Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

PLATAFORMA DE COMPOSICIÓN, PROVISIÓN Y CONSUMO DE SERVICIOS PARA USUARIOS EN MOVILIDAD

TRABAJO FIN DE MÁSTER

Ramón Pablo Alcarria Garrido

Junio 2010

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

Máster Universitario en
Ingeniería de Redes y Servicios Telemáticos

TRABAJO FIN DE MÁSTER

PLATAFORMA DE COMPOSICIÓN, PROVISIÓN Y CONSUMO DE SERVICIOS PARA EL USUARIO PROSUMER EN MOVILIDAD

Autor
Ramón Pablo Alcarria Garrido

Director
Tomás Robles Valladares

Departamento de Ingeniería de Sistemas Telemáticos

Junio 2010

Resumen

La sociedad actual necesita de nuevas soluciones en el ámbito de la creación y provisión de servicios. Se demanda un nuevo modelo de relación entre individuos, empresas y ciudades en el que el usuario pueda aportar sus sugerencias, información de interés, e incluso sus propios servicios al resto de la comunidad, además de consumir al instante la información que precisa: por ejemplo el estado del tráfico, junto con recomendaciones sobre cuál es la ruta y medio de transporte más adecuado para el recorrido que hacemos habitualmente en ese día y hora. De tener la posibilidad de conseguir anticipadamente lo que vamos a necesitar para nuestro recorrido: peajes, aparcamiento, tickets, etc.

El objetivo de este Trabajo Fin de Máster es sentar las bases para la creación de una nueva generación de servicios en movilidad, orientados a usuarios y que pueden ser compuestos y prestados por los mismos, a través de su terminal móvil, extendido con las capacidades del entorno, tanto de interfaz como de presentación de la información.

El concepto de Universo Inteligente otorga un mayor sentido a las relaciones entre el usuario, su terminal y el entorno que lo rodea. Utilizando el terminal móvil, los usuarios crean servicios, contribuyendo al Universo Inteligente y permitiendo a los demás usuarios consumirlos.

En este trabajo se propone el diseño y desarrollo de una arquitectura funcional para una plataforma de creación y composición de servicios para este nuevo universo Inteligente, en el que el usuario, que actúa como proveedor y consumidor de servicios, se encuentra en movilidad. La plataforma propuesta se encargará de gestionar el ciclo de vida de los servicios: Creación, Publicación, Búsqueda y descubrimiento, Ejecución, Resolución de capacidades e Interoperabilidad.

Abstract

The current society needs new solutions in the service creation and provision fields. A new model in which individuals, companies and cities are related is needed. In this new model the user can share suggestions, interesting information, and even his services with the rest of the community. Also, he can instantly consume the information that he may need: For example the traffic status, with recommendations about which is the best path to follow and the mean of transport that is optimal in a given time, or having the possibility to get in advance what we will need during our trip: toll tickets, parking, etc.

The aim of this Master Thesis is to settle the basis for the creation of a new generation of mobile, user-oriented services, that can be composed and lent by the users themselves, using their mobile terminal and extended with some environment capabilities.

The concept of Intelligent Universe focuses in the relationship among the user, his terminal and the environment that surrounds him. Using the mobile phone, users can create services, contributing to the Intelligent Universe and allowing other users to consume them.

In this work we propose the design and implementation of a functional architecture for a service creation and composition platform for this new Intelligent Universe, in which the user, which acts as service producer and consumer, is on the move. The proposed platform with deal with the service lifecycle management: Creation, Publication, Search & Discovery, Execution, Capability Resolution and Interoperability.

Índice General

Índice de figuras

Índice de tablas

Glosario de Términos y Acrónimos

3GPP	3rd Generation Partnership Project
AAA	Authentication, Authorization and Accounting
API	Application programming interface
B2B	Business-to-Business
BPEL	Business Process Execution Language
BPM	Business Process Model
BPML	Business Process Modeling Language
DAML	DARPA Agent Markup Language
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
IMS	IP Multimedia Subsystem
IP	Internet Protocol
IPTV	IP Television
IPV4/IPV6	IP version 4/ IP version 6
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Micro Edition
JS	JavaScript
JSON	JavaScript Object Notation
JSR	Java Specification Request
LWUIT	Lightweight UI Toolkit

MIDP	Mobile Information Device Profile
NGN	Next Generation Networking
OSGI	Open Services Gateway Initiative
OWL	Ontology Web Language
OWL-S	Service OWL
P2P	Peer-to-Peer
PC	Personal Computer
PDA	Personal Digital Asistant
QOS	Quality of Service
RDF	Resource Description Framework
REST	Representational State Transfer
RFID	Radio Frequency IDentification
RTP	Real Time Protocol
SAML	Security Assertion Markup Language
SAWSDL	Semantic Annotations for WSDL
SDL	Service Description Language
SDP	Service Discovery Protocol
SIP	Session Initiation Protocol
SLA	Service Level Agreement
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
TAG	Etiqueta
TCP	Transmission-Control-Protocol
UA	User Agent
UDDI	Universal Description, Discovery and Integration

UDP	User Datagram Protocol
UI	User Interface
UML	Unified Modeling Language
UPNP	Universal Plug and Play
URIS	Uniform Resource Identifiers
URL	Uniform Resource Locator
VOD	Video On Demand
VOIP	Voice over Internet Protocol
W3C	World Wide Web Consortium
WSMO	Web Service Modeling Ontology
XHTML	eXtensible Hypertext Markup Language

1 Introducción

Un nuevo modelo de provisión de servicios es necesario en una sociedad donde los individuos, las empresas y las ciudades están relacionados, y en la que los usuarios contribuyen con recomendaciones, información de interés e incluso sus propios servicios al resto de la comunidad.

Para resolver los requisitos de evolución de la sociedad actual existen algunas líneas que se deben seguir[1]:

La creatividad en productos, servicios e información digital será mejorada gracias a la evolución de las tecnologías que permiten a los usuarios crear y prestar sus servicios de forma inmediata, utilizando sus terminales móviles extendidos con capacidades del entorno.

El estudio de tecnologías de la interfaz, que permiten desarrollar servicios que combinan la información real con la virtual. También la utilización de sistemas inteligentes para producir servicios digitales permitirá a los usuarios interactuar con productos tangibles.

Es necesario desarrollar nuevas tecnologías que permitan a las personas elegir libremente y de forma fácil qué servicios quieren y cuáles no les son necesarios. Además, la posibilidad de generar servicios por usuarios y la existencia de elementos de red que rodeen al usuario y a su terminal móvil contribuyen al paradigma Mobile 2.0[2] y a la evolución de una sociedad de masas a una sociedad de red.

Con respecto a envejecimiento de la población, las nuevas interfaces y servicios proporcionarán una mejor calidad de vida para las personas mayores al existir microservicios que puedan, por ejemplo, monitorizar un paciente por un familiar o un trabajador social.

Este trabajo analiza este entorno tecnológico enunciando el paradigma de Universo Inteligente, que sitúa al usuario en el centro del entorno. Este usuario, utilizando su terminal móvil, podrá aprovecharse de las capacidades que ofrece el entorno y de las proporcionadas por entidades externas. El usuario, en movilidad, interactuará con elementos cercanos, próximos y remotos, que le proporcionarán funcionalidades que podrá utilizar para componer sus propios servicios.

---

[1] Basado en el informe de la ISTAG [1] para la comisión Europea en Marzo 2006.

[2] Adaptación del paradigma Web 2.0 aplicado al entorno móvil.

1.1 El nuevo Universo Inteligente

Esta sección introduce el concepto de Universo Inteligente, como la evolución lógica de la relación de los usuarios con sus terminales móviles y con los servicios que les rodean. Esta nueva visión tecnológica significa un cambio en la utilización de los servicios móviles en términos de funcionalidad y utilidad. El usuario final será capaz de crear y proporcionar servicios inteligentes al resto de usuarios.

La plataforma de composición, provisión y consumo de servicios que se ha diseñado permite a los usuarios prestar servicios ubicuos en un entorno inteligente. Las tecnologías que hacen esto posible tiene que adaptarse a cada individuo y a su contexto, mientras que el terminal móvil se utiliza como base de interacción con los servicios proporcionados por empresas y otros usuarios en movilidad.

El concepto de universo inteligente asume que la persona es el elemento principal del entorno. Se necesita proporcionar un mecanismo fácil e intuitivo para establecer comunicaciones con el entorno y con las redes personales y corporativas. Para que esto sea posible, se necesitan nuevas interfaces de acceso que faciliten la vida diaria de los usuarios, permitiendo máxima simplicidad de interacción entre las personas y la información contextual presente en el entorno en el que éstas se mueven.

El universo inteligente permitiría a las personas el acceder de forma instantánea a la información que precisan, información sobre la compra de productos, niveles de congestión de tráfico, recomendaciones sobre la ruta óptima a seguir, los mejores productos a consumir, dependiendo de preferencias seleccionadas previamente por el usuario o inferidas por el sistema, utilizando para ello la información contextual. Estos servicios pueden incluso permitir la reserva de una plaza de parking o el pago de tasas y productos. Las opiniones de los usuarios también son tenidas en cuenta a través de servicios de recomendación para eventos, restaurantes y a través de la publicación de comentarios sobre películas, lugares turísticos, etc.

Hoy en día, las personas son conscientes de la necesidad de estar en contacto permanente, sin restricciones. Está concienciación ha sido causada por la revolución que ha supuesto el teléfono móvil, no sólo en el campo de las telecomunicaciones, sino también como herramienta de interacción entre los individuos y su entorno. Por tanto, el teléfono móvil se ha convertido en un elemento esencial para la vida de las personas: Se ha transformado en un objeto cotidiano, como el reloj y las gafas desde hace siglos.

La convergencia entre la necesidad de estar en contacto y el concepto de terminal móvil como objeto cotidiano genera la idea de móvil como dispositivo que proporciona a sus usuarios una conexión constante con otras personas (voz, datos, videollamadas) y

con el vasto catálogo de servicios disponibles (información, entretenimiento, relaciones sociales, servicios públicos, ver Figura 1).

Figura 1. Tecnologías y aplicaciones móviles

Esta tendencia señala un futuro en el que los ciudadanos participen activamente en este entorno digital en continua evolución. La información relevante integrada en el contexto y la existencia de personas más colaborativas que no sólo consumen servicios sino que también los componen nos llevan al nuevo escenario del ciudadano digital que utiliza las capacidades del entorno. De la misma forma que unas gafas correctoras magnifican las capacidades del individuo permitiéndole observar su entorno más cercano de forma más correcta un dispositivo móvil convertirá a su poseedor en un individuo con capacidades magnificadas digitalmente.

De esta forma, se necesita sacar partido a las funcionalidades de un dispositivo que está siempre cerca del usuario y siempre conectado, para que se convierta en una ventana abierta a la información relevante y personalizada que el usuario requiere en cada momento. La plataforma de provisión de servicios diseñada en este trabajo supone una parte fundamental de este terminal.

Esta revolución de la sociedad y de las tecnologías constituye el paradigma de Mobile 2.0 (ver Figura 2), que consolida las bases para la creación de una nueva generación de servicios en movilidad.

Figura 2. Nube de tags para Mobile 2.0

1.2 Usuario *prosumer*, nuevas posibilidades

Analizando la evolución del mercado móvil y comparándolo con las necesidades detectadas en la sociedad es posible predecir que el futuro nos proveerá no sólo de nuevas funcionalidades para los dispositivos sino también de soluciones innovadoras que permitan obtener el máximo provecho de las futuras capacidades. Los usuarios serán capaces de utilizar el móvil para participar, ofreciendo sus opiniones, creando sus propias aplicaciones e incluso prestando sus aplicaciones como servicios desde sus terminales, con garantías de privacidad y seguridad.

Los usuarios *prosumer* son los principales actores de este futuro universo inteligente. El término *prosumer* [2] es un acrónimo formado por la fusión de los términos ingleses *producer* (productor) y *consumer* (consumidor). Actualmente este término se aplica a aquellos usuarios que son al mismo tiempo consumidores y productores de servicios o contenidos. Un *prosumer* no tiene propósitos lucrativos, únicamente participa en el mundo digital del intercambio de información (por ejemplo los sistemas P2P). La palabra *prosumer* describe perfectamente a millones de usuarios de la Web 2.0, ya que cada vez están más y más involucrados en contribuir con información a la red mientras, al mismo tiempo, son consumidores de la información que existe en ella. Este término también tiene similitudes con el modelo EMEREC de Jean Cloutier [3], que asume que los nuevos medios de comunicación permiten que los usuarios sean a la vez emisores y receptores de mensajes.

