadas a su experiencia o familiaridad se les solicitó escoger entre una escala del cero al nueve el valor más apropiado para cada pregunta. La primera de ellas es acerca de qué tanta experiencia tenían en cuanto a la programación de sistemas en la Web. El 67% de los participantes declaró tener una experiencia no mayor de cinco, el 33% restante declaró siete. En cuanto a su experiencia con el lenguaje PHP dos participantes escogieron cero, uno de ellos 3 y el resto seleccionó entre 6 y 8. Acerca de su familiaridad con las arquitecturas REST dos participantes seleccionaron cero, los demás dos, cuatro, seis y ocho. Finalmente, acerca de su familiaridad con el protocolo HTTP tres participantes seleccionaron 8, el resto uno, dos y cuatro. La Figura 6.6 muestra la distribución de las respuestas.

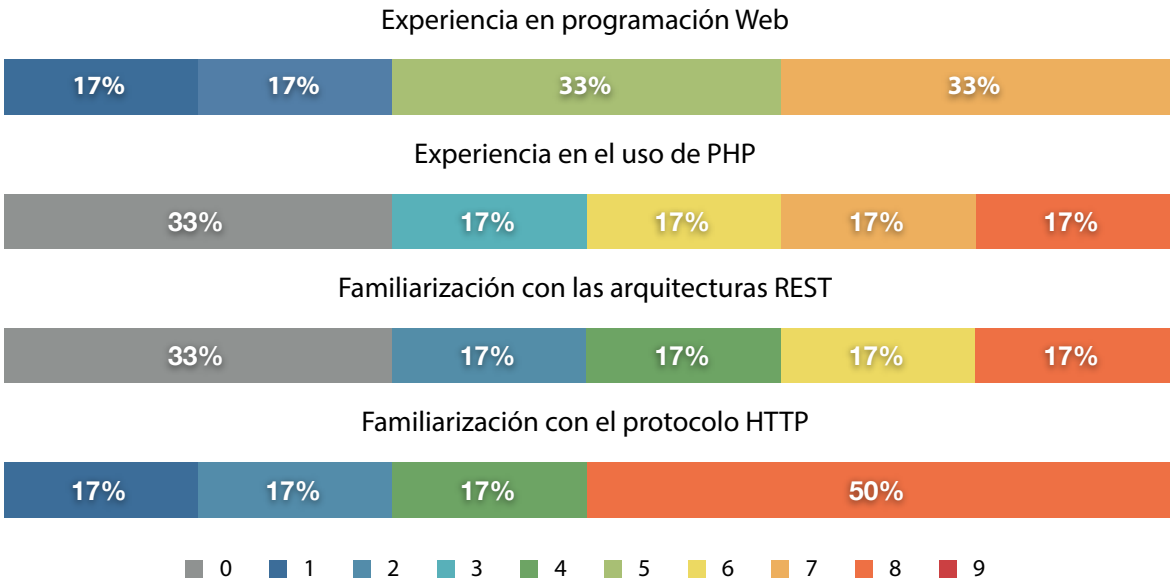


Figura 6.6: Experiencia o familiaridad de los participantes a diversos factores.

Partes del sistema

Esta sección del cuestionario tiene el objetivo de identificar si los participantes realmente comprendieron el propósito y naturaleza del sistema. Para ello primero se les pregunta directamente para qué se utiliza, qué producto se obtiene al utilizarlo y cuál es la notación con la que se trabaja. Además, se les pide que den un estimado acerca de la proporción del tiempo que dedicaron a:

- Buscar información dentro de la notación.
- Trasladar cantidades importantes de información de uno a otro lado en el sistema.
- Agregar cantidades pequeñas de información a una descripción previamente creada.
- Reorganizar y reestructurar descripciones previamente creadas.
- Jugar con nuevas ideas en la notación sin saber qué es lo que va a pasar.

Para empezar, se les pregunta: ¿para qué tarea o actividad se utiliza el sistema? Cuatro de seis participantes respondieron que para crear aplicaciones o servicios en escenarios de

cómputo ubicuo o de la Internet de las Cosas (IoT). Uno de ellos respondió que para interactuar con tecnologías de la IoT y otro más que se utiliza en sistemas de producción donde las circunstancias sean muy relevantes (sic). Como se puede ver, la mayoría estuvieron en lo correcto.

A la segunda pregunta: ¿cuál es el producto de utilizar este sistema? Los participantes respondieron que aplicaciones mashup, aplicaciones que utilizan REST, aplicaciones basadas en HTTP, sistemas de sensado, aplicaciones para administrar tecnologías IoT y servicios Web autónomos de sensado. Todas las respuestas son válidas.

Acerca de ¿cuál es la notación principal del sistema? Las respuestas fueron servicios, recetas, las conexiones entre servicios y código PHP. La notación principal del sistema es la relacionada a la creación de los manifiestos de aplicación, que son arrastrar y soltar representaciones gráficas de servicios, su interconexión y la definición de las recetas.

Finalmente, la Figura 6.7 muestra la distribución de las estimaciones del tiempo dedicado a distintos tipos de tareas. En promedio, el 15% del tiempo se dedicó a buscar información dentro de la notación, 20% trasladando información de una parte a otra en el sistema, 22% agregando pequeñas cantidades de información a algo previamente creado, 23% reorganizando o reestructurando al previamente creado y 20% jugando con nuevas ideas. Los resultados son muy similares como para obtener conclusiones claras de ellos. Sin embargo, dos de los participantes más experimentados con las aplicaciones Web y la programación PHP declararon haber dedicado 40% de su tiempo a jugar con nuevas ideas, lo que nos podría decir que el sistema ayuda a la evaluación progresiva como se discute más adelante. Además, la actividad con menor tiempo es la de buscar lo que nos podría decir que el sistema está organizado de manera que no es difícil encontrar las cosas que se necesitan para completar la tarea a la mano.

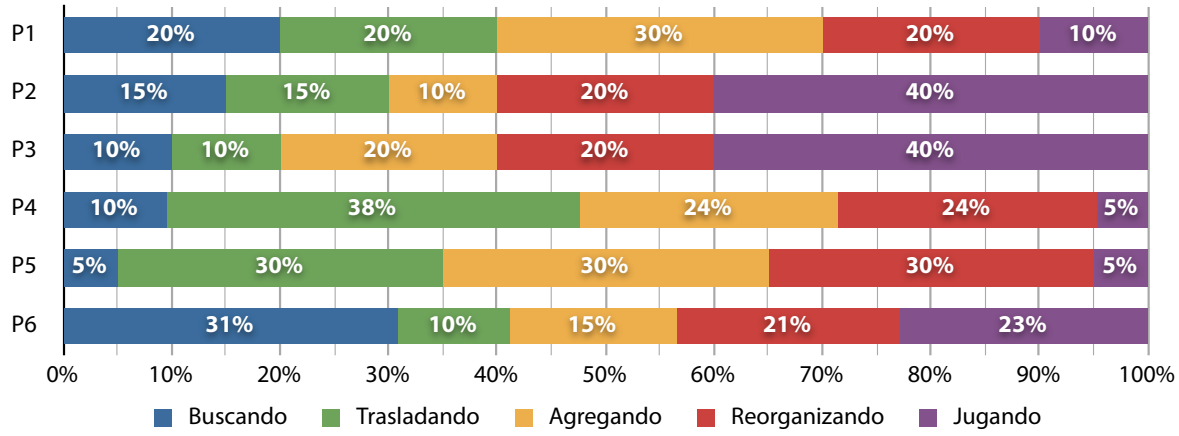


Figura 6.7: Estimación del tiempo dedicado a diferentes tareas.

Los resultados de esta sección se utilizan más adelante en el análisis de las respuestas correspondientes a cada una de las dimensiones.

Visibilidad y yuxtaposicionalidad

Esta dimensión se refiere a la habilidad de identificar fácilmente los componentes del sistema. También se refiere a la problemática para poder ver cosas al mismo tiempo cuando hay necesidad de ello. Para medir esta dimensión se le pide al participante que califique en una escala del cero al nueve la facilidad de observar o encontrar las diferentes partes de la notación, que diga cuál es el tipo de cosas que es más difícil de observar o encontrar y que si, cuando necesita comparar o combinar diferentes partes de la notación, puede verlas al mismo tiempo.

Cuatro de cinco participantes calificaron como ocho a la facilidad de observar o encontrar las diferentes partes de la notación. El cual es un resultado muy positivo. Sólo un participante contestó "*5, las hojas de instrucciones contenían lo necesario para la prueba, pero cualquier actividad extra requiere de conocimientos previos en PHP/Programación*", se decidió omitir esta respuesta ya que no se refiere directamente a la actividad de encontrar las

partes de la notación sino a la dificultad del participante para trabajar con ella dada su experiencia previa con el lenguaje. En cuanto a las cosas que son difíciles de observar o encontrar los participantes mencionan:

- Cuando hayan muchas recetas disponibles será difícil encontrar la que se necesita.
- El acceso a la edición de una receta podría ser más rápido.
- Cuando se está editando una receta se deja de ver el flujo de datos, lo que dificulta tener una idea del comportamiento general de la aplicación.
- La estructura de los datos de entrada o salida.

Como se puede ver, las respuestas se refieren directamente a problemáticas de la interfaz gráfica que pueden ser fácilmente resueltas y no al sistema en sí. Es decir, los participantes no encontraron dificultades en cuanto a identificar alguna parte de la notación que les impidiera realizar las tareas requeridas. Para ayudar a encontrar entre muchas recetas se puede implementar una búsqueda o filtrado, el acceso a la edición de una receta se puede realizar mediante una acción simple como un doble clic, el diálogo de edición se puede mover para permitir ver el flujo de datos completo y se puede agregar ayuda visual para que el que edita una receta pueda saber la estructura de los elementos que está utilizando. Cuando se necesitan comparar o combinar partes diferentes de la notación, la mitad de los participantes comentaron que sí es posible ya que las representaciones de los servicios están todas visibles cuando se está trabajando con la combinación o formación del flujo de datos. Sin embargo uno de ellos menciona que si el sistema es complejo, podría haber un problema de visibilidad al no poder ver el flujo completo. Dos participantes comentan que no es posible ver código de las recetas al mismo tiempo algo que es cierto pero que puede ser corregido fácilmente al mejorar la interfaz gráfica del sistema. El otro participante dijo que no es posible pero no explicó por qué.

Viscosidad

Esta dimensión se refiere a la facilidad para hacer un cambio en el sistema. La terminología viene de la mecánica de fluidos en donde se refiere a la oposición de un fluido a ser deformado por una fuerza tangencial. Para medir la viscosidad de un sistema se les pide a los participantes calificar en una escala de cero a nueve qué tan fácil es hacer un cambio a un trabajo anterior y el por qué, también se les pregunta si hay cambios que son más difíciles que otros.

En promedio, los participantes califican la facilidad de hacer cambios en el sistema con un ocho. Como razón mencionan que sólo hay que reorganizar las conexiones para cambiar la lógica de la aplicación o editar una receta. Un participante menciona que si acaso hay una dificultad, esta está relacionada a la falta de experiencia con la programación en PHP y no a un impedimento del sistema.

Cuatro de cinco participantes creen que no hay cambios más difíciles que otros. Sólo un participante menciona que sería difícil modificar una receta si no se sabe bien qué estructura tienen los elementos que se utilizan dentro de la receta. De nuevo, algo que puede ser solucionado modificando la interfaz gráfica. Como se puede ver, se podría concluir que la viscosidad del sistema es baja, es decir, no es difícil hacer cambios a un producto creado anteriormente.

Difusidad

La dimensión de difusidad se refiere a la expresividad del lenguaje. Es decir, si se puede expresar la acción o tarea que necesitan con poco o mucho uso de la notación. Para medirla se les pregunta a los participantes si la notación les permitió expresar lo que querían con una brevedad razonable o si fue tediosamente extensa y qué tipo de cosas toman más espacio para describirlas.

Todos los participantes mencionaron que la notación les permitía hacer lo que querían con brevedad. Uno de ellos menciona que son las conexiones lo que te ayuda a ser breve cuando se está definiendo la lógica de una aplicación.

En cuanto a las cosas que toman más espacio en definir sólo mencionaron la escritura en sí de una receta. La razón dicen que se debe a que se necesita escribir mucho para extraer datos de un servicio de entrada y escribirlos como salida. Para resolver este problema se podría trabajar en la inclusión de mejores extensiones de PHP que ayuden a tratar estructuras de datos con pocas líneas de código.

Operaciones mentales difíciles

Esta dimensión se refiere a si el sistema requiere que los desarrolladores efectúen operaciones mentales difíciles al estar trabajando en un producto. Esta es una de las causas principales que afectan la usabilidad de un sistema. Para medir la dimensión se les pregunta a los participantes qué cosas son las que requieren más esfuerzo mental en la notación y si existen cosas que parezcan ser más complejas de lo que se necesita.

Los participantes identificaron al trabajo de creación o edición de recetas como la actividad principal que requiere más trabajo mental. Esto debido a que dicha actividad no sólo involucra la notación gráfica sino también la textual al tener que conocer la estructura de los datos y la relación lógica que tendrán los servicios. La mitad de los participantes creen que la causa es el tener que recordar los nombres de los atributos y elementos de los datos de entrada y salida.

En cuanto las cosas que parecen ser más complejas de lo que podrían ser los participantes mencionan la asignación de las recetas a cada servicio. La idea de tener el editor de aplicaciones es que este almacene las recetas que se han utilizado con el tiempo para que estén disponibles al crear nuevas aplicaciones. Las recetas apropiadas para una co-

nexión en el flujo de datos se seleccionan automáticamente, sin intervención del usuario. Durante la tarea de implementación ninguna de las recetas necesarias estaba previamente disponible, los participantes debieron programarlas. Tal vez sea esta la razón por la que encontraron demasiado compleja la asignación de recetas a los servicios.

Propensión al error

La dimensión de la propensión al error se refiere a qué tan fácil es cometer errores utilizando la notación. Una notación con mal diseño podría causar más problemas del que se quiere resolver. Para medirla se les pregunta a los participantes si encontraron algún tipo de error que fuera común o fácil de cometer y si se encontraron cometiendo pequeños errores que los irritaran o hicieran sentir mal.

Los errores en la notación que los participantes encontraron fueron el hacer conexiones entre servicios que no tenían nada que ver en la implementación de la funcionalidad solicitada y errores en la sintaxis al trabajar con las recetas. La problemática de conexión entre servicios que no son compatibles no fue cubierta en el prototipo que utilizaron los participantes durante el experimento de evaluación. El diseño del editor contempla la introducción de animaciones y alertas gráficas al intentar conectar dos servicios que no pueden ser conectados ya sea por restricciones lógicas o por las características mismas de los servicios.

En cuanto a los pequeños errores, los participantes mencionan el olvidar escribir un punto y coma al final de las líneas en las recetas y escribir correctamente el nombre de una variable. De nuevo, el diseño del editor contempla la validación de la sintaxis al escribir las recetas pero esta funcionalidad no fue incluida en el prototipo utilizado por los participantes.

Cercanía del mapeo

La cercanía del mapeo se refiere a qué tanto se acerca la notación, es decir la representación de la solución al problema, al dominio de aplicación del sistema. Para revisar la dimensión se les pregunta a los participantes qué tan cercanamente relacionada está la notación al resultado que describieron y qué partes parecen ser difíciles de describir.

Todos los participantes manifestaron que el sistema se acerca bastante al dominio de aplicación gracias a la representación gráfica de servicios y sus conexiones.

En cuanto a las partes difíciles de describir mencionan que no existe tal problema una vez que se han familiarizado con la notación. Uno de los participantes opinó que tal vez la estructura de los datos pertenecientes a diferentes servicios podrían presentarse de una manera similar, así no existiría el problema de conocer estructuras diferentes según los servicios con los que se interactúan.

Expresividad del rol

Esta dimensión se refiere a si es posible identificar claramente cuál es el objetivo y propósito de cada elemento en la notación. Para medirla se les pregunta a los participantes si es fácil decir cómo participa cada parte de la notación en la conformación del objetivo final, si hay algunas partes que sean particularmente difíciles de interpretar y si hay partes que no sabían qué significaban pero que las incluyeron sólo porque siempre se habían incluido.

Todos los participantes manifestaron que es fácil saber qué hace cada parte de la notación en la conformación del objetivo final. La razón es que la notación gráfica es muy intuitiva y permite conocer el vínculo y la relación entre los servicios.

En cuanto a las partes difíciles de interpretar mencionan la estructura de los datos de entrada y el saber realmente qué hace cada receta, que si bien fue fácil durante el desarrollo del experimento, esto se puede complicar en aplicaciones más complejas.

En cuanto a las partes de la notación que no sabían qué significaban pero que las incluían porque lo habían visto hacer anteriormente mencionan la suscripción a los servicios y el uso de la lista de datos de entrada. Sin embargo, mencionan sólo fue inicialmente ya que lograron entender el significado de las mismas durante el desarrollo de las tareas.

Dependencias ocultas

La dimensión de las dependencias ocultas se refiere a las dependencias entre partes de la notación que ni son visibles ni están explícitamente señaladas. Para medirla se les pregunta a los participantes si todas las dependencias entre partes de la notación están visibles, que si pudieran existir problemas si el producto requiriera de una descripción larga, y si las dependencias entre partes siempre se mantuvieron iguales o si estas cambiaron durante el desarrollo.

Todos los participantes opinan que las dependencias entre servicios están claramente visibles gracias al uso de un grafo para representarlas gráficamente.

La mitad de los participantes cree que si el producto que se desea construir fuera complejo y requiriera de una descripción particularmente larga existiría un problema de visibilidad al no poder observar el grafo del flujo de datos por completo. Recomiendan evitar la introducción de *scroll* en el editor ya que ocultaría parte del grafo. Para evitar esto se podrían implementar diferentes formas de visualización basadas en el uso de *zoom* y agrupamiento dinámico de servicios.

En cuanto a si las dependencias permanecen iguales o hay algunas acciones que las bloqueen, todos los participantes creen que no existe tal problemática.

Evaluación progresiva

Esta dimensión se refiere a si los desarrolladores pueden detenerse en cualquier momento durante la creación de un producto y verificar su progreso. Para medir la dimensión se les pregunta a los participantes si es fácil detenerse a la mitad de la creación con la notación y revisar el trabajo en el estado en que quedó, si puede decir qué tanto progresó o en qué etapa del trabajo se está, y si es posible probar versiones parcialmente completadas del producto.

Todos los participantes manifestaron que sí es posible ir evaluando progresivamente las aplicaciones; además que es muy fácil ir probando fácilmente parte por parte para corregir el comportamiento de la aplicación según se vaya necesitando.

De igual manera, todos los participantes manifestaron como afirmativa la posibilidad de decir qué tanto progreso se ha hecho o en qué etapa del trabajo se encuentran. Uno de los participantes mencionó que el progreso se puede visualizar directamente gracias al resaltado de las conexiones en verde o gris con el que se indica la existencia de una receta para los puntos en el flujo de datos.

Finalmente, todos afirman que es posible probar versiones parcialmente completadas del producto. Uno de los participantes afirmó haberlo hecho para probar el correcto funcionamiento del servicio para publicar en Twitter.

Provisionalidad

La dimensión de la provisionalidad se refiere al grado de compromiso que debe hacer el desarrollador mientras realiza acciones o marcas con la notación. Es decir, si es posible jugar con ideas cuando no se está seguro de qué hacer. Para medir esto, se les pregunta a los participantes si la notación permite ser vago cuando no se sabe cómo proceder y se le

pide mencionar qué tipo de cosas puede hacer cuando no quiere ser muy preciso acerca del resultado exacto que se intenta obtener.

Todos los participantes manifestaron que sí es posible experimentar, jugando con ideas cuando no se está seguro de qué hacer para completar el objetivo de una aplicación. Como razones citan la posibilidad de experimentar al conectar servicios de distinto tipo para ver cómo reaccionan las recetas. Uno de los participantes destacó que las características visuales de la notación lo animaron a experimentar con ella.

En cuanto a las cosas que pueden hacer cuando no quieren ser precisos acerca del resultado mencionan: manipular recetas, especificar en primera instancia la notación gráfica para luego escribir las recetas, evaluar partes de la aplicación por separado y buscar cómo utilizar recetas ya disponibles.

Compromiso prematuro

Esta notación se refiere a si el orden de las acciones que realizan los desarrolladores está predeterminado o limitado por la misma notación. Para analizar la dimensión se les pregunta si creen poder hacer el trabajo que se les solicitó en cualquier orden, o si el sistema los obliga a pensar antes para tomar decisiones y, si así es el caso, qué tipo de problemas les podría causar en su trabajo.

Todos los participantes manifestaron que el sistema no los obliga a llevar un orden preestablecido cuando están trabajando. Sin embargo, la mayoría cree que lo más conveniente al utilizar el sistema es primero especificar los servicios con sus conexiones y entonces crear o modificar las recetas según se necesite. Como posible problemática uno de los participantes menciona que tal vez un desarrollador pudiera conectar los servicios de una manera tal que dificulte de sobre manera la escritura de las recetas para cumplir con el objetivo.

Consistencia

La dimensión de la consistencia se refiere a la utilización de una semántica similar en partes de la notación del sistema. Para revisar esta dimensión se les pregunta a los participantes por las partes de la notación que significan cosas similares y si hay algunos lugares en donde las cosas digan ser similares pero la notación las haga diferentes.

Dos de los participantes no encontraron notaciones similares en el sistema. Uno de ellos refiere a los servicios que visualmente son todos iguales. El resto se refiere a las recetas ya que la definición de su lógica es muy similar.

En cuanto a las cosas que parecen ser iguales pero no lo son se encuentran los servicios ya que estos parecen ser iguales visualmente pero, por ejemplo, es imposible distinguir entre los servicios que notifican eventos y los que no. De igual manera con las recetas, se escriben de manera similar, con el mismo nombre de variables, pero la estructura de los datos difiere según el servicio utilizado.

Notación secundaria

La notación secundaria se refiere a información extra creada por los desarrolladores en medios diferentes a los de la sintaxis formal introducida por el sistema. Para analizar esta dimensión se les pregunta a los participantes si el sistema les permite hacer notas o agregar comentarios, qué cosas escribirían o dibujarían si la notación estuviera escrita en papel, y si han agregado marcas adicionales o comentarios adicionales para clarificar o enfatizar algo.

Sólo dos de los participantes afirman que es posible hacer notas pero sólo dentro de la definición de las recetas. Al estar descritas como *scripts* en PHP, podrían utilizar líneas

de comentario propios de PHP. Los demás manifestaron no haber visto la posibilidad o no haberlo intentado hacer.

Si la notación estuviera en papel, todos los participantes coinciden en que primero dibujarían lo servicios utilizando texto y cuadros y después las conexiones utilizando rayas. Escribirían notas encima de las rayas para definir las interacciones. Como se puede ver, harían algo muy parecido a lo que ya es posible en el sistema.

Ningún participante realizó marcas adicionales para clarificar o enfatizar alguna idea.

Administración de la abstracción

Esta dimensión se refiere a la posibilidad por parte de los desarrolladores de introducir nuevas posibilidades o términos dentro de la notación, que les permita describir nuevas cosas o expresar ideas más claramente. Para analizarlo, se les pregunta a los participantes acerca de si esto es posible y si el sistema insiste en iniciar definiendo nuevos términos antes de poder hacer cualquier otra cosa.

Todos los participantes afirman que es posible introducir nuevas abstracciones en la forma de recetas y nuevos tipos de servicios. Uno de los participantes sugiere agrupar visualmente partes de la aplicación (grupos de conexiones) y formar grupos que puedan interconectarse como servicios. Esta funcionalidad es parte del diseño de nuestras contribuciones pero no formó parte del prototipo utilizado durante la experimentación.

Acerca de si el sistema los obliga a definir primero nuevos términos antes de poder hacer cualquier otra cosa, la mayoría de los participantes declaran como inexistente dicha obligación.

Respuestas adicionales

Para finalizar el cuestionario se le hicieron dos preguntas adicionales a los participantes. La primera acerca de si utilizaron la notación en formas inusuales que el diseñador no esperaba. La segunda, si pueden pensar en formas obvias en las que el sistema pudiera ser mejorado.

Entre las formas inusuales del uso de la notación se encuentran el tratar de conectar los servicios por lados incorrectos (es decir, salidas en salidas, entradas con entradas), olvidar guardar la aplicación al terminar de desarrollarla y un sin número de errores de sintaxis al crear o editar recetas.

Como sugerencias de mejora piden poder agregar servicios externos al prototipo (durante el experimento la lista de servicios era fija), las conexiones deberían indicar la dirección (el flujo de datos es en realidad un grafo dirigido pero visualmente no se observa), diferenciar más claramente las conexiones válidas de las inválidas, poder editar una receta directamente en el lugar en que se use y contar con un modo de prueba que corra la aplicación paso a paso deteniéndose en cada receta.

6.2 Evaluación funcional

Utilizando el análisis de los resultados de la aplicación del marco de trabajo de las dimensiones cognitivas, en esta sección se discute el soporte de cada uno de los requerimientos de diseño que condujeron el presente trabajo de tesis. Entre las discusiones de los requerimientos se explican aspectos tales como las decisiones de diseño, la experimentación llevada a cabo y las conclusiones acerca del cumplimiento de cada requerimiento.

- **Invisibilidad.** *Cualquier dispositivo o servicio que sea utilizado para aumentar un ambiente inteligente debe estar totalmente escondido a los usuarios.* Nuestras contribuciones promueven la implementación de servicios que reaccionan y apoyan a los usuarios sin necesitar de interacción explícita con los mismos. Esto gracias al soporte para generar e intercambiar eventos y a las máquinas de ejecución que se activan para provocar acciones cuando hay cambios en el flujo de datos de las aplicaciones. Los servicios Web tradicionales no cuentan con dichos mecanismos pero se podrían implementar a través de una tarea de desarrollo que no es sencilla y que requeriría de conocimientos especializados por parte de los desarrolladores. El soporte del marco de trabajo para invisibilidad se ejemplifica en la Sección 5.3.3 donde se detalla un escenario de aplicación experimental en donde los usuarios (personas en sillas de ruedas) utilizan un servicio de identificación de comandos de voz para interactuar con puertas y ventanas en el ambiente en donde se encuentren. Los usuarios no están conscientes de los motores en las puertas y ventanas, ni de los nodos de la red inalámbrica que continuamente monitorean el sonido ambiental en busca de comandos de voz. Tampoco están enterados de la infraestructura de *hardware* y *software* que hace funcionar todo.

- **Movilidad.** *Los usuarios no están limitados a un sólo ambiente o área física.* En nuestras contribuciones, un usuario no necesita estar limitado a un espacio determinado para poder utilizar los servicios de un ambiente aumentado. Ni tampoco necesita conocer o configurar algo antes de hacer uso de un nuevo ambiente aumentado. Como se ejemplifica en la Sección 5.2.2 en donde se presenta el editor de *mashups*, un usuario sólo necesita entrar al área física en donde un nuevo ambiente aumentado está disponible para interactuar con los servicios físicos y virtuales que hay en él. No sólo eso, sino que también puede utilizar los servicios y datos de ése ambiente y combinarlos con los de ambientes que previamente visitó o de los que conoce su existencia, incluso con los de proveedores externos o en línea. La infraestructura de

servicios del marco de trabajo está diseñada para proveer dicha funcionalidad ya que incluye como parte integral el descubrimiento de nuevos servicios y la inclusión de servicios genéricos que podrían estar disponibles en un momento dado dentro del flujo de datos de las aplicaciones.