Otorgar la posibilidad a los usuarios de que se transformen en diseñadores de sus propios servicios es un objetivo que ha sido perseguido desde múltiples iniciativas. Este cambio de paradigma en el que los usuarios se transforman en proveedores de servicio, denominado Nomadic Mobile Service Provisioning [4], tiene una gran relevancia, ya que es un paso más hacia la realización práctica de muchos paradigmas de computación que están siendo investigados actualmente, como la computación ubicua, la inteligencia ambiental o los sistemas sensibles al contexto, entre otros.

En el entorno integral del usuario prosumer en movilidad se persiguen los siguientes objetivos:

- Creación móvil de servicios móviles: explorar las tecnologías que permitan que el usuario sea capaz de crear sus propios servicios móviles usando el terminal móvil para ello. Investigar en tecnologías de semántica de servicios [5] [6], descripciones formales de servicios, soluciones gráficas de creación, creación mediante lenguaje natural, etc., con el objetivo último de permitir que el usuario cree el servicio que necesita de una manera usable, fácil y rápida.
- Prestación de servicios desde terminal móvil: permitir que los servicios que creen los usuarios se puedan prestar desde el propio terminal móvil, explorando las posibilidades (y limitaciones) del móvil como servidor y las tecnologías de ejecución de servicios móviles. Asimismo, establecer los mecanismos por los cuales servicios obtenidos de otros usuarios, de terceros o de otros terminales del propio usuario puedan ejecutarse en el propio terminal y obtener la información que requieran para su prestación de una manera estandarizada.
- Interoperabilidad de servicios creados por distintos usuarios: en un entorno de servicios creados por usuarios el número de posibles servicios es inmenso, pero el de prescriptores de un servicio concreto puede ser muy pequeño; por ello, se explorarán tecnologías que permitan resolver el problema de que servicios creados por distintos usuarios puedan compartir información e interoperar. El entorno integral de servicios debe asimismo permitir que los servicios creados por los usuarios puedan interoperar a su vez con servicios prestados por terceros, empresas e instituciones [7].
- Descubrimiento, recomendación y búsqueda de servicios móviles relevantes: el ecosistema de servicios creados por usuarios favorece el aumento exponencial de las clases de servicio existentes. El carácter móvil de la creación y prestación de servicios hace que la inmediatez y la relevancia de los servicios (su adaptación al momento, lugar y contexto del usuario) sean factores claves que se deben tener en cuenta en los mecanismos de búsqueda (proactiva por parte del usuario), recomendación y descubrimiento (pasivos) de servicio. Se examinarán tecnologías de búsqueda descubrimiento semántico y sensible al contexto, que

maximicen la relevancia de los servicios encontrados sin perjuicio de su inmediatez.

- Combinación transparente de múltiples fuentes de información: explorar tecnologías que permitan al usuario combinar diferentes fuentes de información en sus servicios sin exponerle las particularidades de cada fuente: el contexto del usuario, información proveniente de otros servicios u otros usuarios del ecosistema, información del operador, de empresas, web, etc.
- Gestión de la heterogeneidad de los terminales: definición de un modelado y armonización de capacidades (o componentes de servicio) que gestione de manera transparente para el usuario móvil la configuración específica de su terminal: qué capacidades posee (y están activas) y cuáles no, qué capacidades residen en el propio terminal y cuáles en terminales adyacentes o en la red del operador, qué servicios pueden prestarse y crearse en función de las capacidades disponibles.

2 Objetivos, métodos y fases

2.1 Objetivos

El objetivo principal de este trabajo es **ofrecer una visión de los nuevos paradigmas de creación, provisión y consumo de servicios para la creación de una nueva generación de servicios en movilidad, orientados a usuarios y que pueden ser compuestos y prestados por los mismos, a través de su terminal móvil y utilizando para ello redes convergentes**. Así mismo, se pretende establecer una arquitectura de referencia para una plataforma de despliegue de servicios mediante el diseño y el desarrollo de los mecanismos necesarios para gestionar el ciclo de vida de un servicio, desde su creación hasta su ejecución, describiendo los procesos necesarios para que el servicio se relacione con las capacidades del entorno.

Este objetivo global se traduce en los siguientes objetivos operativos:

- Realizar un estudio del arte de tecnologías a alto nivel para definir el marco tecnológico de la actividad global. Se describen los dominios tecnológicos de trabajo y las herramientas y tecnologías que pueden aplicarse a los ámbitos descritos.

- Proponer y analizar escenarios de uso. Para entender mejor la interacción de los usuarios con los sistemas propuestos se elaboran unos escenarios de uso genéricos y se identifican unos requisitos según las necesidades detectas o definidas en la propuesta.

- Extraer los casos de uso aplicables al entorno. Tras el análisis de los escenarios propuestos se identifican las actividades comunes en esos escenarios y se comparan con los elementos que participan en el ciclo de ida de un servicio.

- Recoger requisitos, condicionantes y restricciones para el entorno *prosumer* en movilidad. Este objetivo se elaborará en dos fases. En la primera se obtendrán los requisitos de usuario en movilidad, piezas clave determinan las necesidades en el diseño e implementación de la plataforma. En la segunda fase se analizarán los requisitos impuestos por el proceso de creación sobre el lenguaje de descripción de servicios.

- Definir funcionalmente y formalmente un servicio y el resto de conceptos clave del entorno. Existen distintas formas de definir y representar un servicio dependiendo del estado en el que se encuentre, los actores que lo rodean y la utilización que hagan de éste. Por tanto será necesario dedicar una sección a definir la estructura lógica de un servicio.

- Realizar una propuesta de entorno que satisfaga los requisitos, condicionantes y restricciones citados con anterioridad. Para ello se volverá a tener en cuenta el paradigma de Universo Inteligente, que determina la finalidad y los tipos de

servicios que se van a analizar y las restricciones sobre la plataforma de provisión de servicios.

2.2 Métodos y fases

El presente trabajo de investigación se enmarca en el conjunto de las actividades de investigación del grupo y en los trabajos específicos del alumno, relativos a la realización de su tesis doctoral que describirá contribuciones para la composición de servicios en movilidad para el usuario *prosumer*. Prueba de ello es el conjunto de contribuciones relacionadas con esta temática en las que el alumno ha participado y que se describen en la sección A. Publicaciones Relacionadas.

El procedimiento que se ha adoptado para la realización de este trabajo ha sido realizar un camino de recolección de requisitos de entorno partiendo de escenarios de aplicación del concepto de Universo Inteligente ya estudiado, pasando por las definiciones de elementos que intervienen en el entorno, una revisión tecnológica y la relación con otras actividades del proyecto, para poder finalmente realizar una propuesta inicial de entorno que sirva de referencia para proseguir las tareas de investigación en distintas áreas.

El punto de partida de este trabajo es la definición **del concepto de Universo Inteligente** y de lo que se entiende por **usuario *prosumer***. Esta tarea ya ha sido descrita en la sección de **Introducción**.

La segunda fase será identificar qué **ámbitos tecnológicos** pueden ser aplicables y condicionar al entorno *prosumer* en movilidad (**Sección 3**). Por lo tanto, el alumno realizará una prospección para identificar las tecnologías, que se desarrollarán en profundidad en cada tarea específica. La utilidad de este estudio será clasificar y definir el ámbito tecnológico de la actividad del trabajo, así como detectar las restricciones iniciales que la tecnología pueda imponer al entorno.

Una vez analizadas las tecnologías existentes se describen los escenarios relativos al comportamiento del usuario en el futuro universo inteligente. Se iniciará un proceso para la **definición de escenarios propios**, que recojan la visión del entorno *prosumer*. La identificación de los elementos comunes de los escenarios permitirá extraer **los casos de uso troncales** que definirán qué es lo que un usuario espera del entorno *prosumer*. Después de este proceso se podrán encapsular los **requisitos** que se esperan para el diseño de una plataforma como la que se pretende desarrollar en este trabajo. El trabajo realizado en esta fase se refleja en los **Apartados 4.1 y 4.2**.

La siguiente fase que se considerará en este trabajo es la **conceptualización del servicio de tipo *prosumer* y su definición (Apartado 4.3).** Encontrar una terminología adecuada y estudiar todos los aspectos derivados de un servicio como resultado de las situaciones expresadas en los distintos escenarios será la función más importante de este apartado.

Una vez se hayan definido escenarios, plasmado los conceptos más importantes de los servicios creados por y para los usuarios el alumno propone el diseño de una **arquitectura genérica para la provisión de servicios (Sección 5).** Además, en esta sección se describe el desarrollo de un activo experimental que simula un entorno de ejecución de servicios basados en componentes, utilizando la tecnología OSGi, orientado a los proyectos CARDEA [10] y CARDINEA [11].

Posteriormente, en la sección **6 Desarrollo de una arquitectura avanzada para la composición, provisión y consumo de servicios en movilidad** se describe el modelo de arquitectura orientado al proyecto mIO! [13], que estudia, define y desarrolla tecnologías para prestar servicios en movilidad en el futuro universo inteligente. A modo de validación, el alumno, con la ayuda del grupo de investigación en el que está integrado, ha realizado un prototipo limitado del entorno de creación para la creación de servicios por y para usuarios *prosumer* en movilidad.

En la **Sección 7 Identificación del lenguaje de descripción de servicios** se define el documento que contiene la especificación de los elementos que intervienen en el entorno: Servicio y Componente. Se ha trabajado en la definición de un lenguaje de descripción de servicios para satisfacer las condiciones del entorno. Concretamente se exponen los lenguajes de descripción de servicios para las vistas de componente y de servicio en notación BNF [14].

Por último, en la sección **8 Conclusiones y trabajos futuros** se exponen las conclusiones de este trabajo mediante el análisis de las conclusiones parciales de cada fase y se describen las contribuciones de este trabajo, indicando la línea de investigación a la que pertenecen y los trabajos futuros que se planean sobre cada línea.

3 Estado del arte de servicios en movilidad

Una vez establecidos los conceptos de Universo Inteligente y usuario *prosumer* se puede identificar que el entorno tecnológico integral aplicado al entorno *prosumer* vendrá definido por la integración de cuatro dominios tecnológicos diferentes para cubrir un propósito definido que es la definición del entorno integral. A continuación se describen estos cuatro dominios: el dominio de red, el de plataforma móvil, el de servicio y el semántico.

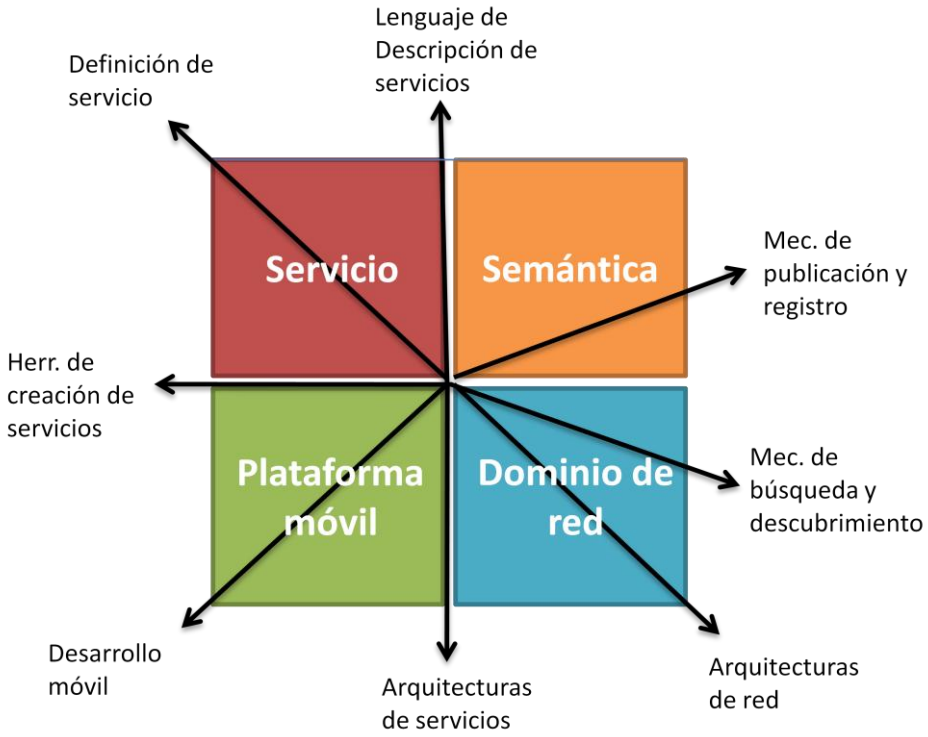

Figura 3. Dominios tecnológicos de trabajo

Como hemos observado en la figura anterior, cada dominio tecnológico está atravesado por una o varias fechas, que describen las tecnologías y herramientas que pueden aplicarse a los ámbitos descritos.