- **Consciencia de contexto.** *Todo espacio debe estar equipado con sensores y con una infraestructura inalámbrica con la que se pueda monitorear y estar al tanto de información útil para la tarea que se esté llevando a cabo.* Una parte integral de la infraestructura de *hardware* y *software* del marco de trabajo es la de permitir la adquisición, procesamiento y agregación de datos de sensado que puede posteriormente ser utilizada en flujos de datos que a su vez generan eventos o información de contexto más compleja y especializada. Para más información acerca de esta funcionalidad véase la Sección 5.1.2.

- **Anticipación.** *Los sistemas deben ser capaces de actuar en representación de sus usuarios sin intervención explícita.* Utilizando la adquisición de contexto y el mecanismo de notificación de eventos, nuestras contribuciones permiten desarrollar aplicaciones en las que la funcionalidad se basa en la detección y activación de actividades en representación de los usuarios. Como ejemplo véase la Sección 5.3.2 en donde los usuarios son pacientes que deben ser constantemente monitoreados para evitar caídas y actividades peligrosas. En este escenario de aplicación cada usuario trae consigo un sensor de movimiento con el que se puede inferir qué actividad está realizando. En la eventualidad de una caída, un evento de alerta se dispara que es a su vez utilizado por los servicios y aplicaciones que se encuentra en el ambiente aumentado para brindar ayuda al usuario.

- **Comunicación natural.** *Los usuarios deben comunicarse e interactuar con el ambiente por medio de los dispositivos y objetos a los que están acostumbrados en su vida diaria.* Si bien nuestro marco de trabajo no se enfoca directamente en la creación de dispo-

sitivos, sí permite su desarrollo e integración al utilizar los diferentes mecanismos y facilidades disponibles en los ambientes aumentados. Por ejemplo véase la Sección 5.3.1 en donde se analiza un escenario de aplicación en la que el usuario utiliza las botellas de pastillas a las que siempre ha estado acostumbrado para saber si debe tomar o no su medicina. Esto se hace mediante el uso de una herramienta de visión aumentada implementada con nuestro marco de trabajo. El escenario de uso es simple, el usuario toma una de las botellas, la observa a través de un teléfono inteligente y visualmente aparece, encima de la imagen en video de la botella, si debe esperar una cierta cantidad de minutos o si ya se tomó la dosis indicada por el médico.

- **Adaptabilidad.** *Los sistemas deben estar diseñados para resolver anormalidades o situaciones excepcionales al adaptarse por sí mismos en tiempo de ejecución.* Si bien este requerimiento no funcional no fue directamente atendido durante el diseño de nuestras contribuciones, se puede decir que es efectivamente soportado. Las aplicaciones en nuestro marco de trabajo pueden utilizar servicios genéricos, es decir, servicios de los que no se conoce su ubicación concreta sino cuál es la funcionalidad o tipo de datos que proveen. De esta manera, si un servicio deja de funcionar debido a cualquier razón, otro servicio que provea la misma funcionalidad puede ser automáticamente detectado y utilizado como reemplazo. Además, muchos de los mecanismos del marco de trabajo están diseñados para que puedan seguir funcionando a pesar de problemas. Por ejemplo, el mecanismo de notificación de eventos. En lugar de que un servicio que provee eventos notifique directamente a los servicios interesados, existe una entidad a la que se inscriben los servicios interesados y es esta entidad la que realiza las notificaciones. En caso de problemas con dicha entidad, el ambiente puede utilizar múltiples notificadores de servicios para balancear la carga o para reemplazar servicios problemáticos.

- **Robustez y disponibilidad.** *Los sistemas deben continuar funcionando a pesar de cualquier mal uso o error.* Como se explica anteriormente, las aplicaciones pueden indicarle al ambiente que necesitan de cierta funcionalidad o tipo de datos sin decirle directamente quién se la va a proveer. De esta manera, el ambiente puede escoger el mejor proveedor de acuerdo a múltiples condiciones tales como si el servicio tiene menos carga o si su latencia empezó a incrementarse en los últimos minutos.

- **Extensibilidad.** *Se debe permitir la fácil incorporación de nuevos componentes o recursos en tiempo de ejecución.* Nuestras contribuciones permiten la fácil integración de nuevos servicios y la incorporación de nueva funcionalidad. Las aplicaciones en nuestro marco de trabajo están desarrolladas en base a servicios que implementan interfaces genéricas. Para integrar un nuevo servicio sólo hace falta indicar cuáles de esas interfaces genéricas implementa para que un ambiente aumentado que ya está en funcionamiento pueda integrarlo en tiempo de ejecución. De igual manera pueden agregarse más interfaces genéricas.

- **Seguridad.** *Los sistemas deben hacer exactamente el trabajo para el que fueron diseñados.* Como se menciona anteriormente, las aplicaciones pueden seleccionar, en tiempo de ejecución, a los servicios o instancias que les proveerán una funcionalidad o información necesaria. Es por ello que se pueden desarrollar aplicaciones basadas en nuestro marco de trabajo que siempre puedan cumplir con la funcionalidad para que fueron diseñadas.

- **Protección.** *Los sistemas deben garantizar un grado definido de privacidad para las personas o dispositivos bajo observación.* Si bien la privacidad es un requerimiento no funcional que no fue directamente utilizado durante el desarrollo del marco de trabajo, sí se tuvo en cuenta durante la implementación de los servicios básicos tales como la geolocalización bajo techo. Como se ejemplifica en la Sección 5.1.1, para poder aproximar la posición de un usuario este debe llevar consigo un dispositivo que

constantemente obtiene lecturas acerca de la intensidad de señal hacia puntos de acceso inalámbricos. El servicio que procesa las señales y provee una aproximación de la posición está diseñado de manera que no almacene ni el estado ni la información de sus usuarios (a menos que el usuario así lo quiera para por ejemplo mejorar la estimación).

- **Oportunidad.** *Los sistemas deben entregar información oportunamente cuando cualquier retraso o demora pudiera afectar su propósito.* Para cumplir con este requerimiento se introdujo el mecanismo de notificación de eventos. El mecanismo no sólo envía las notificaciones, este implementa un protocolo que permite utilizar múltiples instancias de notificación de eventos que incluso pueden ser utilizadas de manera escalonada para distribuir la carga y asegurar que cada notificación será entregada a tiempo. Véase la Sección 4.4 para más detalles.

- **Eficiencia.** *Los sistemas deben administrar y utilizar los recursos de manera eficiente.* Todo mecanismo y elemento en la infraestructura tanto de *hardware* como de *software* está diseñado para para eficientar los recursos disponibles. Se hace uso de técnicas tales como el almacenar temporalmente respuestas a las solicitudes de servicio, la persistencia de conexiones entre servidores y clientes, la compresión de datos al intercambiarse, entre otras.

- **Agnóstico al lenguaje.** *Los sistemas deben poder ser escritos en cualquier lenguaje de programación y dejar la selección del mismo al desarrollador.* El marco de trabajo está basado enteramente en servicios y la comunicación está basada en la Web. Prácticamente todos los lenguajes de programación modernos tienen posibilidades de comunicación con la Web así que los desarrolladores pueden utilizar los mismos lenguajes y herramientas a los que están acostumbrados.

- **Eliminar barreras.** *Los sistemas deben eliminar barreras de implementación de bajo nivel al proveer alternativas sencillas de alto nivel de abstracción.* Todo elemento en la

infraestructura de nuestro marco de trabajo está modelado como un servicio REST. De esta forma, la interacción con cualquiera de esos servicios es la misma. Sólo difieren en la estructura de datos que intercambian, estructura que debe seguir una forma genérica predefinida para permitir la composición. Como ejemplo de cómo se promueve la eliminación de barreras véase la Sección 5.1.2 en donde se describe el servicio de sensado. Dicho servicio provee lecturas y notificaciones de sensado por medio de una simple solicitud GET. Lo que en realidad sucede es que una red inalámbrica de sensores está funcionando y manteniendo esquemas de comunicación compleja, cuando es hora de que uno de los nodos notifique un cambio, los mismos nodos en la red se coordinan hasta que el cambio llega a un nodo especial, denominado *gateway* que es el que finalmente alimenta al servicio de sensado. El usuario no está consciente de tal comunicación, él sólo solicita y utiliza la lectura de sensado.

- **Minimizar mantenimiento.** *Reducir a un mínimo las tareas de mantenimiento y no esenciales.* Muchos sistemas distribuidos necesitan a veces de acciones de mantenimiento que deben ser llevadas a cabo de vez en cuando. La infraestructura de nuestro marco de trabajo está diseñada para evitar dichas acciones. Por ejemplo, cuando es necesario reemplazar o agregar un nuevo dispositivo físico el ambiente, sólo debe encender para que sea detectado por la infraestructura. Siempre y cuando exista un servicio REST que sirva para representar y controlar dicho dispositivo y que utilice las facilidades del marco de trabajo para registrarse en el ambiente.

- **Encapsular patrones.** *El ambiente debe encapsular conceptos de diseño y patrones que han comprobado su efectividad con el objetivo de que puedan ser incluidos en los sistemas con muy poco esfuerzo de implementación.* El diseño de nuestras contribuciones fue basado en diferentes conceptos y patrones de diseño. Nuestro marco de trabajo encapsula patrones tales como la orientación a servicios, el modelo de comunica-

ción *publish/subscribe*, el uso de hipermedia para dirigir la lógica de la aplicación, entre otros. Gracias a que todos los elementos del marco de trabajo están modelados como servicios, se pueden encapsular muchos más patrones ya que se puede ver a un servicio como una "caja negra" dentro de la que se pueden implementar los patrones.

- **Interfaz de programación sencilla.** *El ambiente deberá presentarse a sí mismo de manera simple y concisa que promueva y siga la forma en las que las personas piensan acerca del dominio de aplicación permitiendo que se enfoquen en la solución y no en la problemática de bajo nivel.* Las aplicaciones en nuestro marco de trabajo están principalmente conformadas por la composición de datos y lógica a partir de servicios independientes. Por ello, se ha cuidado el diseño de la interfaz de programación para que los desarrolladores mantengan la idea de que están trabajando con un flujo de datos y lógica en el que intervienen servicios. Como se discute en la Sección 6.1.9 en la que se analizan los resultados de un experimento de evaluación con usuarios potenciales, los desarrolladores han encontrado que la interfaz de programación se acerca bastante al dominio de la aplicación.

- **Brevedad.** *Hacer que las cosas simples se puedan realizar con pocas líneas de código.* De igual manera que la interfaz de programación se diseñó para que estuviera cercana a el cómo piensan los desarrolladores acerca del dominio de aplicación, se procuró que la notación que utilizaran fuera lo más breve posible. Los resultados se pueden observar en las Sección 6.1.9 en la que se analiza la difusidad de la notación. De igual manera la Sección 6.1.9 trata acerca de la dificultad de hacer cambios en un sistema previamente creado.

- **Ensamblajes transitorios.** *El ambiente debe permitir crear aplicaciones simples cuya existencia esté limitada a un conjunto particular de usuarios por un periodo de tiempo limitado.* Esto es posible en nuestro marco de trabajo gracias al uso del editor de *mas-*

hups (véase la Sección 5.2.2). Al utilizarlo los usuarios pueden llegar a un ambiente aumentado y crear aplicaciones que utilicen los servicios y datos que están disponibles a su alrededor. Dichas aplicaciones pueden ser desde pequeñas pruebas de conceptos hasta escenarios de composición complejos.

- **Resultados inmediatos.** *Los resultados deben ser utilizables inmediatamente para que los desarrolladores puedan experimentar con diferentes alternativas.* Este es uno de los requerimientos a los que se les dio más importancia durante el diseño de nuestro marco de trabajo. La posibilidad de evaluar un desarrollo mientras está en proceso garantiza que sean más efectivos al poder probar diferentes soluciones. La Sección 6.1.9 discute cómo un grupo de usuarios potenciales efectivamente pudieron comprobar dicha funcionalidad a través del uso de un prototipo del editor de *mashups*.

6.3 Respuestas a las preguntas de investigación

Una vez analizado el cumplimiento de los requerimientos de diseño y después de conocer los resultados de la experimentación de nuestras contribuciones por parte de un grupo potencial de usuarios, se responde a cada una de las preguntas de investigación que orientaron el presente trabajo.

1. *¿Cómo integrar sensores, actuadores y otros aparatos en una infraestructura en común que tome en cuenta aspectos tales como la necesidad de notificación de eventos y el control de acceso?*

   Mediante la combinación del modelo de orientación a servicios junto con el de sistemas basados en contexto. Los sensores y actuadores deberán ser modelados como servicios estandarizados cuyas interfaces de programación sean similares para cualquier servicio. Dichos servicios deberán poder ser descubiertos automáticamente

dentro de un ambiente aumentado utilizando un protocolo estándar de descubrimiento. Para comunicar lecturas de sensado o mensajes de control desde los actuadores los servicios deberán implementar el modelo de comunicación *publish/subscribe* para notificar a las entidades interesadas acerca de una nueva actualización en los recursos. Para implementar el control de acceso, cada servicio deberá contar con *tokens* de autentificación para identificar a los usuarios y en su caso limitar el acceso a cierta funcionalidad. De igual manera, utilizando el mismo control de acceso, se deberá denegar el acceso concurrente a servicios cuyos recursos estén directamente representando a una entidad física que no pueda ser utilizada por más de un usuario al mismo tiempo.

2. *¿Cómo podría simplificarse la composición de fuentes de datos y servicios de manera que esta pueda ser descrita e implementada por desarrolladores no expertos?*

   Ya que los servicios cuentan con una interfaz de programación estandarizada basada en REST, lo natural sería pensar en utilizar algún lenguaje de composición de servicios como por ejemplo BPEL. Sin embargo, dichos lenguajes requieren que los desarrolladores cuenten con experiencia en aspectos tales como la interpretación de representaciones, la extracción de datos a partir de recursos Web, la especificación exacta de los parámetros y estructura que conforman una representación, entre otros aspectos técnicos de bajo nivel. Para evitarlo, proponemos un lenguaje de composición simplificado de alta granularidad que cualquier desarrollador puede utilizar incluso sin tener mucha experiencia de programación en la Web. Esto se demuestra en el resultado del experimento de evaluación donde sólo dos de seis participantes tenían tal experiencia pero no tuvieron dificultades para completar las tareas de implementación (véase la Sección 6.1.9). Adicionalmente se ha trabajado en herramientas tales como el editor de *mashups* (véase la Sección 5.2.2) que permite la definición de composiciones al simplemente arrastrar y conectar representaciones gráficas de los servicios, incluso requiriendo cero líneas de código en algunos casos.