En el dominio del servicio debe explorarse el concepto de definición del servicio, que será estudiado en el apartado 4.3 Definición de servicio en el contexto del nuevo Universo Inteligente.

Las tecnologías que se estudian en este apartado son: Lenguajes de descripción de servicios, arquitecturas de servicios, arquitecturas de red, desarrollo móvil, mecanismos de búsqueda y descubrimiento de servicios, mecanismo de publicación y registro de servicios y por último, herramientas de creación de servicios.

3.1 Lenguajes de descripción de servicios

3.1.1 Introducción

Dentro de las plataformas orientadas a servicios, como es el caso de la plataforma objeto de estudio, es muy importante la información que permite la descripción de los diferentes servicios disponibles dentro de dicha plataforma. Se llama lenguaje de descripción de servicios al lenguaje que nos permite la descripción de los diferentes aspectos importantes a la hora de describir un servicio, como por ejemplo, la funcionalidad que nos aporta. Esta descripción no se queda simplemente en la mera descripción de qué hace un servicio (capacidades funcionales), sino que puede incluir la descripción de más aspectos del servicio, como son aspectos de seguridad, autoría, etc. (capacidades no funcionales), e incluso la interacción con otros servicios.

Otro aspecto relevante de esta plataforma es que se trata de una plataforma de servicios en dispositivos móviles, lo que hace que sea necesario que el lenguaje de descripción de servicios elegido sea lo suficientemente ligero como para poder ser utilizado en este tipo de dispositivos. Se busca la utilización de un lenguaje que nos permita definir no solo los servicios, sino también las capacidades y componentes.

Dentro de los lenguajes de descripción de servicios vamos a distinguir tres tipos:

- Los que describen semánticamente o definen el servicio (OWL-S, WSDL, WSMO, etc).
- Los que describen el modelado de servicios (BPMN)
- Los orientados a ejecución de servicios (BPEL, SCA, etc)

3.1.2 OWL-S

La iniciativa OWL-S (*A Semantic Markup for Web Services*) es una propuesta de varias instituciones a nivel mundial en la que se define una tecnología para la descripción semántica de servicios Web y para la automatización de una serie de tareas, como son el descubrimiento de servicios en la Web, la invocación automática de un servicio y la composición de los mismos para satisfacer las necesidades de los usuarios.

Actualmente la especificación de OWL-S está completada y se ha remitido al W3C como una *member submission* en Noviembre de 2004.

OWL-S es una ontología para la descripción de servicios que se articula en descripción de Perfil (*Profile*), *Grounding* y Modelo (*Model*) [15]. No tiene estatus de estándar, aunque ha sido remitida al W3C para su consideración.

En la actualidad, la última versión es la 1.1, en la que se incorpora la capacidad de utilizar reglas SWRL [16] para extender la expresividad de OWL en la descripción semántica de las capacidades y del interfaz de un servicio Web. Esta nueva versión permite además mejorar la definición del flujo de trabajo entre cliente y servicio Web (la coreografía de un servicio).

El funcionamiento de esta tecnología consiste en una interacción entre un agente (aplicación, otro servicio Web, una persona, etc.) que es capaz de descubrir un servicio Web a través de su perfil público (owls:ServiceProfile). A partir de aquí, el agente dispone de la información del proceso de negocio proporcionado por ese servicio Web de manera que puede invocarlo mediante la información provista en el *grounding* (owls:ServiceGrounding).

Conclusión

Desde el punto de vista técnico, OWL-S es conceptualmente una alternativa viable y el enfoque adoptado es sencillo y pragmático facilitando su implantación. También cuenta con un buen número de herramientas en un estado más o menos estable y con las cuales se puede pensar en utilizar la tecnología sin correr demasiados riesgos técnicos. No obstante, desde la perspectiva B2B quedan algunos puntos y responsabilidades sin asignar y de las cuales no se puede prescindir como pueden ser las transacciones o la seguridad. Otros aspectos como la escalabilidad de la tecnología y la intervención manual para gestionar los servicios Web cuestionan la viabilidad de OWL-S en un entorno de ejecución real.

3.1.3 WSDL

WSDL [17] es un lenguaje basado en XML [18] que permite describir los principales aspectos de un servicio Web, incluyendo sus operaciones, gramática de los mensajes recibidos y producidos por el servicio, y puntos de acceso del servicio (*endpoints*). Esta tecnología es el complemento de SOAP, el protocolo de intercambio de mensajes XML entre distintos procesos. Si bien no es necesario utilizar WSDL para lograr la comunicación efectiva con un servicio Web, la disponibilidad de una descripción WSDL del servicio Web (habitualmente generada y publicada por la entidad que oferta el servicio) facilita enormemente el desarrollo rápido de los artefactos necesarios para consumir el servicio.

A partir de una descripción WSDL, resulta posible generar automáticamente los *stubs* del servicio en el lado del cliente (en el caso de una invocación estática), o la composición e interpretación de mensajes al vuelo (en el caso de una invocación dinámica). La sintaxis de los mensajes XML intercambiados con el servicio Web se define mediante una gramática XML Schema.

Por otra parte, WSDL ofrece sólo una descripción básica del servicio, que deja fuera cuestiones muy importantes desde la perspectiva empresarial, como la calidad de servicio (QoS), la seguridad, la coreografía, etc. Para paliar esta situación, ha aparecido toda una serie de especificaciones conocidas como "WS-* stack"[3]. Finalmente, entre los aspectos que forman parte de una descripción WSDL, no se incluye la semántica. Diversas propuestas como SAWSDL [19], WSDL-S o OWL-S han surgido para enriquecer las descripciones de los servicios Web con semántica que permita la interpretación automática de la funcionalidad ofrecida por el servicio y los mensajes que consume y produce.

Aunque WSDL 2.0 [20] aporta un buen número de ventajas sobre la versión anterior, entre las que se incluyen una mayor estabilidad de la especificación o soporte para servicios Web REST, su despegue no está siendo rápido. Debido a la gran cantidad de descripciones WSDL 1.1 actualmente existentes, las herramientas de desarrollo de servicios Web están adoptando WSDL 2.0 con mucha cautela. Mientras tanto, la pujanza de los servicios Web basados en el paradigma REST ha restado protagonismo a los servicios descritos con WSDL 1.1.

Conclusión

En el ámbito de este trabajo resulta evidente que la construcción de un ecosistema dinámico de servicios distribuidos exige algún mecanismo efectivo para documentar el interfaz de estos servicios, de manera que sea posible simplificar la invocación y combinación de los servicios. No obstante, no está claro que WSDL sea la tecnología apropiada en este contexto, dado las peculiaridades de los entornos móviles y, sobre todo, las diferencias entre los servicios web clásicos y los servicios del universo inteligente.

3.1.4 BPMN

BPMN [21] es una representación gráfica que describe procesos de negocio mediante flujos de trabajo (*workflows)*. Es un estándar para el modelado de procesos de negocio y proporciona una notación gráfica para especificar procesos de negocio en un diagrama de procesos de negocio (BPD), basada en una técnica de seguimiento de flujo muy similar a los diagramas de actividad del lenguaje UML. El objetivo de BPMN es permitir la gestión de procesos para usuarios tanto técnicos como empresariales proporcionando una notación que es intuitiva para los usuarios corporativos y también es capaz de representar procesos semánticos complejos. La especificación BPMN

[3] WS-Security, WS-Policy, WS-Privacy, WS-Trust, etc.

también proporciona la capacidad de traducir entre la notación gráfica y los lenguajes de ejecución que irían por debajo, particularmente BPEL.

Desarrollo del estudio

En BPMN existen cuatro tipos de categorías:

Los **objetos de flujo** son los elementos principales para describir acciones en BPMN y se dividen en tres elementos:

Eventos: Representado con un círculo describe algo que ocurre. Existen dos tipos de eventos, los que esperan recibir un mensaje para arrancar un proceso y los que generan un mensaje cuando finaliza. Existen eventos de arranque (que activan el proceso), de parada (representan el final del proceso) e intermedios (que representan algo que ocurre durante la ejecución del proceso).

Tabla 1. Tipos de evento en BPMN

Evento de arranque	Evento de parada	Evento intermedio
		

Actividades: Representada como un rectángulo con esquinas redondeadas describe qué tipo de trabajo debe realizarse.

Figura 4. Actividad en BPMN

Puertas de enlace: Representadas con forma de rombo indican los distintos caminos en los que se divide o por los que se integra un determinado flujo dependiendo de las condiciones especificadas.

Figura 5. Puerta de enlace en BPMN

Conexiones: Elementos que conectan objetos de flujo y que se dividen en tres tipos:

Flujo de secuencia: Se representa como una flecha sólida e indica el orden en que deben ejecutarse las actividades.

Flujo de mensaje: Mediante una flecha de puntos indica los mensajes que se intercambian los distintos participantes.

Flujo de asociación: Mediante una línea de puntos representa los enlaces entre objetos de flujo y los artefactos, que se estudiarán más adelante.

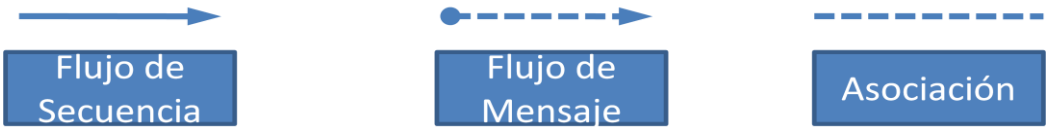

Figura 6. Conexiones en BPMN

Swimlanes: Su nombre viene de las pistas que se habilitan en una piscina para cada nadador y es una práctica utilizada en diagramas de flujos donde se representa el diagrama con distintas pistas en las cuales se asigna un participante en el proceso.

Artefactos: Permiten a los desarrolladores introducir más información dentro del modelo o diagrama. De esta forma el diagrama se hace más fácil de entender.

Existe la posibilidad de enlazar los modelos creados por esta notación con otros lenguajes más enfocados a la ejecución de eventos para poder llevar a la práctica los diseños de servicios creados en BPMN. Un objetivo clave para el desarrollo de BPMN es crear un puente entre la notación de modelado de procesos de negocios y los lenguajes de ejecución para los procesos que hay dentro de un sistema. Los objetos gráficos de BPMN, más un buen número de atributos de estos objetos, se han mapeado al Business Process Execution Language para Web Services [22] (BPEL4WS v1.1), el estándar de facto para la ejecución de procesos.

Implementaciones

Las implementaciones de esta notación más extendidas son el plugin para Eclipse BPMN Modeler [23] y la herramienta Intalio [24], que se analizará en el apartado 4.1.5 **¡Error! No se encuentra el origen de la referencia.**.

Eclipse BPMN Modeler

Esta implementación de la notación BPMN constituye un editor de diagramas de procesos de negocio para analistas, integrado en el entorno de desarrollo Eclipse.

El primer BPMN Modeler fue desarrollado por Intalio en 2006. En 2008 fue desarrollado como un componente de la plataforma de herramientas SOA (SOA Tools Platform) pero evolucionó a subproyecto en 2008.

Existe diversa documentación [25] acerca de esta herramienta que la hace fácil de utilizar. Además resulta sencillo extender el modelo definiendo nuevas propiedades, anotando código WSDL y produciendo código ejecutable para BPEL u otros lenguajes de workflow.

Conclusiones

Tras el análisis de esta tecnología desde el punto de vista de su utilización para el diseño de servicios se han extraído las siguientes conclusiones:

- No se ha encontrado ninguna herramienta para móviles que genere archivos BPMN o que permita construir de forma gráfica un diagrama de flujo por lo **que es poco probable que pueda utilizarse esta notación para contribuir al lenguaje de descripción de servicios que necesitamos definir.** Asimismo, la ausencia en este lenguaje de características de ejecución **le impiden formar parte del SDL para la vista de ejecución del servicio**.

- A partir de las dos conclusiones anteriores se determina que la utilidad de este lenguaje irá en la línea de **proporcionar a los desarrolladores del SDL información importante acerca del diseño de servicios**. Esta información se ha extraído del análisis y se expresa a continuación:

3.1.5 BPEL

Introducción

BPEL (Business Process Execution Language) [26], en español "Lenguaje para la ejecución de procesos de negocio" es un lenguaje basado en XML el cual ha sido diseñado para permitir la composición de tareas mediante computación distribuida o un entorno de granja de servidores. BPEL nace como evolución de las especificaciones WSFL [27] (lenguaje de flujo de Servicios Web) de IBM y XLANG [28] de Microsoft.

BPEL es un lenguaje de orquestación, no de coreografía [29], aunque en las versiones actuales está soportada. La principal diferencia entre orquestación y coreografía es el alcance. Podemos definir el alcance de un elemento como el ámbito de actuación de ese elemento sobre los demás. En el caso de la coreografía se ofrece un alcance específico a cada componente centrado en la vista de los mismos. Un modelo de orquestación, sin embargo, centra su alcance en el orquestador y abarca a todos los componentes del sistema y sus interacciones asociadas, dando una vista global. La distinción entre orquestación y coreografía está basada en analogías: la orquestación

describe el comportamiento del control central como un director de orquesta, mientras que en la coreografía el comportamiento del control se distribuye entre cada participante individualmente basándose estos en eventos exteriores, como cuando en una coreografía cada bailarín reacciona al comportamiento de su pareja.

Además de proporcionar facilidades para permitir enviar y recibir mensajes, las ayudas del lenguaje de programación de BPEL también proporcionan:

- Un mecanismo de correlación de mensajes basado en propiedades.
- Variables codificadas en XML y WSDL
- Un modelo de lenguaje extensible basado en plug-ins que permite escribir expresiones y peticiones en múltiples lenguajes: BPEL soporta Xpath 1.0 por defecto [30].
- Construcciones típicas de la programación estructurada como sentencias de if-else-then, while, sequence (para ejecutar comandos en orden) y flujo (para permitir la ejecución de comandos en paralelo).
- Un sistema de campos para permitir el encapsulado de lógica con variables locales, manejo de eventos, fallos, etc.
- Campos serializados para controlar el acceso concurrente a las variables.

Utilización de BPEL para este proyecto

BPEL se propone como lenguaje de descripción de servicios ofrecidos por la plataforma de estudio. En realidad se debe considerar a BPEL como una especificación estándar para la descripción de la ejecución de procesos de negocio. La necesidad de definir un lenguaje de descripción de servicios que aporte una vista de ejecución al servicio, de modo que pueda ser interpretada por un motor en tiempo de ejecución podría estar resuelta si se determina que BPEL es un lenguaje adecuado para este fin. Además, no debe considerarse el lenguaje de manera aislada, porque las implementaciones de BPEL son en realidad motores que implementan la especificación BPEL (por ejemplo la BPEL4WS2.0) y que sirven como intérpretes del lenguaje permitiendo ejecutar los modelos diseñados (con extensión .bpel). Además este tipo de implementaciones integran herramientas para transformar los modelos escritos en BPEL a otros lenguajes ejecutables mediante máquinas virtuales como Java. En el apartado del desarrollo del estudio se comprobará si BPEL cumple con los requisitos exigidos para ser utilizado en el sistema o si, por el contrario, se debe buscar otra solución que se adapte más a las necesidades del proyecto.

Relación entre BPEL y BPMN

BPEL, tal y como se ha estudiado, no posee una notación gráfica estándar ya que el Comité Técnico de OASIS [31] consideró que no entraba en el alcance de la especificación. Se ha propuesto que el lenguaje BPMN se utilice como herramienta gráfica para representar descripciones de procesos BPEL. Como ejemplo de utilización de esta alternativa, la especificación BPMN incluye una conversión informal y parcial de BPMN a BPEL. Una conversión más detallada puede ser llevada a cabo por herramientas especializadas, algunas de ellas gratuitas como BPMN2BPEL y otras de cariz corporativo como Intalio, que se estudiará en este mismo apartado.

De todos modos, el desarrollo de estas herramientas ha mostrado diferencias fundamentales entre BPMN y BPEL. En algunos casos estas diferencias son insalvables, y no permiten generar código BPEL legible para los modelos. Un problema mayor surge a la hora de mantener los modelos actualizados, ya que cualquier cambio que se haga en uno de ellos será propagado al otro.

A continuación tenemos un segmento de un proceso de negocio que marca el mapeo con BPEL4WS.

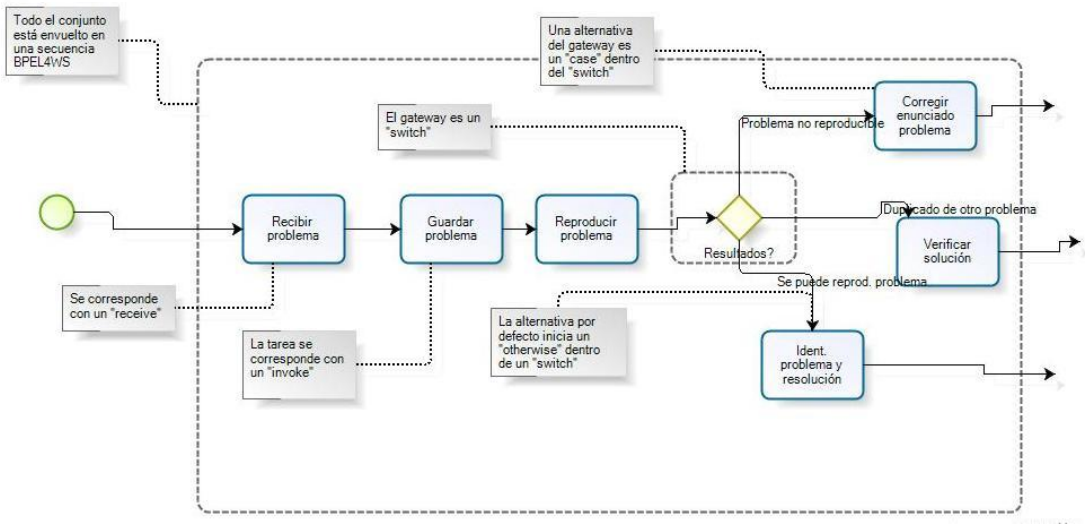

Figura 7. Conversión BPMN a BPEL

Implementaciones de BPEL

ActiveBPEL

Motor *open-source (GPL)* que implementa la especificación **BPEL4WS 1.1** y **BPEL4WS2.0 estándar**. La empresa **Active Endpoints**, que comercializa versiones "avanzadas" del mismo, se encarga de su mantenimiento actualmente. Es una interesante iniciativa para acercar los mundos de los procesos de negocio y los servicios web al Open Source.

ActiveVOS

Herramienta comercial creada por Active Endpoints que proporciona un conjunto de herramientas para la orquestación de servicios y el modelado de procesos de negocio. Una de las características más destacadas es la **posibilidad de crear modelos BPMN que pueden ser convertidos directamente a BPEL**.

Intalio BPM

Solución propuesta por la compañía Intalio. Plataforma de código abierto construida sobre la herramienta de modelado BPMN de Eclipse (Eclipse STP BPMN modeler) y sobre el motor BPEL de Apache (Apache ODE BPEL Engine). Proporciona todos los componentes requeridos para el diseño, despliegue y administración de los procesos de negocio más complejos

La herramienta Intalio se compone básicamente de un Editor gráfico (basado en la plataforma Eclipse) y de un motor de ejecución (de BPEL 2.0, basado en J2EE).

Sliver

Sliver es un motor de ejecución de workflows escritos en BPEL que soporta una gran variedad de dispositivos desde los teléfonos móviles hasta los sistemas de escritorio [32] [33]. Tiene una versión para J2Se y otra para J2ME, que soporta dispositivos que tienen librerías de MIDP v2.0 o posterior.

Sliver pone de manifiesto la problemática de ejecutar motores de BPEL en entornos móviles, ya que existen tanto limitaciones hardware (alto consumo de memoria y procesador de los motores BPEL convencionales) como limitaciones software (implementaciones diseñadas para J2SE y no para J2ME/MIDP).

Sliver intenta resolver estos problemas, proporcionando una clara separación entre las comunicaciones y el procesado. Además, Sliver sólo depende de las librerías externas, las cuales están diseñadas para entornos móviles. En la siguiente figura se puede observar la arquitectura de Sliver:

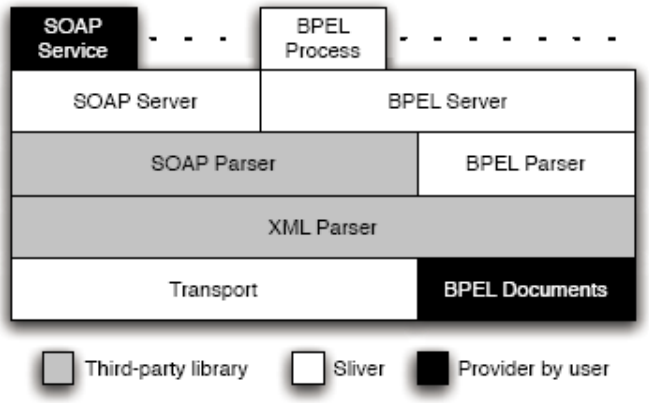

Figura 8. Arquitectura de Sliver

En el nivel más bajo la capa de transporte envuelve varios medios y protocolos de transporte proporcionando una interfaz al nivel superior. La capa de transporte intercambia mensajes en XML que se encargan de descodificar las capas de XML Parser y SOAP Parser. La capa de BPEL Parser se ayuda de la XML Parser para convertir documentos BPEL proporcionados por el usuario en procesos ejecutables concretos.

Conclusiones

Ventajas de la utilización de BPEL

BPEL se ha propuesto como una alternativa muy aconsejable para formar parte del lenguaje de descripción de servicios que quiere definirse en este trabajo. Las ventajas de su utilización se enumeran a continuación:

- Lenguaje maduro y estándar. Sintaxis clara y especificaciones abiertas. Utilizado comúnmente para el diseño de procesos de negocio.
- Posee una implementación del motor para móviles, Sliver, estudiada en el apartado anterior.
- Podría ser válido para la vista interpretada del SDL de un servicio ya que pemite la invocación de servicios. Para el caso de estudio estos servicios serán en realidad componentes, y se invocarán en tiempo de ejecución.

Inconvenientes de la utilización de BPEL

- No es útil como lenguaje de descripción de servicios para la vista de descripción ya que no está diseñado específicamente para desplegar gráficamente un servicio. En este asunto la notación BPMN es más completa.
- La invocación a servicios se realiza mediante comandos SOAP por lo que, de alguna forma el protocolo SOAP deberá estar presente en las comunicaciones entre componentes. Aunque el acceso mediante REST es posible no está muy desarrollado y puede traer complicaciones.

- Aunque se ha estudiado una implementación de BPEL para móviles, sólo consiste en el motor y no en ningún editor ni generador de código BPEL por lo que su aplicabilidad es limitada. Además, Sliver es una implementación muy poco conocida, no está muy madura y el soporte de la comunidad es mínimo.

Como conclusión final, debido a las ventajas e inconvenientes expuestos, BPEL se tendrá en consideración a la hora del diseño de un lenguaje de descripción de la vista interpretable de un servicio y más concretamente su implementación de motor para móviles, Sliver. Sin embargo, no se utilizará toda la funcionalidad que ofrece BPEL sino que se ofrecerá un subconjunto de elementos de su sintaxis que se corresponda con los requisitos impuestos por los servicios a crear desde el entorno de creación.

3.1.6 SCA

SCA (Service Component architecture) [34] es una nueva iniciativa que propone una arquitectura de servicios más adecuada a los principios de la arquitectura orientada a servicios. Está basada en la idea de que una función de negocio se compone de un conjunto de servicios, que están ensamblados unos con otros para crear soluciones que satisfacen una necesidad de negocio concreta. SCA es un modelo basado en SDO (Service Data Objects) el cual proporciona una API única que unifica la programación de datos de distintas fuentes para acceder a fuentes de datos heterogéneas y convierte a las aplicaciones, herramientas y frameworks en elementos más fácilmente accesibles, legibles, actualizables y comprobables.

El modelo de ensamblado de SCA [35] consiste en un conjunto de artefactos que definen la configuración de un dominio SCA desde el punto de vista de las composiciones, las cuales contienen los componentes de servicios ensamblados, las conexiones y los artefactos relacionados, que describen cómo están enlazados.

El artefacto básico de SCA es el **componente**, que es la unidad básica de construcción. Un componente consiste en una instancia configurada de una implementación, donde se define como implementación el trozo de código que proporciona una funcionalidad. Esta funcionalidad es ofrecida por otros componentes como un **servicio**. Las implementaciones pueden depender de servicios proporcionados por otros componentes, estas dependencias son llamadas en este lenguaje **referencias**. Las implementaciones tienen **propiedades** configurables, que son datos que influyen en las operaciones. El componente **configura** la implementación proporcionando valores a las propiedades y resolviendo las referencias a los servicios proporcionados por otros componentes.