3. *¿A qué características y requerimientos especiales de los ambientes inteligentes se les dará soporte en este trabajo?*

   A las características reportadas por Weber *et al.* (2005) que son las comúnmente exhibidas por sus aplicaciones, y a los requerimientos no funcionales reportados por Nehmer *et al.* (2006). El soporte a las características y los requerimientos no funcionales conformaron los requerimientos de diseño del presente trabajo de tesis. Véase la Sección 3.1 para una descripción y discusión a detalle de cada requerimiento.

4. *¿Qué tipo de escenarios de aplicación resultan más apropiados para el soporte basado en la Web, cuáles son los más problemáticos?*

   Aquellos en los que sea posible separar la lógica de aplicación en elementos independientes que no necesiten mantener el estado de la aplicación. Más aún si el escenario involucra el uso de la aplicación por parte tanto de usuarios como de entidades de *software*. Se podría decir que prácticamente cualquier escenario de aplicación puede ser implementado a través de la Web. El problema radica en la conveniencia y practicidad de hacerlo. Por ejemplo, durante el desarrollo de prototipos y experimentación encontramos que el soporte basado en Web no es conveniente para aplicaciones que requieran la notificación inmediata de eventos o aquellas cuya tasa de comunicación sea elevada. También existen problemas de privacidad y seguridad que si bien pueden ser implementados, estos deben ser analizados para cada escenario de aplicación en específico. Cada escenario tiene características de privacidad o seguridad especiales.

5. *¿Qué protocolos, lenguajes de programación y otras tecnologías Web son adecuadas para desarrollar un marco de trabajo para ambientes inteligentes?*

   Necesariamente un marco de trabajo para ambientes inteligentes deberá contar con un mecanismo para la identificación de los servicios en términos de lo que hacen, una plataforma de comunicación en común, un mecanismo para el descubrimiento

de servicios y un método de notificación de eventos que pueda ser utilizado por todas las entidades del marco de trabajo. En nuestras contribuciones hemos utilizado los siguientes protocolos, lenguajes y tecnologías Web:

- *Identificación de servicios.* Para definir qué hace cada servicio en términos de qué puede proveer y lo que hacen en términos de su lógica hemos utilizado el lenguaje WADL (Handley, 2009) para definir interfaces de programación genéricas. En lugar de hacer que un servicio implemente una interfaz, se ha permitido que un servicio implemente múltiples interfaces para promover mayores posibilidades de composición y expresividad.
- *Plataforma de comunicación en común.* Todos los servicios y elementos en nuestro marco de trabajo siguen el modelo REST (Fielding, 2000) y utilizan los mismos esquemas genéricos para las representaciones que intercambian entre ellos. Gracias al uso de REST sólo existen cuatro operaciones básicas: *create*, *retrieve*, *update* y *delete*. Las operaciones se aplican a recursos de los que se obtienen o a los que se les envían representaciones. Las representaciones siguen los esquemas genéricos.
- *Descubrimiento de servicios.* Se ha seleccionado el protocolo Zeroconf para proveer el mecanismo de descubrimiento. La razón es que gracias a su popularidad existe una gran diversidad de bibliotecas para diferentes plataformas y herramientas que pueden ser utilizadas por los desarrolladores. Además, los sistemas operativos móviles de última generación ya la incluyen en sus SDKs.
- *Notificación de eventos.* Para el mecanismo de notificación de eventos se ha utilizado el protocolo PubSubHubub. Este protocolo introduce una nueva entidad en el marco de trabajo (llamada *hub*) que se encarga de administrar las suscripciones y en envío de notificaciones cuando un recurso se actualiza. Entre sus múltiples beneficios se encuentra la de poder utilizar múltiples *hubs* de-

pendientes unos de otros para controlar la carga de los servidores y permitir la escalabilidad.

6. *¿Cuál es el proceso que un desarrollador de aplicaciones de ambientes inteligentes debe seguir para construir este tipo de sistemas utilizando el marco de trabajo?*

   Primero, debe analizar qué servicios y mecanismos ya disponibles en el marco de trabajo podría utilizar para completar el escenario de aplicación que necesita implementar. Si el escenario necesita de nuevos dispositivos o servicios entonces el desarrollador necesitará introducirlos en el marco de trabajo siguiendo los principios de diseño de nuestras contribuciones. Una vez hecho esto, deberá crear un manifiesto de aplicación incluyendo una lista de los servicios involucrados y las interconexiones de los mismos con lo que se define el flujo de datos y lógica en la aplicación. Ya con el manifiesto listo, el desarrollador debe enviarlo a una de las máquinas de ejecución del marco de trabajo en donde será ejecutada su aplicación. Si bien este proceso no es complejo, los desarrolladores noveles podrían encontrar alguna dificultad en seguirlo. Esta es la razón detrás de la inclusión del editor de *mashups* como parte de nuestras contribuciones.

7. *¿Qué tan adecuada es la abstracción que ofrecen los mashups de servicios Web para los desarrolladores de aplicaciones en ambientes inteligentes?*

   De acuerdo a los resultados del experimento de evaluación, los *mashups* de servicios Web son una abstracción muy adecuada para los desarrolladores de ambientes inteligentes. Esto debido a que la notación utilizada en dichos sistemas es muy cercana al cómo piensan realmente los desarrolladores acerca de cómo los datos entran y salen de los servicios Web. Cuando se les preguntó a los participantes cómo harían para describir un sistema si sólo pudieran utilizar papel y lápiz estos respondieron que sería algo muy similar a lo que aparece gráficamente en un sistema *mashup*, lo que ejemplifica lo apropiado que resulta la abstracción. Adicionalmente, los *mas-*

hups logran esconder efectivamente la complejidad de la coordinación de solicitudes, respuestas, eventos, composición de datos, entre otras actividades que de otra manera el desarrollador hubiera tenido que realizar por sí mismo.

8. *¿Qué modelo de interacción de usuario seguirían las personas que utilicen los ambientes inteligentes basados en mashups?*

   Nuestro marco de trabajo utiliza principalmente a la Web como plataforma principal. Esto no quiere decir que necesariamente todas las aplicaciones creadas con el marco de trabajo necesiten seguir el modelo de uso de una aplicación Web. Los desarrolladores podrían crear, por ejemplo, aplicaciones de escritorio que se comporten de manera tradicional pero en los que las fuentes de datos y funcionalidad interna estén en la Web. En la Sección 5.3 se describe cómo podría ser el modelo de interacción de algunos escenarios de aplicación basados en nuestras contribuciones.

9. *¿Cómo podría permitirse la fácil extensión de las capacidades del marco de trabajo para permitir su aplicación y la inclusión de nuevas tecnologías?*

   El marco de trabajo es fácilmente extensible gracias a que cada elemento en la infraestructura tanto de *hardware* como de *software* están modelados como un servicio Web independiente en el que la funcionalidad está encapsulada como una "caja negra". Cualquiera de estos elementos podría intercambiarse para introducir nuevos dispositivos de *hardware* o nuevas tecnologías no consideradas anteriormente. Por ejemplo, las máquinas de ejecución actualmente utilizan un algoritmo sencillo para el procesamiento de datos siguiendo el lenguaje simple de composición propuesto como parte de nuestras contribuciones. Si un desarrollador así lo considera, podría incluir WS-BPEL para realizar la ejecución de composiciones. De igual manera, de existir una opción más apropiada para el descubrimiento de servicios sólo se tendrían que sustituir los servicios de directorio por otros que hagan uso de la nueva tecnología. Finalmente, cualquier sistema legado puede ser incluido en el marco

de trabajo al proveer un servicio REST que se encargue de la comunicación entre el marco de trabajo y el sistema legado.

6.4 Conclusiones

A lo largo de este capítulo se ha analizado la evaluación del presente trabajo de tesis siguiendo una estrategia de tres etapas. En la primera se llevó a cabo un experimento de usabilidad con el objetivo de evaluar la efectividad de nuestras contribuciones en el desarrollo de aplicaciones por parte de un grupo de usuarios potenciales de la misma, de identificar obstáculos de usabilidad que impidieran lograr los objetivos y de analizar el estado de nuestro trabajo aplicando el marco de trabajo de las dimensiones cognitivas. Como resultados de la primera fase no se encontraron problemáticas graves que indicaran algún obstáculo en la forma en la que está diseñada la notación (que es lo que se utiliza al trabajar para lograr un objetivo) ni se encontró que esta no fuera entendible por los desarrolladores. También se comprobó la factibilidad de nuestro marco de trabajo en el desarrollo de aplicaciones para ambientes inteligentes. Se encontraron algunos problemas menores pero todos enfocados a la interfaz gráfica del prototipo y no al marco de trabajo en sí. Durante la segunda fase se realizó una evaluación funcional en la que se verificó el soporte a cada una de los requerimientos de diseño del marco de trabajo. Finalmente, la tercera fase se concentró en la introducción de respuestas para las preguntas de investigación. El objetivo principal del trabajo de tesis era el siguiente:

Desarrollar un marco de trabajo, utilizando las tecnologías e ideas de la Web de las Cosas, para la creación de aplicaciones a partir de la adquisición, composición y agregación de las fuentes de datos, servicios y dispositivos disponibles en los ambientes inteligentes e Internet de manera efectiva, sencilla y fácil.

Creemeos que este trabajo cumplió el objetivo ya que efectivamente se proporcionó una forma para adquirir datos a partir de dispositivos físicos, de permitir la agregación de información y lógica de aplicación y de descubrir los elementos de un ambiente inteligente permitiendo al mismo tiempo la integración de servicios y aplicaciones disponibles en la Internet. Todo esto mediante el uso de interfaces de programación sencillas y estandarizadas que promueven la independencia de plataforma y lenguaje de programación.

Conclusiones

En esté capítulo final se hace un breve resumen de las contribuciones del presente trabajo de tesis, se discuten algunos retos abiertos de la Internet de las Cosas y se habla acerca del trabajo a futuro.

Contribuciones

En esta tesis se presentó, como contribución principal, un marco de trabajo para el desarrollo de aplicaciones basadas en la composición de servicios Web independientes ofreciendo funcionalidad básica que en conjunto conforman la funcionalidad necesaria por el escenario de una aplicación. Dichos servicios pueden estar implementados en cualquier lenguaje de programación y residir tanto en dispositivos móviles como en cualquier servidor Web tradicional. También pueden ser utilizados para permitir la incorporación de sistemas legados y cualquier tipo de servicio externo siempre y cuando exista una entidad en el marco de trabajo que se encargue de la traducción. Otra de nuestras contribuciones es la propuesta de un lenguaje de composición simple de alta granularidad que es utilizado para agregar la funcionalidad y los datos de una aplicación. Como parte de las contribuciones también se encuentra el mecanismo de notificación de eventos y la utilización del mismo en los servicios individuales. Finalmente, si bien nuestro marco de trabajo fue enfocado a los requerimientos y características especiales de los escenarios de inteligencia ambiental, este puede ser aplicado en otros dominios de aplicación o incluso en el desarrollo de aplicaciones tradicionales.

En resumen, las contribuciones de este trabajo son:

- Un marco de trabajo con una arquitectura basada en recursos para simplificar las tareas de composión de servicios y que incorpora notificaciones de eventos y la in-

tegración de entidades físicas en una aplicación (como proveedores de contexto y de actuación).

- Un modelo de interacción de los usuarios con los ambientes inteligentes basado en *mashups* que facilita la creación de aplicaciones y el consumo de funcionalidad.
- La posibilidad de desarrollar aplicaciones no sólo para cómputo ubicuo sino en cualquier dominio de aplicación que pueda ser descrito como una arquitectura basada en recursos.

Retos abiertos en la Internet de las Cosas

A lo largo de este documento, y sobre todo en los capítulos de experimentación y evaluación, se habla acerca de la conveniencia de la utilización de la Web como la plataforma en común para aplicaciones en el contexto de los ambientes inteligentes. En un principio la Web estaba diseñada para servir como el lugar común en donde compartir información de sólo lectura. Fue hace ya algunos años cuando empezó a tener contenido muy dinámico y generado por sus propios usuarios. Más recientemente, la Web empezó a ser vista como una plataforma de aplicaciones. Para poder utilizar efectivamente a la Web como una plataforma para aplicaciones en ambientes inteligentes se tienen que resolver dos problemáticas. La primera es el modelo de comunicación, en la Web, los clientes deben solicitar la página o recurso que necesitan. En cambio, en un ambiente inteligente son los proveedores de recursos (dispositivos tales como sensores y actuadores) quienes deben notificar a los clientes cuando haya cambios. Aún hay mucho trabajo adelante acerca de cómo mejorar el modelo de comunicación de la Web para permitir notificaciones que no sólo son efectivas, sino también para resolver aspectos tales como la latencia, escalabilidad, extensibilidad, entre otros. Es muy problable que la Web avance hacia un modelo de comunicación bidireccional. El segundo problema del uso de la Web en ambientes inteligentes es que estos son de naturaleza local, es decir, las transacciones y solicitudes de

servicio se efectúan en una red local cerrada. La necesidad básica de un ambiente inteligente en donde el usuario interactúa con el entorno sin conocer su naturaleza es la del descubrimiento automatizado de los servicios que están disponibles. Afortunadamente existen muchos protocolos y propuestas que pueden ser utilizados para ello.

Finalmente, la Web está basada en HTTP. Un protocolo basado en texto que en ciertas ocasiones resulta ser demasiado "pesado" como para considerar seriamente su uso. En la Internet de las Cosas convivirán desde dispositivos grandes y complejos hasta pequeños dispositivos con un mínimo de energía y capacidades de procesamiento limitadas. Si consideramos que muchos de los dispositivos pequeños necesitarán comunicarse con otros dispositivos también limitados el uso de un formato binario es más apropiado que HTTP. Es por ello que se han propuesto soluciones tales como la del protocolo CoAP (Shelby *et al.*, 2012) que está diseñado para ser una alternativa a HTTP que ofrece exactamente la misma funcionalidad pero que no está basado en texto sino en un formato binario. La ventaja de CoAP es que es fácilmente traducible en HTTP y viceversa lo que facilitaría la integración de dispositivos y redes de dispositivos pequeños y limitados.

Trabajo futuro

La idea del presente trabajo de tesis se originó al observar la variedad de herramientas de desarrollo, arquitecturas y tecnologías que la misma evolución de la Web estaba introduciendo. Herramientas tales como los editores de *mashups*, el poder crear aplicaciones Web al sólo arrastrar y soltar elementos visuales. Arquitecturas como REST en donde la funcionalidad se modela como recursos a los que se les pueden aplicar operaciones básicas. Tecnologías como las que están poco a poco permitiendo que todo dispositivo físico pueda ser aumentado y conectado a la Internet. Todo esto conformó la idea de poder permitir crear aplicaciones Web que pudieran dar soporte a escenarios en el contexto de la inteligencia ambiental. Para hacerlo, se debían resolver algunos problemas tales como la

integración de dispositivos físicos, la notificación de eventos, el descubrimiento de servicios en un ambiente, la composición de los mismos, el cómo identificar a cada servicio en función de lo que hacen, el cómo sería la interacción de los desarrolladores con dichas aplicaciones, entre otros. La solución de todos estos problemas conforman ahora un marco de trabajo que puede ser utilizado para apoyar investigaciones que se concentren en problemáticas más especificas.

Uno de los trabajos a futuro que se han identificado específicamente es la problemática de la privacidad. Una vez que la visión de la Internet de las Cosas nos provea de sensores y otros dispositivos que puedan permitir que aspectos tales como la actividad de los usuarios puedan ser accedidos a través de la Web, existirá cada vez más el problema de cómo proteger a las personas para que sus datos y actividades no sean utilizados más que para los fines que ellos mismos hayan permitido. Nuestro marco de trabajo puede ser extendido con características de privacidad al agregar a las descripciones de los servicios una especie de "contrato" especificado para cada usuario que indique qué se puede hacer con los datos y la funcionalidad de cada servicio. La capa de descubrimiento entonces podría filtrar servicios cuyos fines permitidos no coincidan con el propósito para el que se están buscando servicios. De esta manera las máquinas de ejecución compondrían funcionalidad y datos que garanticen el perfil de privacidad de cada persona. Por supuesto que hay muchos detalles que deberían ser identificados y resueltos.

Otro trabajo a futuro podría ser la incorporación de semántica a las descripciones y búsquedas de servicios. Algunos autores han mencionado que la próxima etapa de la Web podría ser una en la que la Web es semántica. Páginas y recursos en la Web tendrían una explicación acerca de qué es lo que significan y el contexto al que aplican que podría ser leída y entendida por *software* lo que ayudaría a que los sistemas ofrezcan mejores respuestas a los usuarios. La semántica también podría ser introducida en las representaciones mismas de los recursos lo que permitiría que las tareas de composición y agregación pudieran

ser más inteligentes. De igual manera esta es una problemática compleja pero que podría brindar muchos beneficios tanto a los desarrolladores de ambientes inteligentes, como a los usuarios de los mismos.

Para finalmente concluir, creemos que nuestras contribuciones ayudarán a promover aún más la visión de la Internet de las Cosas al facilitar la experimentación de escenarios de aplicación por desarrolladores que no necesitan ser expertos en el desarrollo Web ni que necesitan resolver problémáticas de bajo nivel tales como la adquisición de datos a partir de dispositivos físicos. Nuestro marco de trabajo estará disponible como código abierto en el sitio Web: `http://www.ubisoa.net/`

Referencias bibliográficas

Aarts, E., Harwig, H. y Schuurmans, M. (2002). *The invisible future: The seamless integration of technology into everyday life*. McGraw-Hill. ISBN 0-07-138224-0.

Abowd, G. y Mynatt, E. (2002). The human experience. *IEEE pervasive*, 1(1), 48–57.

Albinola, M., Baresi, L., Carcano, M. y Guinea, S. (2009). Mashlight: A lightweight mashup framework for everyone. *MEM '09: 2nd workshop on mashups, enterprise mashups and lightweight composition on the Web. Madrid, España, 20 de abril de 2009*.

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I. y Weerawarana, S. (2003). BPEL4WS specification: Business process execution language for Web services version 1.1. Recuperado de `http://public.dhe.ibm.com/software/dw/specs/ws-bpel/ws-bpel.pdf` el 6 de diciembre de 2012.

Android (2012). `http://www.android.com/`, consultado el 6 de diciembre de 2012.

Auto-ID Labs (2012). `http://www.autoidlabs.org/`, consultado el 6 de dic. de 2012.

Avilés-López, E. y García-Macías, J.A. (2009). TinySOA: A service-oriented architecture for wireless sensor networks. *Service oriented computing and applications*, 3(2), 99–108.

Avilés-López, E. y García-Macías, J.A. (2009). UbiSOA Dashboard: Integrating the physical and digital domains through mashups. *Human interface and the management of information. Designing information environments*, 5617(4), 466–474.

Avilés-López, E., García-Macías, J.A. y Villanueva, I. (2010). Developing ambient intelligence applications for the assisted living of the elderly. *ANT '10: International conference on ambient systems, networks and technologies. París, Francia, 8–10 de noviembre de 2010*.

Avilés-López, E., Villanueva-Miranda, I., García-Macías, J.A. y Palafox-Maestre, L.E. (2009). Taking care of our elders through augmented spaces. *CLIHC '09: Proceedings of the latin-american conf. on human-computer interaction. Mérida, Yucatán, 9–11 de nov. de 2009*.

Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D. y Patel-Schneider, P.F., editors (2003). *The description logic handbook: Theory, implementation, and applications*. Cambridge University Press. ISBN 0-521-78176-0.

Blackwell, A.F. y Green, T.R.G. (2000). A cognitive dimensions questionnaire optimised for users. *Proceedings of the twelth annual meeting of the psychology of programming interest group, Corigliano Calabro, Italia, 10–13 de abril de 2000*, 137–152.

Blau, B., Lamparter, S. y Haak, S. (2009). Remash! blueprints for RESTful situational Web applications. *WWW'09: Proc. of the 18th international conference on World Wide Web. Madrid, España, 20–24 de abril de 2009*.

Brewer, E.A. (2001). Lessons from giant-scale services. *IEEE Internet computing*, 5(4), 46–55.

Caswell, D. y Debaty, P. (2000). Creating Web representations for places. *HUC '00: Proceedings of the second international symposium on handheld and ubiquitous computing. Bristol, Inglaterra, 25–27 de septiembre de 2000*, 114–126.

Cervantes, H. y Hall, R. (2003). Automating service dependency management in a service-oriented component model. *CBSE '03: Proceedings of the 6th international workshop on component-based software engineering. Portland, Oregon, 3–4 de mayo de 2003*.

COSM Community (2012). Recuperado de http://community.cosm.com/?q=node/1 el 2 de diciembre de 2012.

Crossbow (2012). `http://www.xbow.com/`, consultado el 6 de diciembre de 2012.

Davies, N. y Gellersen, H. (2002). Beyond prototypes: Challenges in deploying ubiquitous systems. *IEEE pervasive computing*, 1(1), 26–35.

Davies, N., Wade, S., Friday, A. y Blair, G. (1997). Limbo: A tuple space based platform for adaptive mobile applications. *ICODP/ICDP'97: Proceedings of the international conference on open distributed processing/distributed platforms. Toronto, Canadá, 27–30 de mayo de 1997*, 291–302.

Digital Living Network Alliance (2012). `http://www.dlna.org/`, consultado el 30 de noviembre de 2012.

Endres, C., Butz, A. y MacWilliams, A. (2005). A survey of software infrastructures and frameworks for ubiquitous computing. *Mobile information systems journal*, 1(1), 41–80.

EPCglobal (2011). Tag data standard version 1.6. Recuperado de `http://www.gs1.org/gsmp/kc/epcglobal/tds` el 6 de diciembre de 2012.

EPCglobal (2012). `http://www.gs1.org/epcglobal`, consultado el 6 de dic. de 2012.

Fielding, R.T. (2000). *Architectural styles and the design of network-based software architectures*. Tesis de doctorado, University of California, Irvine.

Flickr (2012). `http://www.flickr.com/`, consultado el 6 de diciembre de 2012.

García-Macías, J.A., Alvarez-Lozano, J., Estrada, P. y Avilés-López, E. (2011). Browsing the internet of things with sentient visors. *Computer*, 44(5), 46–52.

Gershenfeld, N., Krikorian, R. y Cohen, D. (2004). The Internet of Things. *Scientific american*, 291(4), 76–81.

Google Maps (2012). `http://maps.google.com/`, consultado el 6 de diciembre de 2012.

Google Mashup Editor (2012). `http://editor.googlemashups.com/`, consultado el 6 de diciembre de 2012.

Google PowerMeter (2012). `http://google.com/powermeter/`, consultado el 6 de diciembre de 2012.

Green, T.R.G. (1989). Cognitive dimensions of notations. *People and Computers*, 443–460.

Green, T.R.G. (1991). Describing information artefacts with cognitive dimensions and structure maps. *HCI '91: People and computers VI, usability now. Edimburgo, Escocia, 20–23 de agosto de 1991*.

Green, T.R.G. y Patre, M. (1996). Usability analysis of visual programming environments: A cognitive dimensions framework. *Journal of visual languages and computing*, (7), 131–174.

Greenberg, S. (2009). Promoting creative design through toolkits. *CLIHC '09: Proceedings of the latin-american conference on human-computer interaction. Mérida, Yucatán, 9–11 de noviembre de 2009*.

Grimm, R., Davis, J., Lemar, E., MacBeth, A., Swanson, S., Gribble, S., Anderson, T., Bershad, B., Borriello, G. y Wetherall, D. (2001). System-level programming abstractions for ubiquitous computing. *UbiTools '01: Workshop on application models and programming tools for ubiquitous computing. Atlanta, Georgia, 30 de septiembre de 2001*.

Griswold, W.G., Boyer, R., Brown, S.W. y Truong, T.M. (2003). A component architecture for an extensible, highly integrated context-aware computing infrastructure. *ICSE '03: Proceedings of the international conference on software engineering. Portland, Oregon, 3–10 de mayo de 2003*, 363–372.

Guinard, D. y Trifa, V. (2009). Towards the Web of Things: Web mashups for embedded devices. *WWW '09: Proceedings of the 18th international conference on World Wide Web. Madrid, España, 20–24 de abril de 2009*.

Guinard, D., Weiss, M. y Graml, T. (2009). Energie visible: A sustainable Web of Things project. Recuperado de `http://www.webofthings.org/2009/02/11/energie-visible-a-sustainable-web-of-things-project` el 2 de diciembre de 2012.

Handley, M.J. (2009). Web application description language (WADL). W3C member submission, Sun Microsystems. Recuperado de http://www.w3.org/Submission/wadl/ el 6 de diciembre de 2012.

Haque Design + Research Ltd. (2008). Extended environments markup language (EEML). Recuperado de http://www.eeml.org/ el 2 de diciembre de 2012.

Hartmann, B., Doorley, S. y Klemmer, S.R. (2008). Hacking, mashing, gluing: Understanding opportunistic design. *Pervasive computing*, 7(3), 46–54.

Helal, S. (2005). Programming pervasive spaces. *IEEE pervasive computing*, 4(1), 84–87.

Herzog, G., Merten, H.K.S., Ndiaye, A., Poller, P. y Becker, T. (2003). MULTIPLATFORM testbed: An integration platform for multimodal dialog systems. *HLT-NAACL 2003 workshop: Software engineering and architecture of language technology systems (SEALTS). Edmonton, Canadá, 27–31 de mayo de 2003*, 75–82.

IANA (2012). Link relations. Recuperado de http://www.iana.org/assignments/link-relations/link-relations.xml el 2 de diciembre de 2012.

IETF Zeroconf Working Group (2012). Recuperado de http://www.zeroconf.org/ el 30 de noviembre de 2012.

iGoogle (2012). http://www.google.com/ig, consultado el 6 de diciembre de 2012.

iOS (2012). http://www.apple.com/ios, consultado el 6 de diciembre de 2012.

IPSO Alliance (2012). http://ipso-alliance.org/, consultado el 6 de dic. de 2012.

Kato, H. y Billinghurst, M. (1999). Marker traking and HMD calibration for a video-based augmented reality conferencing system. *IWAR '99: Proceedings of the 2nd international workshop on augmented reality. San Francisco, California, 20–12 de octubre de 1999*.

Kleinberger, T., Becker, M., Ras, E., Holzinger, A. y Müller, P. (2007). Ambient intelligence in assisted living: Enable elderly people to handle future interfaces. *Universal Access in Human-Computer Interaction. Ambient Interaction*, 103–112.

Krumm, J. y Horvitz, E. (2004). LOCADIO: Inferring motion and location from Wi-Fi signal strengths. *MobiQuitous '04: The first annual international conf. on mobile and ubiquitous systems, networking and systems. Boston, Massachusetts, 22–26 de agosto de 2004*, 4–13.

LabVIEW (2012). http://www.ni.com/labview/, consultado el 6 de diciembre de 2012.

Lagendijk, R. (2000). Ubiquitous communications (Ubicom) – updated technical annex 2000. *Tecnical report, ubiquitous communications program TU-Delf*.