Desarrollo del estudio

Los elementos que describen el contenido y los enlaces de una aplicación son llamados "*composites*". Los *composites* pueden contener componentes, servicios, referencias, declaraciones e información que describe las conexiones entre estos elementos. Además pueden enlazar componentes creados en distintas tecnologías ya que los *composites* constituyen implementaciones de estos componentes.

SCA define un formato de archivo en XML para la representación de sus artefactos. Además, estos artefactos y la forma de relacionarse entre ellos se representan mediante diagramas. Existen tres tipos de artefactos:

- Los **servicios**, dibujados en verde, son funcionalidades ofrecidas por las implementaciones de otros componentes con los que éste está relacionado. Determinan la entrada del componente.

- Las **referencias**, dibujadas en morado constituyen las relaciones de este componente con los demás. Determinan la salida del componente.

- Las **propiedades** son parámetros configurables que influyen en las operaciones. Pueden asemejarse a las opciones de personalización que existirán en alguno de los componentes utilizados en tiempo de creación.

El acceso a capacidades del entorno también estaría descrito a través de este modelo de componente. Como puede verse en la siguiente figura un componente del estilo del descrito arriba puede conectarse a través de sus entradas y salidas a otros servicios o componentes. Por lo tanto y por poner un ejemplo, una salida de un componente de generación de audio puede ser utilizada por otro componente que almacene el flujo de audio o por una capacidad, que reproducirá el flujo a través de un altavoz.

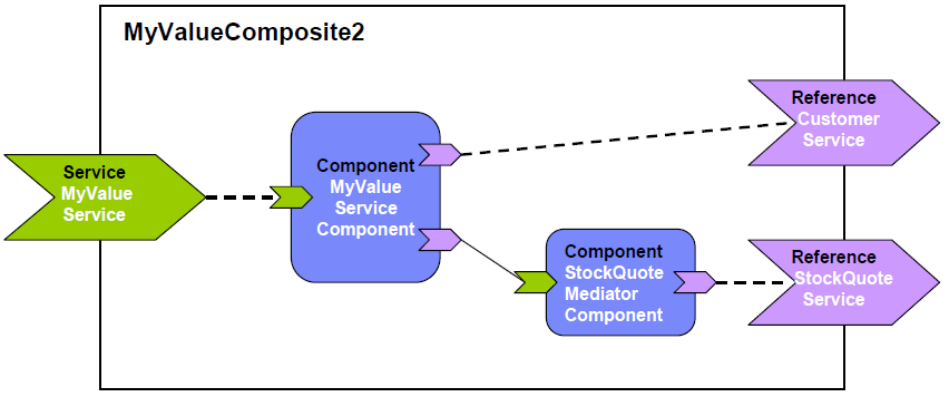

Figura 9. Composición en SCA

Como se puede observar existe una diferencia de concepto entre el modelo de composición SCA y el modelo de composición definido en este trabajo y es que existen algunos componentes que desde un primer momento van ligados a capacidades

(servicios en SCA). Para ello existe el concepto de "autowire" (autocableado), por la que se **simplifica el ensamblado de componentes.**

Conclusiones

En este apartado se exponen las ventajas y desventajas del uso de SCA como lenguaje de descripción de servicios para este trabajo:

Ventajas del uso de SCA

- Gracias al modelo de componentes descrito durante el desarrollo del estudio este lenguaje cumple las especificaciones para ser utilizado como lenguaje de descripción de la "vista descriptiva" de un servicio. Al contrario de otros modelos de descripción de servicios, el modelo impuesto por SCA describe un servicio en tiempo de creación como un conjunto de componentes, relacionados entre sí y que admiten una serie de propiedades o personalizaciones. Además SCA permite generar una versión ejecutable del modelo sin ninguna transformación intermedia, al contrario que otros lenguajes de modelado centrados en l diseño como UML o BPMN.

- El concepto de "smart-wire" es muy potente. El hecho de conectar servicios productores con servicios consumidores sin necesidad de que la conexión sea contra un único punto permite que existan lugares de conexión donde se recojan elementos del sistema que no sean fácilmente conectables al resto de componentes, como los mecanismos de seguridad, auditoría, etc. Estos elementos podrán ser considerados propiedades de los conectores en vez de proveedores o consumidores de servicios. Al mantener estos elementos como propiedades de los conectores su utilización resulta más comprensible para un usuario no experto además que, al establecer las políticas de seguridad en los conectores el impacto de un cambio en las políticas de seguridad no implica rediseñar el servicio y por tanto el código lógico de un servicio no se modifica.

Inconvenientes del uso de SCA

- Como lenguaje centrado en diseño sólo participa en aspectos concretos relacionados con el tiempo de ejecución. El uso de este lenguaje como alternativa para la definición de la vista interpretable de un servicio puede limitar las capacidades de la vista ejecutable una vez generada en el entorno de ejecución.

- La idea de soportar composición multilenguaje complica mucho esta arquitectura, aunque la capacidad de poder conectar, por ejemplo, un componente escrito en Java con otro escrito en C++ no está soportada. En la última revisión de este lenguaje se recomienda a los desarrolladores de componentes que les provean de una interfaz WSDL si van a proporcionarlo al sistema de manera remota.

- En general parece que se han detectado problemas con la utilización de SCA sobre entornos Java, y en especial sobre el API JSR-181, JAX-WS y EJB 3.

- No existen implementaciones para móviles.

3.2 Arquitecturas de servicios

3.2.1 Introducción

En este apartado se van analizar las arquitecturas de servicios existentes, intentando particularizar el estudio para dispositivos móviles. Estas arquitecturas siguen el modelo SOA, que basándose en las tecnologías de aplicaciones distribuidas y diseños orientados a objetos, aglutinan a servicios desarrollados en distintas tecnologías que se comunican a través de un interfaz común.

Los servicios Web son los mecanismos que se están imponiendo en los últimos tiempos a la hora de publicar aplicaciones y servicios en Internet, y por tanto, son una parte fundamental de las Arquitecturas tipo SOA. Dentro de las arquitecturas de servicios basadas en servicios Web, existen dos grupos fundamentales, arquitecturas basadas en SOAP (*Simple Object Access Protocol*) y las basadas en el modelo REST (*Representational State Transfer*).

En primer lugar se analizará el modelo SOA y se mencionarán distintas arquitecturas SOA existentes y posteriormente se analizará el modelo REST, que es la alternativa principal a SOAP y a WSDL. REST destaca por una gran ligereza y flexibilidad lo que le ha permitido formar parte de grandes sistemas como Google o Facebook.

3.2.2 Arquitectura SOA

La arquitectura orientada a servicios es tanto un marco de trabajo para el desarrollo de software como un marco de trabajo de implantación. Las metodologías de desarrollo de software orientadas a servicios han tenido en los últimos tiempos una gran aceptación apareciendo, tanto en entornos académicos como en el mundo empresarial, multitud de arquitecturas de referencia para soluciones SOA junto con su correspondiente metodología de desarrollo de software.

Estas metodologías son útiles para los proyectos de integración de aplicaciones empresariales (EAI) y el desarrollo de sistemas Business to Business (B2B). Algunos ejemplos de estas metodologías son:

- SAE - *Service Architecture and Engineering* (CBDI) [36]
- SMART - *Service Oriented Migration and Reuse Technique* (SEI) [37]
- SOAD - *Service-Oriented Analysis and Design* (IBM) [38]

- SODA - *Service-Oriented Development of Applications* (Gartner) [39]
- SOMA - *Service-Oriented Modelling and Architecture* (IBM) [40]
- RUP para SOA (IBM) [41]

De forma resumida, podemos decir que la arquitectura orientada a servicios (SOA) es un enfoque para la **construcción de sistemas software donde la funcionalidad se agrupa en procesos de negocio** y se **empaqueta como servicios autónomos e ínter operables**. Por tanto, generalizando, una arquitectura SOA describe la infraestructura IT que **permite a diferentes aplicaciones intercambiar datos entre sí**, cuando éstos están ejecutando esos procesos de negocio.

SOA supone una estrategia general de organización de los elementos de IT, de forma que una colección de sistemas distribuidos y aplicaciones complejas se pueda transformar en una **red de recursos integrados simplificada y flexible**. SOA establece un marco de diseño para la integración de aplicaciones independientes de manera que desde Internet pueda accederse a sus funcionalidades, las cuales se ofrecen como servicios. La solución final se compone, en general, de servicios desarrollados en diferentes lenguajes de programación, ubicados en plataformas muy dispares, con distintos modelos de seguridad y procesos de negocio.

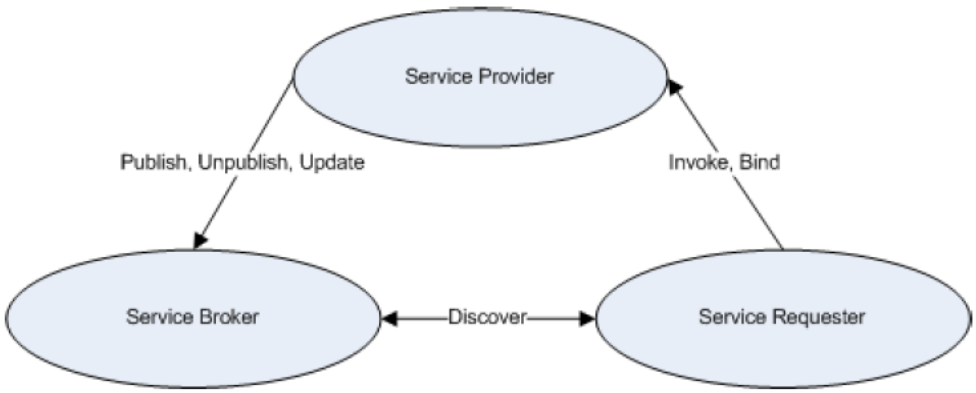

Figura 10. Modelo orientado a servicios

A continuación se presenta una tabla con las ventajas y desventajas derivadas de la implantación de una arquitectura SOA:

Tabla 2. Ventajas y desventajas de la utilización de SOA

SOA	
VENTAJAS	**Fácil de utilizar** debido a la madurez de la tecnología.

	Alta interoperabilidad y escalabilidad. Envolver APIs escritas en diferentes lenguajes es sencillo. **Incrementa la privacidad** mediante el uso de estándares WS-*.
DESVENTAJAS	Alto **gasto computacional** debido a la necesidad de procesado en la interpretación de los mensajes XML. **Baja optimización del espectro** debido al uso de grandes cabeceras en los mensajes SOAP.

Como puede verse en la tabla, los problemas de SOA tienen bastante importancia de cara a una implementación en un ambiente móvil. Es necesario tener en cuenta que la optimización de los recursos en una solución móvil es de vital importancia debido a la naturaleza de la misma (problemas de batería, perdida de conexión, etc.). Sin embargo, la continua mejora en la capacidad de procesado de los dispositivos móviles junto con el aumento del ancho de banda en redes móviles puede mitigar este efecto negativo.

3.2.3 OSGi

La tecnología OSGi [42] (Open Service Gateway initiative) proporciona un entorno de desarrollo basado en Java y compatible 100% con J2ME (Java 2 Mobile Edition). Este entorno de desarrollo permite manejar componentes dinámicos y las interacciones entre estos y, demás, permite a los desarrolladores administrar remotamente el ciclo de vida de las aplicaciones. Por ejemplo, se puede instalar, desinstalar, ejecutar o detener cualquier aplicación desplegada como componente de OSGi.

El "framework" de OSGi fue diseñado inicialmente para ser utilizado en dispositivos con gran porvenir comercial, ya que necesitan ejecutar una gran variedad de software. Éste es el caso de puertas de enlace domésticas (set-top boxes), dispositivos para la automoción y dispositivos PDA (Personal Digital Assistant). Conforme se ha ido descubriendo la potencia de esta tecnología su utilización se ha situado más allá del contexto de puerta de enlace. Por ejemplo, OSGi ha sido seleccionada como la tecnología que constituye el núcleo de la plataforma de desarrollo Eclipse y como la base de aplicaciones empresariales como Java EE Spring.

Las aplicaciones en OSGi están empaquetadas en forma de bundles, que en realidad tienen el formato de un archivo JAR. Como un bundle de OSGi está completamente autocontenido (ver características de los contenedores), toda la información necesaria para que el bundle funcione debe permanecer dentro del propio bundle. En este caso está contenida en un archivo de manifiesto (MANIFEST.MF).

Para que un bundle pueda exportar sus servicios al resto de la plataforma o para que pueda utilizar los servicios que ofrecen los demás bundles estos servicios deben estar registrados. Para que los servicios registrados estén disponibles, los bundles que los implementan deben estar instalados en la plataforma. A partir de ahora hablaremos del estado de un bundle cuando nos refiramos a la situación en la que se encuentra dentro del framework de OSGi.