López-de-Ipiña, D., Vazquez, J.I. y Abaitua, J. (2007). A Web 2.0 platform to enable context-aware mobile mash-ups. *AmI '07: Proceedings of the 2007 european conference on ambient intelligence. Darmstadt, Alemania, 7–10 de noviembre de 2007*, 266–286.

MacIntyre, B. y Feiner, S. (1996). Language-level support for exploratory programming of distributed virtual environments. *UIST '96: ACM symposium for user interface software and technology. Seattle, Washington, 6–8 de noviembre de 1996*, 83–95.

Mealling, M. y Denenberg, R. (2002). Report from the joint W3C/IETF URI planning interest group: Uniform resource identifiers (URIs), URLs, and uniform resource names (URNs): Clarifications and recommendations. Internet RFC 3305.

Microsoft Popfly (2012). `http://www.popfly.com/`, consultado el 6 de diciembre de 2012.

Naguib, H., Coulouris, G. y Hopper, A. (2001). Middleware support for context-aware multimedia applications. *DAIS '01: Distributed applications and interoperable systems. Krakow, Polonia, 17–19 de septiembre de 2001*.

Nehmer, J., Becker, M., Karshmer, A. y Lamm, R. (2006). Living assistance systems: An ambient intelligence approach. *ICSE '06: Proceedings of the 28th international conference on software engineering. Shanghai, China, 20–28 de mayo de 2006*, 43–50.

Nelson, T.H. (1965). Complex information processing: A file structure for the complex, the changing and the indeterminate. *ACM '65: Procedings of the 1965 20th national conference. Cleveland, Ohio, 24–26 de agosto de 1965*, 84–100.

Newman, J., Ingram, D. y Hopper, A. (2001). Augmented realidy in a wide area sentient environment. *ISAR '01: Proceedings of the international symposium on augmented reality. Nueva York, EE.UU., 29–30 de octubre de 2001*.

Nicklas, D., GroBmann, M., Schwarz, T., Volz, S. y Mitschang, B. (2001). A model-based, open architecture for mobile, spatially aware applications. *SSTD '01: Proceedings of the 7th international symposium on advances in spatial and temporal databases. Redondo Beach, California, 12–15 de julio de 2001*, 117–135.

Nielsen, J. y Landauer, T.K. (1993). A mathematical model of the finding of usability problems. *Proceedings of the INTERACT '93 and CHI '93 conference on human factors in computing systems. Amsterdam, Holanda, 24–29 de abril de 1993*, 206–213.

Nike (2012). Nike+ running application. Recuperado de `https://itunes.apple.com/mx/app/nike-gps/id387771637?mt=8` el 2 de diciembre de 2012.

Nike+ para correr (2012). `http://nikerunning.com/`, consultado el 6 de dic. de 2012.

OASIS (2012). OASIS Web services business process execution language (WSBPEL) TC. Recuperado de https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel el 2 de diciembre de 2012.

Olsen, D.R., Nielsen, S.T. y Parslow, D. (2001). Join and capture: A model for nomadic interaction. *UIST '01: Proceedings of the 14th annual ACM symposium on user interface software and technology. Orlando, Florida, 11–14 de noviembre de 2001*, 131–140.

OMA (2012). OMA EMML documentation. Recuperado de http://www.openmashup.org/omadocs/v1.0/ el 2 de diciembre de 2012.

OpenEnergyMonitor (2012). http://openenergymonitor.org/emon/, consultado el 6 de diciembre de 2012.

Oppermann, R., Rashev, R. y Kinshuk, K. (1997). Adaptability and adaptivity in learning systems. *Knowledge transfer*, 2, 173–179.

Pachube/Cosm (2012). http://www.pachube.com/, consultado el 6 de diciembre de 2012.

Palafox, L.E. y García-Macias, J.A. (2009). Wireless sensor networks for voice capture in ubiquitous home environments. *ISWPC '09: Proceedings of the 4th international conference on wireless pervasive computing. Melbourne, Australia, 11–13 de febrero de 2009*, 332–336.

Panoramio (2012). http://www.panoramio.com/, consultado el 6 de diciembre de 2012.

Pautasso, C. (2009). RESTful Web service composition with BPEL for REST. *Data and Knowledge Engineering*, 68(9), 851–866.

Phidgets (2012). http://www.phidgets.com/, consultado el 6 de diciembre de 2012.

Piekarski, W. y Thomas, B. (2003). An object-oriented software architecture for 3D mixed reality applications. *ISMAR '03: Proceedings of the international symposium on mixed and augmented reality. Tokyo, Japón, 7–10 de octubre de 2003*.

Ploggs (2012). http://plogginternational.com/, consultado el 6 de dic. de 2012.

PubSubHubbub (2012). http://code.google.com/p/pusubhubbub, consultado el 6 de diciembre de 2012.

Rellermeyer, J.S., Duller, M., Gilmer, K., Maragkos, D., Papageorgiou, D. y Alonso, G. (2008). The software fabric for the Internet of Things. *IOT '08: Proceedings of the 1st international conference on the Internet of Things. Zúrich, Suiza, 26–28 de marzo de 2008*, 87–104.

Riba, N. y Cervantes, H. (2007). A MDA tool for the development of service-oriented component-based applications. *ENC '07: Eight mexican international conference on computer science. Morelia, México. 24–28 de septiembre de 2007*.

Rodríguez, M. y Favela, J. (2003). A framework for supporting autonomous agents in ubiquitous computing environments. *UbiComp '03: Fifth annual conference on ubiquitous computing. Seattle, Washington, 12–15 de octubre de 2003*.

Roman, M., Hess, C.K., Cerqueira, R., Ranganathan, A., Campbell, R.H. y Nahrstedt, K. (2002). Gaia: A middleware infraestructure to enable active spaces. *IEEE pervasive computing*, 1(4), 74–83.

Routledge, P.A., O'Mahony, M.S. y Woodhouse, K.W. (2004). Adverse drug reactions in elderly patients. *British journal of clinical pharmacology*, 57(2), 121–126.

Ruan, C. y Yeo, S.S. (2009). Modeling of an intelligent e-consent system in a healthcare domain. *Journal of universal computer science*, 15(12), 2429–2444.

Schmalstieg, D., Fuhrmann, A., Hesina, G., Szalavari, Z., Encarnação, L.M., Gervautz, M. y Purgathofer, W. (2002). The studierstube augmented reality project. *Presence*, 11(1).

Schoenberger, C.R. (2002). The Internet of Things. *Forbes magazine*, 169(6), 155–160.

Schönfelder, R., Wolf, G., ReeBing, M., Krüger, R. y Brüderlin, B. (2002). A pragmatic approach to a VR/AR component integration framework for rapid system setup. *Proceedings of the paderborn workshop "augmented und virtual reality in der produktentstehung". Paderborn, Alemania, 11–12 de junio de 2002*.

Shelby, Z., Hartke, K., Bormann, C. y Frank, B. (2012). Constrained application protocol (CoAP). Reporte técnico: IETF Internet-draft. Consultado el 2 de diciembre de 2012, Recuperado de `https://datatracker.ietf.org/doc/draft-ietf-core-coap/`.

Ubiquitous Web Applications (2012). `http://www.w3.org/2007/uwa/`, consultado el 6 de diciembre de 2012.

UPnP Forum (2012). `http://upnp.org/`, consultado el 30 de noviembre de 2012.

Vasseur, J.P. y Dunkels, A. (2010). *Interconnecting smart objects with IP: The next Internet*. Morgan Kaufmann. ISBN 978-0-12-375165-2.

Vazquez, J. y López-de-Ipiña, D. (2008). Social devices: Autonomous artifacts that communicate on the Internet. *IOT '08: Proceedings of the 1st international conference on The Internet of Things. Zúrich, Suiza, 26–28 de marzo de 2008*, 308–324.

Want, R., Schilit, B., Adams, N., Gold, R., Petersen, K., Ellis, J., Goldberg, D. y Weiser, M. (1995). The PARCTAB ubiquitous computing experiments. *Technical report CSL-95-1, Xerox Palo Alto Research Center. Marzo de 1995*.

Weber, W., Rabaey, J.M. y Aarts, E.H.L., editors (2005). *Ambient intelligence*. Springer. ISBN 3540238670.

Weiser, M. (1993). Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36(7), 75–84.

Weiser, M. (1999). The computer for the 21st century. *SIGMOBILE mobile computing communications review*, 3(3), 3–11.

Weiser, M. y Brown, J.S. (1997). The coming age of calm technology. *Beyond calculation: The next fifty years of computing*, 75–85.

Wisneski, C., Ishii, H., Dahley, A., Gorbet, M.G., Brave, S., Ullmer, B. y Yarin, P. (1998). Ambient displays: Turning architectural space into an interface between people and digital information. *CoBuild '98: Proceedings of the first international workshop on cooperative buildings, integrating information, organization, and architecture. Darmstadt, Alemania, febrero de 1998*, 22–32.

Wohlgemuth, W. y Triebfürst, G. (2000). ARVIKA: Augmented reality for development, production and service. *DARE '00: Proceedings of DARE 2000 on designing augmented reality environments. Elsinore, Dinamarca, 12–14 de abril de 2000*, 151–152.

Wong, J. y Hong, J. (2006). Marmite: End-user programming for the Web. *CHI'06: Extended abstracts on human factors in computing systems*, 1541–1546.

World Wide Web Consortium (2012). `http://www.w3.org`, consultado el 6 de dic. de 2012.

Wright, A. (2009). Get smart. *Communications of the ACM*, 52(1), 15–16.

Yahoo! Pipes (2012). `http://pipes.yahoo.com/`, consultado el 6 de diciembre de 2012.

Yang, H., Jansen, E. y Helal, S. (2006). A comparison of two programming models for pervasive computing. *SAINT-W '06: Proceedings of the international symposium on applications on Internet workshops. Phoenix, Arizona, 23–27 de enero de 2006*, 134–137.

Zang, N., Rosson, M.B. y Nasser, V. (2008). Mashups: Who? what? why? *CHI '08: Extended abstracts on human factors in computing systems. Florencia, Italia, 5–10 de abril de 2008*, 3171–3176.

Zhang, L. y Mitton, N. (2011). Advanced Internet of Things. *IEEE iThings 2011. Dalian, China, 19–22 de octubre de 2011*. Dalian, China.

Zhong, N., Ma, J., Huang, R., Liu, J., Yao, Y., Zhang, Y. y Chen, J. (2010). Research challenges and perspectives on Wisdom Web of Things (W2T). *The journal of supercomputing*, 1–21.

Apéndice A

Materiales de evaluación

A continuación se incluyen los materiales utilizados durante el experimento de evaluación. El Apéndice A.1 son las instrucciones entregadas a cada participante antes de la realización de la prueba. El Apéndice A.2 es el cuestionario que se les entregó al completar las tareas de implementación. Para más detalles véase el Capítulo 6.

Apéndice A.1

Instrucciones

Las siguientes son algunas instrucciones para apoyarte en el desarrollo de la evaluación del sistema UbiSOA. Durante la evaluación se te solicitará desarrollar dos aplicaciones. Durante la primera, podrás conocer cómo utilizar la interacción de arrastrar, soltar y enlazar representaciones gráficas de los servicios para crear una nueva aplicación. La segunda aplicación requiere hacer una modificación a la primera para cambiar el escenario de aplicación a uno más complejo.

Servicios

Todos los servicios en UbiSOA son servicios HTTP que siguen el comportamiento de las arquitecturas REST. Es decir, toda la interacción con los servicios se basa en el intercambio y en la creación de recursos. Los recursos tienen todos su propio URI único que sirve para identificarlos en el sistema. A su vez, cada recurso tiene muchas representaciones. Por ejemplo, un objeto puede ser obtenido como un documento XML, una cadena JSON o una fuente de datos Atom. Para interactuar se utiliza: GET para leer, POST para crear, PUT para actualizar y DELETE para eliminar.

Hay un tipo especial de servicios, los servicios que envían notificaciones vía *push*. Con los servicios *push* no hay necesidad de enviar solicitudes GET recurrentes para conocer si uno de sus recursos fue actualizado. El servicio mismo enviará una solicitud POST a aquellos servicios que se hayan suscrito para recibir notificaciones. Durante la evaluación utilizarás 2 servicios *push*: el servicio de sensado y el de RFID.

Servicio de sensado

Éste servicio es un servicio *push* que provee los datos del último mensaje de sensado que haya recibido desde un sensor inalámbrico. Si se desea recibir notificaciones cuando se reciban nuevos mensajes de sensado, se debe enviar una solicitud de suscripción al servicio de sensado utilizando un servicio *hub*.

El servicio de sensado provee recursos como el siguiente:

```
{
    "timestamp":   "2011-07-09 13:12:53",
    "nodeId":      201,
    "humidity":    56.43033981323242,
    "voltage":     3.0829999446868896,
    "light":       138,
    "temperature": 28.770000457763672
}
```

El ejemplo anterior es una cadena JSON que se transforma en un objeto nativo según la plataforma en donde se interprete. Por ejemplo, en PHP, el objeto anterior se transforma en una variable `$obj` y para obtener el valor del parámetro *temperature* se debería utilizar `$obj -> temperature`.

Servicio de RFID

Éste servicio es un servicio *push* que provee un registro de todos los eventos de lectura de un lector de etiquetas RFID. El servicio provee recursos como el siguiente:

```
{
    "events": [
        {
            "id":        "1500584691",
            "timestamp": "2011-07-09 13:43:25",
            "action":    "gained"
        }, {
            "id":        "1500584691",
            "timestamp": "2011-07-09 13:43:26",
            "action":    "lost"
        }
    ]
}
```

El ejemplo anterior es una cadena JSON. En una variable PHP, `$obj -> events` contendría un arreglo de todos los objetos que representan a cada uno de los eventos de lectura. En particular `$obj -> events[1] -> action` contiene el valor de la acción detectada, la cual es `gained` si la tarjeta está dentro del alcance del lector o `lost` si la tarjeta salió del alcance del lector.

Servicio de LEDs

Éste servicio es un servicio REST común, es decir, no provee notificaciones *push*. El servicio provee acceso al estado de 4 LEDs de colores diferentes. Se puede conocer el estado de los LEDs o puede modificarse. Por ejemplo, la siguiente es una cadena JSON devuelta al consultar el estado actual de los LEDs.

```
{
    "red":    "off",
    "blue":   "off",
    "green":  "off",
    "yellow": "off"
}
```

En el ejemplo anterior los 4 LEDs están apagados, para cambiar su estado, el servicio recibe solicitudes POST con los datos codificados como un formulario HTML. Por ejemplo, para encender el LED azul y apagar los demás LEDs se enviaría una solicitud POST con el cuerpo siguiente:

```
blue=on&green=off&yellow=off&red=off
```

Servicio Servo

Éste servicio permite consultar o modificar la posición de un servomotor. El servicio devuelve lo siguiente:

```
{
    "minimum": "0.0",
    "maximum": "220.0",
    "current": "0.0"
}
```

Para modificar la posición del motor, se le envía lo siguiente:

```
position=200
```

El ejemplo anterior mueve el motor a la posición 200.

Servicio de Twitter

Éste servicio provee acceso al API de Twitter permitiendo el envío de un mensaje directo al usuario que se indique. Por ejemplo, para enviar "Hola" al usuario @eaviles, se le envía lo siguiente:

```
message=Hola&username=eaviles
```

Recetas

Las recetas en UbiSOA están escritas utilizando PHP. UbiSOA provee métodos y variables especiales para ayudar a que la creación de recetas sea más sencilla.

`$target`

Ésta variable contiene toda la información acerca de la instancia del servicio al que apunta la receta.

```
$target -> name;       // El nombre de la instancia.
$target -> host;       // El URL al host.
$target -> port;       // El puerto.
$target -> implements; // La interfaz que implementa.
$target -> data;       // El objeto donde se almacenan los datos de la instancia.
```

La propiedad más importante de ésta variable es `$target -> data`. Puesto que aquí es donde se almacenan todos los datos para la instancia del servicio en la aplicación. En el caso de los servicios *push*, cuando llega una notificación, ésta propiedad se actualiza con el nuevo recurso.

`$inbound`

Ésta variable es similar a la anterior pero en realidad se trata de un arreglo de objetos de instancias de servicio. Pero, de las instancias que sirven como entrada para una receta en particular. Por ejemplo, para obtener los datos de una de las entradas de la receta se utiliza:

```
$inbound[0] -> data;
```

`$data`

Ésta variable contiene los datos de la instancia actual en donde se ejecuta la receta. Los cambios que se realicen al contenido de la variable se almacenarán para que pueda utilizarse en el caso que la instancia actual sea la entrada de otras recetas.

`subscribe(`*`[instancia]`*`)`

Éste método envía una solicitud de subscripción a un servicio *push*.

`post(`*`[instancia], [datos]`*`)`

Éste método envía una solicitud POST al servicio indicado con los datos que se especifican.

`last(`*`[arreglo]`*`)`

Éste método regresa la última variable de un arreglo PHP. Útil cuando por ejemplo se quiere obtener el último evento de un servicio de RFID.

Apéndice A.2

Cuestionario de evaluación

El presente cuestionario recopila tus observaciones acerca de qué tan fácil es utilizar algún tipo de sistema notacional. Un «sistema notacional» es aquel sistema en el que a través de el uso y manipulación de «notaciones» se generan «productos» (encontrarás un ejemplo de sistema notacional en la siguiente sección). Éste cuestionario incluye una serie de preguntas que te motivan a pensar en los medios que necesitas para utilizar un sistema notacional en particular y en cualquier otra cosa que te ayude a hacer lo que necesitas.

Información básica

- ¿Cuál es el nombre del sistema que estás evaluando?

- ¿Qué tanto tiempo lo has utilizado?

- ¿Te consideras competente en su uso?

- ¿Has utilizado otros sistemas similares? (si es así, por favor nómbralos)

- ¿Qué tanta experiencia tienes en programación para la Web?

0	1	2	3	4	5	6	7	8	9

- ¿Qué tanta experiencia tienes en el uso del lenguaje PHP?

0	1	2	3	4	5	6	7	8	9

- ¿Qué tan familiarizado estás con las arquitecturas REST?

0	1	2	3	4	5	6	7	8	9

- ¿Qué tan familiarizado estás con el protocolo HTTP?

0	1	2	3	4	5	6	7	8	9

Definiciones

Es posible que tengas que pensar con cuidado antes de responder las preguntas en las siguientes secciones, por lo que incluimos algunas definiciones y un ejemplo para ayudarte:

Producto: El producto es la razón principal del por qué estás utilizando un sistema notacional – qué cosas pasan como un resultado final o qué cosas serán producidas como un resultado de utilizar el sistema notacional. Éste evento o cosas es lo que se conoce como producto. Cualquier producto que necesita una notación para describirlo usualmente tiene una estructura compleja.

Notación: La notación es cómo te comunicas con el sistema – uno provee información en algún formato especial para describir el resultado final de lo que quieres y la notación provee información que puedes leer. Las notaciones tienen una estructura que corresponde en alguna forma a la estructura del producto que describen. También tienen partes (componentes, aspectos, etc.) que corresponden en alguna forma a partes del producto. Las notaciones pueden incluir texto, imágenes, diagramas, tablas, símbolos especiales o varias combinaciones de éstos. Algunos sistemas incluyen múltiples notaciones. Éstas pueden ser muy similar unas a otras – por ejemplo, cuando se utiliza una máquina de escribir; el texto que produce son sólo letras y caracteres, mientras que la notación en las teclas que presionas te dicen exactamente cómo obtener el resultado que quieres. En otros casos, un sistema podría incluir notaciones que a los humanos nos es difícil producir o leer. Por ejemplo cuando se utiliza un teléfono la notación en los botones es una secuencia simple de dígitos, pero el sonido que se escucha al presionar cada uno de ellos no es tan fácil de interpretar (diferentes tonos por cada número, clics y tonos de llamada). Un teléfono con una pantalla provee otra notación adicional que es más fácil de entender.

Para resumir cuál es la intención del uso de éstos términos, considera el ejemplo de un procesador de texto. El producto de utilizar el procesador de texto es un papel impreso. La notación es la forma en la que las letras se ven en la pantalla – en procesadores de texto modernos es bastante similar a lo que se imprime, pero éste no es siempre el caso. Si uno quiere encontrar y reemplazar una palabra en particular se utiliza la función buscar y reemplazar, usualmente en una ventana más dentro del sistema que tiene su propia notación.

Partes del sistema

- ¿Para qué tarea o actividad se utiliza el sistema?

- ¿Cuál es el producto de utilizar éste sistema?

- ¿Cuál es la notación principal del sistema? Recuerda que la notación es la forma en la que te comunicas con el sistema, es lo que se manipula para lograr conformar un producto.

Al utilizar el sistema, qué proporción de tu tiempo (un porcentaje aproximado) inviertes cuando:

- Buscas información dentro de la notación: ( ) %
- Trasladando cantidades importantes de información de unas fuentes a otras: ( ) %
- Agregando cantidades pequeñas de inf. a una descripción previamente creada: ( ) %
- Reorganizando y reestructurando descripciones previamente creadas: ( ) %
- Jugando con nuevas ideas en la notación, sin saber qué es lo que va a pasar: ( ) %

Preguntas acerca de la notación principal

Visibilidad y yuxtaposicionalidad

- ¿En una escala del 0 al 9, qué tan fácil es observar o encontrar las diferentes partes de la notación mientras se está creando o modificando? ¿Por qué?

- ¿Qué tipo de cosas es más difícil de observar o encontrar?

- Si necesitas comparar o combinar diferentes partes, ¿puedes verlas todas al mismo tiempo? Si no es así, ¿por qué no lo es?

Viscosidad

- Cuando necesitas hacer cambios a un trabajo anterior, en una escala del 0 al 9, ¿qué tan fácil es hacer el cambio? ¿Por qué?

- ¿Hay cambios en particular que son más difíciles o especialmente difíciles de hacer? ¿Cuáles?

Difusidad

- La notación a) ¿te permite decir lo que quieres con razonable brevedad?, o b) ¿es tediosamente extensa? ¿Por qué?

- ¿Qué tipo de cosas toman más espacio para describirlas?

Operaciones mentales difíciles

- ¿Qué tipo de cosas requieren más esfuerzo mental en ésta notación?

- ¿Hay algunas cosas que parezcan especialmente complejas cuando trabajas con ellas? (por ejemplo, cuando combinas muchas cosas) ¿Cuáles son?

Propensión al error

- ¿Hay algunos tipos de errores que parezcan particularmente comunes o fáciles de hacer? ¿Cuáles son?

- ¿Te encuentras a menudo haciendo pequeños errores que te irritan o te hacen sentir tonto? ¿Cuáles son algunos ejemplos?

Cercanía del mapeo

- ¿Qué tan cercanamente relacionada es la notación al resultado que estás describiendo?
- ¿Por qué?

- ¿Qué partes parecen ser particularmente extrañas de hacer o describir algo?

Expresividad del rol

- Cuando se lee la notación, es fácil decir qué cosa hace cada parte en el objetivo final? ¿Por qué?

- ¿Hay algunas partes que sean particularmente difíciles de interpretar? ¿Cuáles?

- ¿Hay alguna parte que no conozcas qué significan o qué hacen pero que las incluyes sólo porque siempre se han incluido? ¿Cuáles son?

Dependencias ocultas

- Si la estructura del producto significa que algunas partes están relacionadas de cerca a otras partes y los cambios a una afectan a las otras, ¿las dependencias están visibles? ¿Qué tipo de dependencias están ocultas?

- Si el producto que se desea construir fuera complejo y requiriera una descripción particularmente larga, ¿podrían existir problemas para identificar quién depende de quién?

- ¿Las dependencias permanecen iguales o hay algunas acciones que causan que éstas se bloqueen? Si es así, ¿cuáles son?

Evaluación progresiva

- ¿Qué tan fácil es detenerse a la mitad de la creación de una notación y revisar el trabajado en el estado en que se quedó? ¿Puedes hacerlo siempre que lo desees? Si no es así, ¿por qué no?

- ¿Puedes decir qué tanto progreso has hecho, o revisar en qué etapa de tu trabajo estás? Si no es así, ¿por qué no?

- ¿Puedes intentar versiones parcialmente completadas de tu producto? Si no es así, ¿por qué no?

Provisionalidad

- ¿Es posible experimentar cuando estás jugando con ideas o cuando no estás seguro de cómo proceder? ¿Qué características de la notación te ayudan a hacerlo?

- ¿Qué tipo de cosas puedes hacer cuando no quieres ser muy preciso acerca del resultado exacto que estás intentando obtener?

Compromiso prematuro

- Cuando estás trabajando con la notación, ¿puedes hacer el trabajo en cualquier orden que desees o el sistema te obliga a pensar antes y hacer algunas decisiones antes?

- Si el sistema te obliga a pensar y hacer decisiones antes de actuar, ¿cuáles son? ¿Qué tipo de problemas puede causar en tu trabajo?

Consistencia

- ¿Cuáles son las partes de la notación que significan cosas similares? ¿Es la similitud clara desde el modo en que aparecen? Por favor da ejemplos.

- ¿Hay algunos lugares donde las cosas dicen ser similares pero la notación las hace diferentes? ¿Cuáles son?

Notación secundaria

- ¿El sistema te permite hacer notas o agregar comentarios?

- Si la notación estuviera impresa en un papel en el que pudieras escribir o rayar (es decir, si la interacción con el sistema no fuera con la computadora, sino en papel), ¿qué escribirías o dibujarías en él?

- ¿Has agregado marcas adicionales o colores o formatos para clarificar, enfatizar o repetir algo?

Administración de la abstracción

- ¿El sistema te ofrece alguna forma de definir nuevas facilidades o términos dentro de la notación para que así puedas extenderlo para describir nuevas cosas o para expresar tus ideas más claramente? ¿Cuáles son?

- El sistema te insiste en iniciar definiendo nuevos términos antes de que puedas hacer otra cosa? ¿Qué tipo de cosas?

Preguntas adicionales

- ¿Te encontraste utilizando la notación en formas que eran inusuales o en formas que el diseñador no consideraba? Si es así, ¿cuáles son algunos ejemplos?

- Después de completar éste cuestionario, ¿puedes pensar en formas obvias en las que el diseño del sistema pueda ser mejorado? ¿Cuáles son? ¿Podría ser mejorado para tus propios requerimientos?

Apéndice B

UbiSOA Editor walkthrough

La presente guía paso a paso en inglés ejemplifica el uso de un prototipo del editor UbiSOA. Incluye una breve descripción de las partes de la notación, ejemplos de recetas, y una explicación acerca de cómo es mantenido el flujo de datos de una aplicación.