Conclusiones

La tecnología OSGi define una plataforma de composición y provisión de servicios con características muy similares a la plataforma que diseñamos en este trabajo. Por tanto, se utiliza esta tecnología para la evaluación y la validación de la plataforma, como podrá verse en la sección 5 Definición de arquitectura genérica para la provisión de servicios.

De todas formas, se han detectado algunas carencias en esta tecnología, ya que por ahora no existen soluciones para la gestión de componentes que no existen en la plataforma. Un escenario simple con dos componentes que intercambian datos puede ser simulado por un framework OSGi con dos bundles, uno como proveedor de datos y el otro como consumidor. Para realizar el intercambio de datos el bundle proveedor debe declarar un servicio para que el consumidor pueda buscarlo según un filtro basado en propiedades de los bundles. De hecho, utilizando esta tecnología necesitamos lidiar con limitaciones en el lado proveedor (falta de dinamismo en la declaración de propiedades de un servicio) y en el lado consumidor (pobre expresividad en la declaración de los filtros). Por eso, se requiere extender la plataforma OSGi con algunos mecanismos que contribuyen a establecer las conexiones entre componentes de forma apropiada. Estos mecanismos se describirán en la sección 5.

3.2.4 REST

REST ***(Representational State Transfer)*** es un modelo de arquitectura de software para sistemas distribuidos. Este término se originó en el año 2000, en una tesis doctoral sobre la *Web* escrita por *Roy Fielding* [43], motivado por crear un modelo representativo de cómo debería funcionar la Web y que sirviese de guía para los protocolos estándar Web existentes.

REST pretende crear una serie de principios que puedan describir la arquitectura de la Web, identificar problemas existentes, comparar soluciones alternativas y asegurarse de que las extensiones de los protocolos no violan las restricciones que hacen que la Web sea útil.

El término REST, en principio, describía un conjunto de principios de arquitectura software para sistemas distribuidos, en la actualidad se utiliza en un sentido más amplio para describir cualquier interfaz Web basada en XML y HTTP, sin las abstracciones adicionales de protocolos basados en patrones de intercambio de mensajes (como por ejemplo SOAP).

Conceptos* y *definiciones sobre las que se basa el modelo de REST:

- **Recursos:** Entidades de información. Por ejemplo una página Web HTML, un documento XML, un Servicio Web, un dispositivo físico, etc.
- **Representación de los recursos:** Es un reflejo o "imagen" del recurso en el instante en el que el cliente lo solicita.
- **URLs para identificar los recursos**: Cada recurso debe ser identificado por una URL única.

Conclusiones

REST es una alternativa de diseño de servicios web con una menor dependencia en *middleware* que otras tecnologías como SOAP y WSDL. En cierta medida, REST supone una regresión a los primeros estándares de Internet (URI y HTTP) anteriores a la aparición de los grandes servidores de aplicaciones.

Al basarse en XML sobre HTTP es una interfaz muy flexible que permite que tanto aplicaciones internas como aplicaciones basadas en AJAX *(JavaScript Asíncrono + XML)* puedan conectarse, ubicar y consumir recursos. Debido a esta combinación con AJAX, REST genera tanta atención hoy en día.

Resulta muy flexible el poder exponer los recursos del sistema con un API REST, de manera que ofrezca datos a distintas aplicaciones, formateados de forma diferente. REST ayuda a cumplir con los requerimientos de integración, que son críticos para construir sistemas en donde los datos tienen que poder combinarse fácilmente (*mashups*) y extenderse.

3.3 Arquitecturas de red

Las arquitecturas de red definen las tecnologías utilizadas para proveer un servicio y la red en la cual se ofrecerá. Las arquitecturas de red abordan los problemas de una manera global, a la vez que ayudan a planificar, evaluar y seleccionar los distintos componentes necesarios para que un sistema funcione correctamente, además de las consecuentes necesidades de interoperabilidad que puedan surgir en la implementación.

Dependiendo del tipo de servicio, sus características y complejidad, se pueden utilizar una o varias arquitecturas de red que provean una solución específica para cada capa que lo componga, por lo tanto, las arquitecturas de redes son necesarias ya que regulan el diseño, construcción, administración y operación de una red de comunicación. Este análisis permitirá contrastar el estado del arte de las distintas arquitecturas de red, su viabilidad, y las ventajas que pudieran ofrecer al ser aplicadas al proyecto.

3.3.1 IMS

IMS [44] es un sistema de control de sesión diseñado con tecnologías de Internet adaptadas al mundo móvil, que hace posible la provisión de servicios móviles multimedia sobre conmutación de paquetes (servicios IP multimedia). El Subsistema Multimedia IP (IP Multimedia Subsystem) forma parte del núcleo de la arquitectura de las redes de siguiente generación. Estas redes son capaces de proporcionar servicios multimedia fijos y móviles.

El IMS utiliza protocolos estándar del IETF, aunque adaptados a los requisitos de los operadores públicos, que entre otros factores incluyen el soporte regulatorio de los diferentes países y las facilidades de interconexión de diferentes operadores del mismo o diferentes países. Si bien estas capacidades de interconexión se limitan fundamentalmente a las facilidades básicas, ya que los servicios avanzados sufrirán de problemas de interoperabilidad debido a los mecanismos utilizados para implementarlos y negociarlos.

Sin embargo el aspecto más relevante que podemos citar del IMS es la separación entre los niveles de aplicación, el de gestión de Servicios y la capa de transporte, junto con la definición de punt0s de referencia con las redes de acceso y otras redes de comunicaciones, como pueden ser la red telefónica convencional.

Estos puntos hacen el IMS una arquitectura flexible y adaptable a cualquier entorno, por lo que ha sido utilizada como base para redes diferentes de las redes móviles europeas para las cuales fue originalmente diseñado, y exportado a otros entornos.

3.4 Desarrollo móvil

En este apartado se describen las tecnologías que se van a utilizar para el desarrollo de una plataforma móvil de despliegue de servicios.

3.4.1 Java Micro Edition (JME)

JME es un subconjunto de la plataforma Java, de Sun Microsystems. Provee de los API´s necesarios para la programación de pequeños dispositivos de consumo como pueden ser PDA´s, teléfonos móviles, etc.

JME no es propiamente un sistema operativo, sino una plataforma de desarrollo, y la máquina virtual (VM) que ejecuta las aplicaciones puede estar implementada sobre Symbian, Windows Mobile, etc. Sin embargo, en numerosos terminales de gama media/baja, que utilizan sistemas operativos propietarios y cerrados (como algunas de las mencionadas en el apartado de Linux), es la única plataforma que permite añadir aplicaciones de terceros. Por otro lado, se trata con diferencia de la plataforma de desarrollo más extendida en el parque de terminales móviles.

LWUIT

Para programar interfaces gráficas en JavaME hay dos opciones: usar los formularios o usar el *canvas*. El primero es mucho muy básico. Provee muy pocos elementos para hacer una UI por lo que las aplicaciones hechas así, son poco atractivas.

La solución a un buen número de estos problemas nos la proporciona la librería LWUIT (*Lightweight UI Toolkit for Java ME*), que está inspirado por Swing. LWUIT permite crear aplicaciones modernas, agradables, al mismo tiempo que hace más sencilla la adaptación de las mismas a los diferentes terminales móviles.

3.4.2 Flashlite

Permite a los desarrolladores proveer a los consumidores experiencias completas de navegación Web, video, y un compendio de contenidos interactivos. Flash Lite permite crear aplicaciones móviles de forma sencilla y rápida, lo que permite seguir las rápidas variaciones del mercado y adaptarse a él.

La integración entre Flashlite y JME puede lograrse a través de iniciativas como la del proyecto Capuchin.

Capuchin

La iniciativa de Sony Ericsson conocida como proyecto Capuchin [45], permite a los desarrolladores combinar la riqueza de J2ME y Flash Lite conjuntamente encapsulando Flash Lite en las aplicaciones J2ME. Esto se hace posible mediante el API Capuchin y la herramienta Swf2Jar que permite ver Flash Lite como la capa de presentación y J2ME como la capa lógica de aplicación. Viéndolo de esta forma, las herramientas Flash pueden usarse para diseñar la UI mientras todavía tienen acceso a los servicios del dispositivo móvil a los que también tiene acceso J2ME.

El proyecto Capuchin obtiene las ventajas de Flash en cuanto a la velocidad de desarrollo del UI, utilizando las grandes herramientas que nos ofrece, mientras que también se beneficia de Java en cuanto a seguridad, diseño de servicios y a su estructura para la posterior distribución. El primer dispositivo en el que podemos observar el proyecto Capuchin integrado es el Sony Ericsson C905.

3.5 Mecanismos de búsqueda y descubrimiento de servicios

En este apartado se describen los mecanismos de descubrimiento más habituales que existen actualmente y que puede ser interesante estudiar para su aplicación en un entorno de movilidad.

3.5.1 SDP

SDP [46] (*Service Discovery Protocol*) es un protocolo definido en Bluetooth que proporciona un mecanismo para descubrir qué servicios ofrece un dispositivo través de la interfaz Bluetooth. Cada dispositivo Bluetooth mantiene un servidor SDP escuchando en un puerto conocido. Este servidor contiene la lista de servicios que el dispositivo ofrece así como información relacionada con dichos servicios (p.e. en qué puerto está disponible el servicio). De esta forma un dispositivo cliente puede preguntar por un servicio determinado al servidor (o incluso por la lista de servicios disponibles) y éste retornarle toda la información necesaria para llevar a cabo la conexión.

3.5.2 UPnP SSDP

UPnP SSDP [47] (*Simple Service Discovery Platform*) es un protocolo de descubrimiento en redes de área local que forma parte de la pila de protocolos de UPnP. UPnP SSDP es utilizado por la tecnología UPnP para el descubrimiento de los servicios en la red, sin necesidad de repositorio central de descripción de servicios. Para ello los dispositivos utilizan mensajes multidifusión o unidifusión sobre UDP para anunciar sus servicios. En una red IP, los dispositivos pueden autoconfigurar su dirección IP ante la ausencia de un servidor DHCP.

3.5.3 JINI – Apache River

Jini [48], Apache-River [49] es una arquitectura distribuida basada en Java y que permite el descubrimiento de los servicios disponibles en el sistema distribuido. Ha sido desarrollada por Sun Microsystems y pretende automatizar la creación y administración de la red formada por federaciones de máquinas virtuales Java.

Tras no alcanzar la repercusión esperada el proyecto quedó durante un tiempo abandonado, hasta que volvió a resurgir de mano de la "Apache Software Foundation" [50] bajo el nombre de "Apache River". Actualmente se encuentra en la "Apache Incubator" a la espera de que en próximas revisiones se compruebe la viabilidad del mismo y pase a formar parte definitivamente de la Apache Software Foundation.

3.5.4 Conclusiones

Del estudio del estado del arte para el descubrimiento en el entorno *prosumer* en movilidad se desprende que para realizar un descubrimiento en este entorno, indiferentemente de si estamos en proximidad o en red, parece más apropiado el uso de mecanismos de descubrimiento basados en arquitecturas con directorios, como es el caso de SDP, Jini, UDDI. Ya que este tipo de arquitecturas nos permiten un ahorro considerable en el ancho de banda consumido por el mecanismo en el intercambio de mensajes necesario para llevar a cabo el descubrimiento, algo que va a resultar crucial en el entorno de movilidad donde la conectividad puede ser limitada en algunas situaciones.

3.6 Mecanismos de publicación y registro de servicios

Los mecanismos de publicación de servicios usados en la actualidad se centran fundamentalmente en la publicación en un registro centralizado de la descripción y forma de acceso a dichos servicios. Estos registros pueden ser accedidos por clientes para conocer que entidades proporcionan servicios, que servicios proveen, las características de estos servicios y cómo y dónde acceder a ellos.

3.6.1 UDDI

UDDI [51] es el protocolo Universal de Descripción, Descubrimiento e Integración aprobado como estándar por OASIS. Se trata de un miembro clave de la pila de Servicios Web y define un método estándar para la publicación y descubrimiento de componentes software basados en red dentro de la arquitectura orientada a servicios SOA.

El registro UDDI nos proporciona el estándar apropiado para el registro de servicios. Cabe mencionar que UDDI almacena, y por lo tanto devuelve, los WSDL que describen los servicios que son alojados en él, para que el cliente sepa como acceder al servicio prestado por el proveedor de dicho servicio.

Por lo tanto en nuestro caso, siguiendo la especificación de UDDI almacenaremos nuestro propio SLD creado para la plataforma diseñada y en cierto modo puede que el

comportamiento del registro cambie al cambiar las funcionalidades o inteligencia que residirá en dicho registro.

3.7 Herramientas de creación de servicios

En esta sección se identifican las herramientas más actuales y que han causado mayor impacto en el ámbito de la creación de servicios por parte de usuarios. En este estudio nos centraremos en las aplicaciones para móviles, para ajustarnos a los criterios establecidos al definir el entorno del trabajo (usuario *prosumer* en movilidad).

Las herramientas analizadas pueden clasificarse en dos grupos. El primero de ellos describe las herramientas disponibles para la creación de pequeñas aplicaciones y el segundo las herramientas que gestionan Widgets en el entorno móvil.

3.7.1 Herramientas para la creación de pequeñas aplicaciones

Hoy en día cada vez resulta más necesario utilizar el móvil "para algo más que para llamar", el usuario del terminal busca ya no que su terminal sea el que más *gadgets* tenga (GPS, WiFi, acelerómetro, radio…) si no que con este pueda realizar todas aquellas funciones que se le ocurran y que supongan un aprovechamiento máximo de la capacidad del mismo.

A continuación se presenta una pequeña lista de posibles herramientas para la creación de pequeñas aplicaciones (o juegos,) tanto en entorno fijo como móvil, que se han tenido en cuenta previamente a la realización de este apartado y que han servido de base a la elección de la herramienta objetivo del estudio (*MoBots*):

- MyGame [52]: herramienta para la creación de juegos en flash añadiendo fotos desde entorno Web.
- Scratch [53]: creación de proyectos pequeños en JAVA tales como juegos, herramientas matemáticas, presentaciones, puzzles, ilusiones ópticas, traductores, etc. Requiere la descarga de un instalable para la creación de los proyectos a compartir.
- Long Jump [54]: permite crear aplicaciones de baja complejidad online sin tener conocimientos de programación (calendarios, agendas, listas, hojas de cálculo…).
- Zoho [55]: Herramienta que permite la creación de aplicaciones a usuarios sin conocimientos de programación mediante el uso de formularios.
- Mobots [56]: herramienta para generar aplicaciones para el móvil que posteriormente se puedan compartir. Orientada a usuarios sin conocimientos de programación. Es necesario descargarse el software de desarrollo que en principio sólo está disponible para plataformas *Symbian* de tercera generación.

Mobots

MoBots [56] es una herramienta que permite la creación de pequeñas aplicaciones en un terminal móvil. *MoBots* son pequeñas aplicaciones que se pueden crear a través de una interfaz muy sencilla para el usuario y que se ejecutan de manera automática en determinadas situaciones y ante una lista de eventos concretos, de ahí su nombre "*Mobile Robots*" o *Mobots*.

Esta herramienta resulta bastante simple ya que contiene solamente tres bloques para construir dichas aplicaciones:

- **Eventos**: bloques de color **amarillo** que representan eventos capaces de despertar la ejecución del Robot en cuestión (p.e. el reloj, llamadas o mensajes entrantes, encontrarse en una determinada localización o en un rango de bluetooth…).

- **Condiciones**: bloques de color **verde** que representan las condiciones que se deben de dar cuando un *MoBots* recibe el evento anterior para permitir la ejecución de una tarea, como por ejemplo el perfil que debe estar activo, la hora, la localización…

- **Comandos**: bloques de color **azul** que representan las acciones que tendrán lugar cuando ocurre todo lo anterior (p.e. vibración, una llamada, envío de un mensaje…).

De esta forma, es posible crear pequeñas aplicaciones que realicen automáticamente y sin interacción directa del usuario, ciertas operaciones en base a unas condiciones previas concretas. Esta creación se realiza mediante la adición de los tres tipos de bloques anteriormente comentados, sin más que interaccionar con las teclas del móvil:

Figura 11. Interfaz de creación de robot en la herramienta MoBots

- Bloques amarillos: eventos que van a despertar al *MoBots* en cuestión (p.e. una llamada, un mensaje entrante, una localización…)

- Bloques verdes: condiciones que se deben dar para que al llegar el evento se realice el comando posterior.

- Bloques azules: comandos que realizará el terminal móvil una vez verificados los dos bloques anteriores.

3.7.2 Herramientas para la gestión de Widgets en entorno móvil

Un *Widget* [57] es una pequeña aplicación que es ejecutada mediante un motor de ejecución especialmente diseñado para estos *Widgets*. Estos elementos destacan por su enorme sencillez, su gran vistosidad y su fácil acceso. Las funcionalidades que implementan son de naturaleza distinta: relojes, calculadoras, previsualización de correo electrónico, noticias, etc.

Entre las aplicaciones para entorno móvil que crean servicios para móviles basadas en widgets destacan:

- *Webwag*: aplicación para integrar en dispositivos móviles todos aquellos *widgets* que pueden interesar de una manera fácil y rápida. Requiere la descarga e instalación en el dispositivo móvil.

- *WidSets*: herramienta gráfica para crear *widgets* para móviles, fundamentalmente orientada a la creación de RSS feeds.

- *Yahoo! Go:* motor de *widgets* que permite tener en los dispositivos móviles todo aquello que se necesite procedente de Internet.

- Widgadget: herramienta para creación de *widgets* desde una aplicación Web. Utiliza una URL para extraer el *feed* de la misma.

3.7.3 Conclusiones

En este apartado se reflejan unas conclusiones extraídas del análisis de las herramientas de creación se serán utilizadas en forma de requisitos y restricciones en la fase de diseño de la interfaz de usuario del entorno de creación.

- Pantalla del terminal móvil con reducidas dimensiones. Deben presentarse los **menús en forma de listado** que, mediante **iconos** fácilmente distinguibles, permite al usuario cubrir todas sus necesidades de una manera rápida e intuitiva.

- **Agrupación de los elementos** a usar en la creación por categorías, de forma jerárquica y de fácil navegación.

- **Previsualización del servicio** creado antes de su publicación.

- Posibilidad de **configurar** que el servicio **pida permiso** al usuario del terminal antes de realizar algún tipo de operaciones, como el acceso a internet o la escritura de ficheros, de manera sencilla y eficaz.

- Existencia de **asistentes** que ayuden al usuario a crear su servicio o posibilidad de escoger **servicios genéricos** para que el usuario no experto no tenga que empezar a crear su servicio desde creo.

3.8 Conclusiones del análisis

Existen algunas conclusiones tecnológicas que se han generado como resultado de la revisión de las tecnologías y las definiciones de los conceptos pertenecientes al universo inteligente:

- Para generar un entorno móvil en el que los usuarios creen, provean y consuman servicios desde sus terminales en movilidad es necesaria una infraestructura externa al terminal móvil, ya sea en red o próxima, que le ofrezca soporte y amplíe su funcionalidad. La funcionalidad del entorno debe ser independiente de la infraestructura subyacente, que debe basarse en la **convergencia de redes** para ofrecer a los usuarios y desarrolladores de la plataforma una infraestructura estable y disponible en todo momento.

- Dentro del entorno propuesto, se han identificado muchas tecnologías disponibles para resolver las necesidades que se derivan de su funcionalidad, lo cual pone de manifiesto la **gran fragmentación tecnológica que existe en el mundo móvil**. La desventaja de este hecho reside en la poca profundización existente en el desarrollo de algunas tecnologías para móviles, como las plataformas de creación de servicios y los entornos de composición.

- Las tecnologías disponibles actualmente no ofrecen soluciones completas para resolver los problemas y retos impuestos por el entorno *prosumer*. Por este motivo se decide abordar las tareas del proyecto considerando la posibilidad de definir nuevos lenguajes y modificar las tecnologías existentes.

- Muchas de las tecnologías necesarias para la realización de ciertas tareas no están disponibles en la actualidad para entornos móviles y se necesitará de una adaptación o simulación para lograr los objetivos propuestos.

4 Servicios en el nuevo Universo Inteligente

El propósito de esta sección es explicar el proceso efectuado desde la definición de los escenarios globales para todo el proyecto hasta definición de servicio en el contexto de Universo Inteligente, pasando por la extracción de los requisitos de usuario en movilidad, los cuales se presentarán en la siguiente sección.

El alumno se ha involucrado en la tarea de análisis de los escenarios globales del proyecto y en la definición de los elementos que intervienen en el entorno: servicio, componente, capacidad y su estructura lógica.

4.1 Definición de escenarios

Los escenarios tienen un papel determinante para la definición de casos de uso y requisitos ya que constituyen el nexo de unión entre las distintas actividades del proyecto y de las tecnologías utilizadas. Por ello, se pretende que los escenarios generados contemplen la mayoría de características deseadas en el sistema y que abarque todos los ámbitos y circunstancias que lo rodean.

En esta sección se describirá un escenario de entre los analizados (alrededor de 20 [8] [9]), que describe la mayoría de las características que se quieren introducir en el diseño de la plataforma objeto de este trabajo.

4.1.1 Escenario: Sport Tracker

Raúl es un gran aficionado al *jogging*, y entrena a diario para mejorar su marca y su condición física. Para mostrar sus avances a sus compañeros del club de *jogging* (usuarios), le gustaría que ellos pudieran ver, en tiempo real durante el entrenamiento, su posición en un mapa, la distancia recorrida y su pulso cardíaco. Raúl decide crear un servicio "Sport Tracker", para lo que dispone de su **teléfono con GPS** y **un pulsómetro compatible**. En otro punto de la ciudad Cristina, amiga de Raúl, está interesada en ver los avances de Raúl en *jogging*.

Raúl toma su móvil, abre la aplicación de creación de servicios y selecciona la opción **crear un servicio nuevo**. Añade una etiqueta de título que contiene "Entrenamiento de Raúl". Más adelante cambiará el color del fondo y quizás incluya alguna fotografía.

En el **menú de componentes de red** encuentra rápidamente un **mapa** (p. ej, Google Maps). La aplicación le ofrece la posibilidad de mostrar una dirección o buscar otra fuente de posición en otro componente o en otro servicio, pero Raúl no le hace caso de momento y añade un mapa sin posición.

Pasa ahora a un **menú de componentes locales** y encuentra, entre otras, la **posición, velocidad y pulso cardíaco**. Al seleccionarlas, la aplicación supone que posición y velocidad se expresarán dentro del mapa. El pulso cardíaco, por ahora, se muestra como un campo de texto que se actualiza en tiempo real. En este momento, el pulsómetro está apagado, pero la capacidad asociada (el pulso cardíaco) aparece de todas formas. Al iniciarse el servicio, éste comprobará dinámicamente si está disponible la capacidad y la vinculará al componente que el usuario seleccionó.

Raúl publica su servicio para que pueda ser utilizado por otras personas y selecciona la opción de publicar servicio, eligiendo el nombre "Sports Tracker" para identificarlo, lo cual crea una plantilla del servicio que se almacena en el Repositorio de Plantillas. Esta plantilla consta de un archivo SDL, que describe al servicio en todas sus facetas (Componentes, interacción con capacidades y lógica [de funcionamiento interno y de orquestación externa]) además de tener asociado un modelado de conocimiento que describe la esencia del servicio.

Raúl decide que es un buen momento de salir a correr y, mediante **su gestor de servicios publicados** decide iniciar el servicio de Sports Tracker. Para ello va al **Repositorio de Plantillas** donde acaba de subir su plantilla, la selecciona y la configura. La información de configuración genera una Instancia de Servicio con el identificador de usuario de Raúl. Esta instancia se despliega en el **Servidor del teléfono móvil**, que es desde donde se provee el servicio. Además, se envía una referencia de esa instancia al **Repositorio de Instancias** de Servicios como "Sigue el entrenamiento de Raúl". A partir de este momento, Raúl se convierte en el proveedor de este servicio.

Parte de la configuración que debe realizar es restringir los usuarios que tendrán acceso al servicio, especificando al grupo de compañeros de *jogging* como los únicos permitidos. En el proceso de publicación, además, se contacta con la **capacidad de presencia** para cambiar el estado de Raúl.

En otro punto de la ciudad Cristina, amiga de Raúl, comprueba en su **Agenda de contactos en red** que el estado de Raúl ha cambiado. Ha pasado a ser: "*Raúl ofrece el servicio Sports Tracker. Conéctate a su servicio!*". Cristina, intrigada por el servicio de Raúl, decide ver los detalles del servicio, comprobando que Raúl ha salido a correr y está publicando sus datos mediante este servicio. Cristina busca instancias del servicio de nombre "Sigue el entrenamiento de Raúl" y encuentra la instancia ofrecida por Raúl para convertirse en cliente de ese servicio.