UbiSOA Editor is a tool to help developers create applications that are based in the UbiSOA Framework. This article is a walkthrough of how to use the editor to first create and later extend a sample application. The latest UbiSOA Editor prototype is online at: `http://editor.ubisoa.net/`

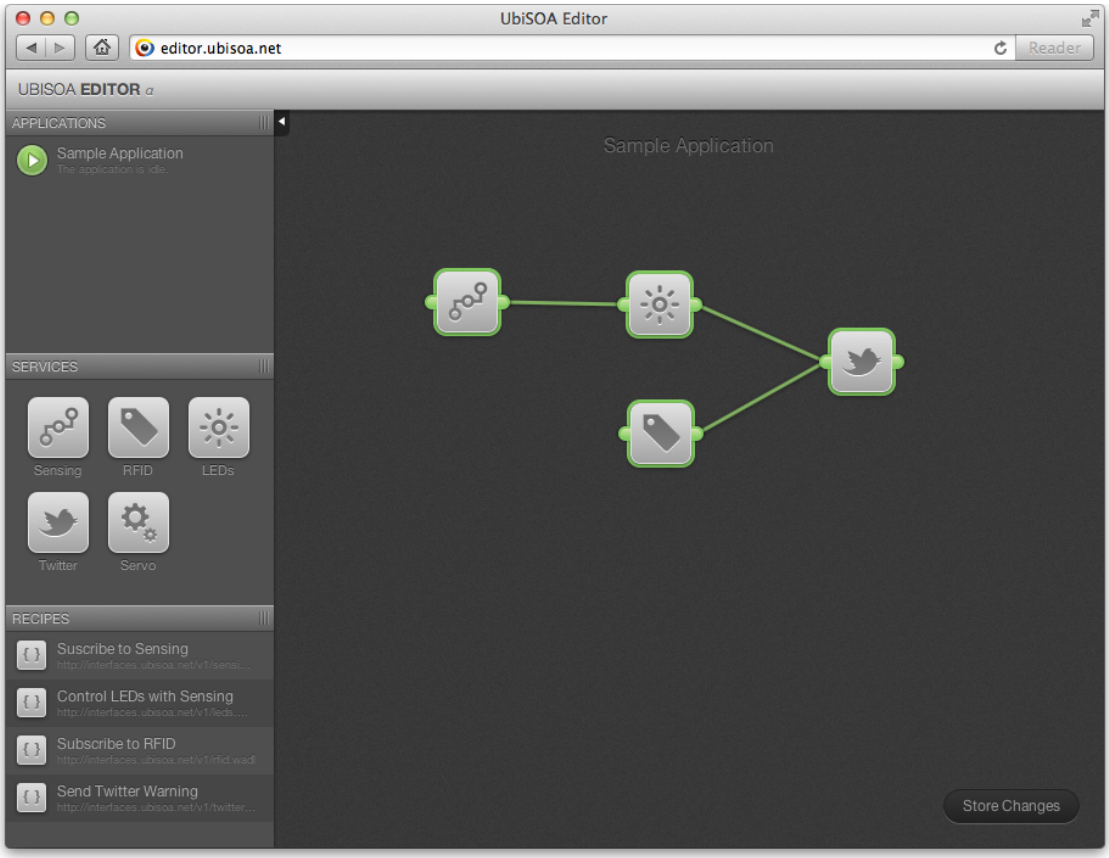
B.1 The user interface

The editor user interface is composed of 2 areas. The left area has 3 sections where the available applications, services, and recipes are listed. The work area is at the right, here is where services are dropped and linked together.

- **Applications**, this section lists all the available applications in the current environment whether they are running or not. At the moment, the prototype only allows to store and manage applications. They will not be able to run as the interaction between editor and execution engines would take place in local networks.
- **Services**, this sections lists all the discovered services in the current environment. The services listed in the prototype are not dynamically discovered but preconfigured.
- **Recipes**, this section lists the stored recipes that can be used in the editor. The work area is the drop place for services. Once services have been placed and connected,

click the Store Changes button to store the newly created manifest. To change the application name click the label on top of the work area.

The work area is the drop place for services. Once services have been placed and connected, click the Store Changes button to store the newly created manifest. To change the application name click the label on top of the work area.



B.2 The services

The following subsections detail each service that is currently supported by the editor prototype. Please note that the interaction with services will be handled by execution engines. The reason to include the descriptions here is to let you know how data is formatted as this information is needed when creating or editing recipes.

B.2.1 Sensing

Provides access to the sensing readings of a Wireless Sensor Network (WSN). The service can be pulled to retrieve the latest readings or clients can subscribe to it by using PubSubHubhub to receive readings as they arrive to the server. You can setup the server as following:

1. Connect a gateway mote to the computer where the sensing server will be deployed. The WSN nodes and the gateway must be programmed with the UbiSOA application for TinyOS.
2. Run the Collect.java application to start receiving packets and sending notifications.
3. Run the SensingServer.java application to start handling requests.

The following is a sensing reading sample in JSON format:

```
{
   ”timestamp”: ”2011-07-09 13:12:53”,
   ”nodeId”: 201,
   ”microphone”: 0,
   ”humidity”: 56.43033981323242,
   ”temperatureInternal”: 31.211763381958008,
   ”voltage”: 3.08299999446868896,
   ”lightVisible”: 44,
   ”light”: 138,
   ”temperature”: 28.770000457763672
}
```

B.2.2 RFID

Provides access to the tag events of an RFID reader. The service can be pulled to retrieve the latest tag events or clients can subscribe to it by using PubSubHubbub. To setup the server follow the next steps:

1. Be sure the Phidgets drivers are correctly installed in the computer where the server will be deployed.
2. Connect a PhidgetRFID reader to the same computer.
3. Run the RFIDServer.java application to start handling requests.

The following is a tag events sample in JSON format:

```
{
   ”events”: [{
         ”id”: ”1500584691”,
         ”timestamp”: ”2011-07-09 13:43:25”,
         ”action”: ”gained”
      }, {
         ”id”: ”1500584691”,
         ”timestamp”: ”2011-07-09 13:43:26”,
         ”action”: ”lost”
      }]
}
```

B.2.3 LEDs

Provides access to control the status of 4 LEDs of different color connected as outputs to a PhidgetInterfaceKit. The service can be pulled to retrieve or change the LEDs status. To setup the server follow the next steps:

1. Be sure the Phidgets drivers are correctly installed.
2. Connect 4 LEDs of different color as outputs of a PhidgetInterfaceKit. Use 0 for blue, 1 for green, 2 for yellow, and 3 for red.
3. Connect the PhidgetInterfaceKit to the same computer.
4. Run the SemaphoreServer.java application to start handling requests.

The following is an LEDs status sample in JSON format.

```
{
   ”red”: ”off”,
   ”blue”: ”off”,
   ”green”: ”off”,
   ”yellow”: ”off”
}
```

To change the LEDs status send a POST request with the content type `application/x-www-form-urlencoded` and the new status for each LED color. For example, to only turn the blue LED on send:

```
blue=on&green=off&yellow=off&red=off
```

B.2.4 Twitter

Provides access to publishing public replies in Twitter. The posted messages will appear in the @UbiSOA Twitter account. To setup the service just run the TwitterServer.java application to start handling requests. To publish a message send a POST request with `application/x-www-form-urlencoded` as content type, the target Twitter account for the public reply, and the message. For example:

```
message=Hello&username=eaviles
```

B.2.5 Servo

Provides access to control the position of a servo motor connected to a servo controller. The service can be pulled to retrieve or change the motor status. To setup the server:

1. Be sure the Phidgets drivers are correctly installed.
2. Connect a servo motor to a Phidgets servo controller.
3. Connect the servo controller to the same computer.
4. Run the ServoServer.java application to start handling requests.

The following is a motor status sample in JSON format.

```
{
   ”minimum”: ”0.0”,
   ”maximum”: ”220.0”,
   ”current”: ”0.0”
}
```

To change the motor position send a POST request with `application/x-www-form-urlencoded` as content type and the new motor position. For example, to move the motor to a new position:

```
position=100
```

B.3 The recipes

Once you start dropping and interconnecting services in the work area, the editor will try to find a recipe that matches the conditions of each data flow step. These conditions are, the interface the target service implements, and the interfaces that implement the services that provide the incoming data for a step. In the current prototype recipes are implemented as scripts of an extension of PHP. The reason to extend PHP is to provide functions and global variables that simplifies the writing of recipes.

B.3.1 Variables

`$target`

Contains all the information about the service that is currently targeted by the data flow step. Data is stored in a structure that we will refer as service structure throughout the rest of this article.

```
1 $target -> name;       // The instance name.
2 $target -> host;       // The host name or IP address.
3 $target -> port;       // The service port.
4 $target -> implements; // The implemented interface.
5 $target -> data;       // A structure with the current data for the instance.
```

`$inbound`

Contains an array of service structures, one for each of the incoming service instances. This is used to retrieve data from incoming connections. For example, `$inbound[0] -> data` contains the data from the first incoming connection, while `$inbound[0] -> name` contains its name.

B.3.2 Functions

`subscribe($instance)`

Sends a PubSubHubbub subscription to the given service structure in `$instance`.

`post($instance, $data)`

Sends a POST request to the given service structure in `$instance` containing the information of the `$data` array but converted as `application/x-www-form-urlencoded`. For example, the following array:

```
1 $data['blue'] = 'on';
2 $data['green'] = 'off';
3 $data['yellow'] = 'off';
4 $data['red'] = 'off';
```

Will be converted to `blue=on&green=off&yellow=off&red=off` before it is sent as a POST to the service described in `$instance`.

`last($array)`

Returns the latest structure of the specified array. Useful for when example getting the last event from a tag events array.

B.4 Sample application

Now that services and recipes have been introduced, lets implement the sample application. The objective of the application is to display the value of the latest sensed light reading using the 4 LEDs. For example, when the light is at 200 (out of 1000), only turn the blue LED on. If the light is at 400, only turn the blue and green LEDs on. If the light is at 1000, turn all the LEDs on.

To implement the application we first need to identify which services will be involved. In this case they are 2, the Sensing and the LEDs services. Drag the Sensing service from the Services section on the editor, and drop it in the work area. Now, as there are no recipes available, a gray border will appear around the service instance. The gray border means that this particular step in the data flow cannot be processed as no recipe was found for the step conditions. Lets create a recipe for it, right click on the recipe instance and select the **Create Recipe** option.

The Create Recipe dialog will appear. Write "Subscribe to Sensing" as the new recipe name, the **Target Type** and **Inbound Types** will already show the expected conditions for the new recipe. They currently match the conditions of the step from where the dialog was opened. Now, for the application we need to get the latest light reading, so we need to subscribe to the PubSubHubbub notifications of the sensing service. To do so, write as the Source Code for the new recipe the following:

```
1 subscribe($target);
```

After you write the recipe source code, click the **Store Changes** button. That's all, whenever the execution engine arrives to this step it will subscribe to the targeted sensing service (if the engine itself isn't already subscribed). Once that the dialog is closed, the border of the service instance in the work area will now be highlighted in green, this means that a recipe is available for the conditions of that particular step in the data flow. You can confirm this by right clicking on the service and selecting the View Properties option, a dialog will appear showing the service instance information. In this dialog you are also able to select between proper valid for the step. To close it click the **Store Changes** button (though, make sure you didn't changed anything).

Now drag and drop an instance of the LEDs service. To connect the Sensing service with the LEDs service drag the circle on the right of the sensing service to the left circle of the

LEDs service. Notice how the meaning of the border color also applies to the connection lines colors. Both the line and the border of the LEDs service will be gray as there is no recipe for the conditions. Right click on the LEDs service and again open the **Create Recipe** dialog.

Write "Control LEDs with Sensing" as the new recipe name. Notice how the **Target Type** and **Inbound Types** show that the recipe applies to a step in which the target service implements the LEDs interface and that has one inbound connection from a service that implements the sensing interface. As the **Source Code** write:

```
1 $value = $inbound[0] -> data -> light;
2
3 $data['blue'] = 'on';
4 $data['green'] = $value > 250? 'on': 'off';
5 $data['yellow'] = $value > 500? 'on': 'off';
6 $data['red'] = $value > 750? 'on': 'off';
7
8 post($target, $data);
```

Click the **Store Changes** button, after the dialog closes both services and the connection between them will be highlighted in green. What the new recipe does is first, it extracts the light reading from the sensing service and stores it in `$value`. Then, sets each LED color to either "on" or "off" based on the light threshold for each color. Finally, the recipe sends a POST request with the updated LED statuses to the LEDs service causing the lights to turn on or off depending on the light perceived by the sensors.

Click on the label at the top of the work area to set the new application name, it currently should read "Untitled Application", write "Sample Application". Finally, click the **Store Changes** button to store the application. Once stored, it will appear in the **Applications** section at the left of the editor, to run or stop the application click the green button.

B.5 Extending the application

Lets extend the previous application by adding new functionality and requirements. The updated objective of the application is to display the value of the latest sensed light reading using the 4 LEDs, and, if the red LED happens to be "on" it should send a warning message to the `@eaviles` Twitter account but only if there is a tag on top of the RFID reader.

To implement the application lets use the previously stored "Sample Application". You will find it in the **Applications** section at the left side of the editor. Right click on it and select **Edit Application**. We need 2 additional services to update the application: the RFID and Twitter services.

Drag and drop an instance of the RFID service. Right click on it and create a new recipe. Set "Subscribe to RFID" as the recipe name and use the following source code:

```
1 subscribe($target);
```

The source code is the same as before but the new recipe targets a service implementing the RFID interface and not the Sensing interface. Now drag and drop the Twitter service. Connect both the RFID and the LEDS to the Twitter service. Right click on the Twitter service and click **Create Recipe**. Use "Send Twitter Warning" as the new recipe name and the following source code:

```
1 $red_led = $inbound[0] -> data['red'];
2 $led_is_on = $red_led == 'on';
3
4 $tag_events = $inbound[1] -> data -> events;
5 $last_event = last($events);
6 $action = $last_event -> action;
7 $tag_is_present = $action == 'gained';
```

```
8
9  $data[”username”] = ’eaviles’;
10 $data[”message”] = ’This is a warning!’;
11
12 if ($red_is_on && $tag_is_present)
13    post($target, $data);
```

What the recipe does is first, gets the red LED status and checks if it's "on". Then, gets the last tag event and checks if it's "gained" it means that the tag is currently over the RFID reader. Finally, if both conditions are true a POST request is sent to the Twitter service to publish the warning. Both the sensing and RFID services send PubSubHubbub notifications, as soon as a new reading is available of any of them it will trigger the execution engine to go along all the subsequent data flow steps in the application, running recipes on each step. This completes the implementation of the updated application.

By using the editor you don't need to manually write, publish, and start the execution of the application manifest, the editor handles all the needed service requests by itself. Also, once recipes have been created, they are later available for creating new applications. If the recipes that are required for an application are already in place, users will only have to drag and drop services and interconnect them to implement functionality.