Cristina lanza su **Cliente de servicios** que se conecta al **Servidor del teléfono móvil** de Raúl. El cliente descarga el código necesario para que pueda operar el servicio. El

cliente con el código descargado es capaz de resolver los componentes definidos en el servicio, de gestionar y orquestar la información que publica el servicio de Raúl (pulsómetro y GPS) y la información obtenida de **capacidades en red** (Mapas del API de Googlemaps en función de las coordenadas GPS) mediante una lógica de servicio representado el resultado sobre la **interfaz gráfica de servicio** definida por Raúl.

Viendo que Raúl está corriendo por la zona, Cristina decide unirse a él siguiendo además la misma iniciativa Sports Tracker. Busca en las plantillas del almacén el servicio por nombre de servicio y decide ejecutar una instancia de él también. Al ejecutarse el servicio el módulo de **Armonización de capacidades** detecta que aunque Cristina tiene GPS no tiene el pulsómetro bluetooth. El armonizador informa al servicio que notifica a Cristina que no publicará esta información pero como el componente no fue definido como restrictivo puede ofrecer el servicio en modo limitado de posicionamiento.

A continuación se representa un gráfico de estados que describe el escenario desde el punto de vista de la interacción del usuario con la interfaz de creación de servicios:

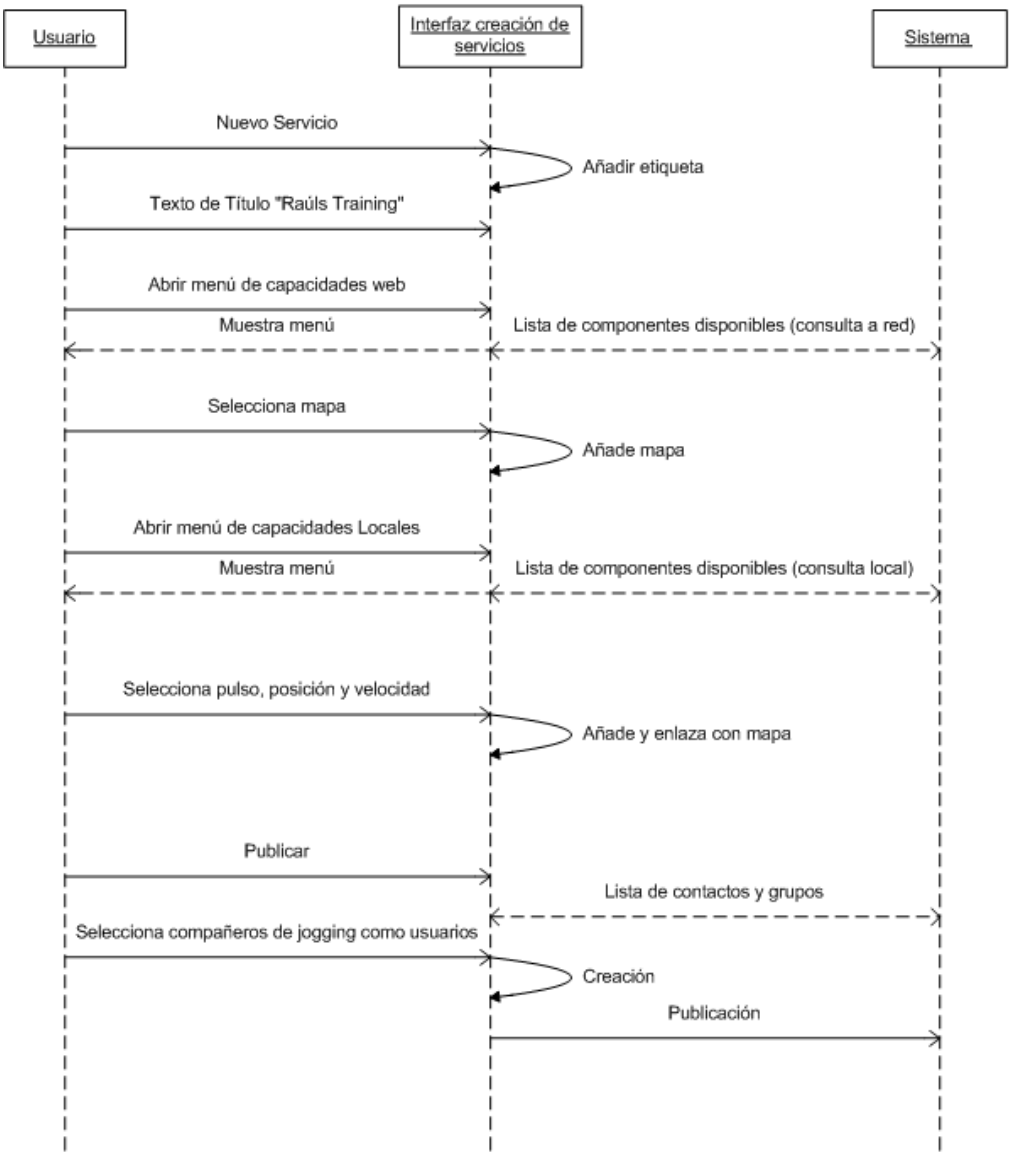

Figura 12. Gráfico de estados, escenario Sport Tracker

4.1.2 Análisis del escenario

El escenario *Sports Tracker* pretende definir el ciclo completo de un servicio mIO! Creado y proporcionado por un usuario móvil. El ciclo de vida abarca desde que el usuario concibe el servicio hasta que otro usuario decide consumir este servicio. En el escenario se destaca el papel del usuario como eslabón que une distintas herramientas y tecnologías para cubrir uno de los objetivos principales de este trabajo: que los usuarios sean los nodos activos de una red de creación y provisión de servicios. Por lo tanto se deduce de la lectura general del escenario que los usuarios van a ser el motor que mantenga viva la actividad del sistema, tanto en la creación como en la provisión y consumo de servicios.

Desde el punto de vista de la **creación** de servicios, es necesario suministrar herramientas de edición intuitivas que permitan a los usuarios abstraer las tecnologías que se usen en los servicios. Con este objetivo, en el marco de este trabajo se han analizado algunas herramientas y tecnologías de creación de servicios en la sección 3 **Estado del arte de servicios en movilidad**. Se introduce el concepto de componente como unidad conceptual manejada por los usuarios para expresar los servicios. Adicionalmente, el escenario da pie a la definición de plantilla de servicio, como el documento que contiene el "código fuente" del servicio, reflejo de las operaciones de creación que realicen los usuarios, y que puede volverse a carga en el entorno de creación para ser modificado.

La **publicación** es la consecuencia natural de la creación y por lo tanto su paso consecutivo. Los usuarios, mediante la herramienta de creación, podrán publicar sus servicios. Para lograrlo, es necesaria la presencia de mecanismos que identifiquen a los creadores de forma segura para garantizar una red de confianza. Los procesos de publicación deberán ser ligeros para el usuario pero a su vez deben realizar una indexación del servicio tan rica como sea posible para facilitar su futuro acceso, búsqueda y recomendación. Del efecto de publicar se deriva la necesidad de mantener un sistema de almacenamiento de servicios accesible por todos los usuarios del entorno.

Una plantilla, resultado de un proceso de creación, no será un servicio en funcionamiento, sino que tan solo un servicio en potencia, que se convertirá en instancia cuando se despliegue dentro del entorno de **provisión** de servicios incluido en los dispositivos móviles de los usuarios. Como se identifica en el escenario, un usuario para ofrecer un servicio al resto de la comunidad, tendrá que obtener (bien de un repositorio en red, bien de otro usuario, bien creándola él mismo) la plantilla del servicio que quiera ofrecer y trasladarla a su móvil. Este proceso implicará un registro de servicio en provisión para que pueda ser accedido por el resto de los usuarios, pudiendo definir políticas de acceso basadas en los mecanismos de identidad y seguridad que se definan.

Otro requisito de alto nivel que se puede extraer del ejemplo es que los servicios se pueden **descubrir** mediante diferentes mecanismos de descubrimiento tanto manuales como automáticos.

Finalmente, en el escenario se ejemplifica la **ejecución** de un servicio. Cuando un servicio se consume desde un cliente, es necesario pasar del plano conceptual definido en la creación a un plano físico de uso de capacidades y recursos. Se necesita resolver lo que quería idealmente hacer el creador en lenguaje de componentes, con lo que en esa situación específica de provisión-consumo

existe. El módulo **armonizador de capacidades** es responsable de realizar la vinculación entre planos de creación y ejecución.

4.1.3 Conclusiones

Del análisis de los escenarios propuestos y, en especial, del escenario Sport Tracker, deben extraerse conclusiones que puedan transformarse en requisitos del sistema.

En el ámbito de la **creación y publicación de servicios** se contempla la posibilidad de poseer un asistente de creación en el terminal móvil que, de acuerdo con la información de perfil, el contexto del terminal y la información del entorno puedan ayudar al usuario con la tarea de creación.

Un servicio está compuesto por un conjunto de componentes relacionados entre sí. Un servicio se compone también de lógica (relaciones lógicas del código fuente), presentación (capa de interacción del servicio con el usuario) y la descripción del servicio (metadatos asociados y vista del servicio como conjunto de componentes).

La información relativa a la definición y descripción de estas 4 partes de un servicio debe residir en el SDL (Service Description Language) que se genere mediante un motor de creación del servicio.

El mecanismo de **publicación** se define como el procedimiento por el cual un servicio se da a conocer, enviando información de provisión (una referencia URI, metadatos, ID de ese servicio) al repositorio de instancias/plantillas. En la creación de la instancia se produce una resolución de capacidades y se introduce información de contexto.

En cuanto al ámbito de ejecución de un servicio, los escenarios nos indican que todos los servicios se ejecutan en el terminal que los provee. Así, una vez que el usuario que desea ejecutar el servicio accede a éste se descarga una lógica cliente del servicio, mientras que en el terminal servidor del servicio reside una lógica de servidor.

Hay que tener en cuenta en esta fase de la **ejecución** de un servicio que debe existir un elemento en el terminal que provee el servicio, que reúna la información de contexto (que incluyen el entorno, el contexto local y el perfil) y la transmita a los consumidores del servicio (que están ejecutando el servicio en ese momento) para que puedan utilizarla de manera adecuada. Podemos llamar a ese elemento **gestor de contexto**.

Una conclusión que se extrae del análisis de los casos de uso correspondientes a ejecución de un servicio es que es un requisito indispensable que la interfaz con la que los servicios se comunican con el usuario sea lo más sencilla e intuitiva posible.

Desde el punto de vista de la **búsqueda y descubrimiento de servicios** el usuario debe interactuar con una interfaz de búsqueda mediante la introducción de términos o requisitos, selección de temáticas u otros mecanismos. El agente de búsqueda (que filtra los servicios encontrados) se encarga de seleccionar los servicios más adecuados según la información proveniente del gestor de contexto, que incluye el perfil, el entorno actual, elecciones anteriores, etc.

En cuanto a la **interoperabilidad de servicios** se establecen relaciones dinámicas entre servicios en tiempo de ejecución, por las que se intercambian contenidos. Debe existir, por tanto, la posibilidad de que el usuario decida crear un servicio que consuma los contenidos que ofrece un servicio que está ejecutándose en ese momento por lo que dentro del contexto, se debe tener especialmente en cuenta qué instancias de servicio se están ejecutando en este momento, ya que es muy probable que el usuario quiera utilizar su información dentro del servicio nuevo.

Por último, relacionado con la **armonización de capacidades**, debe entenderse este proceso como la búsqueda de una capacidad adecuada a las necesidades de un servicio en tiempo de ejecución. Además de profundizar en el concepto de que un servicio está compuesto de una combinación de componentes, que gestionan el acceso a capacidades, se debe hacer hincapié en la importancia de que la manera en la que un componente consigue llevar a cabo la funcionalidad para el que está definido sea transparente para el usuario.

El usuario, para crear su servicio, accede a un repositorio de componentes y busca un componente que cumpla una serie de funcionalidades. En tiempo de ejecución el armonizador de capacidades busca dinámicamente las capacidades asociadas a un componente de acuerdo con su contexto.

4.2 Requisitos del usuario en movilidad

Los "requisitos de usuario" determinan las necesidades en el diseño e implementación del sistema y serán una pieza clave para que el entorno responda a las necesidades reales de los usuarios que pretenden crear y utilizar servicios en el futuro entorno inteligente. A continuación se describen los requisitos funcionales y no funcionales definidos para este tipo de entorno en el proyecto mIO! [13], que estudia, define y desarrolla tecnologías para prestar servicios en movilidad en el futuro universo inteligente

4.2.1 Requisitos no funcionales

Los requisitos no funcionales son el conjunto de funciones o especificaciones adicionales que aseguran que se disponga de un sistema manejable y gestionable que ofrezca la funcionalidad requerida de manera fiable, ininterrumpida o con el tiempo mínimo de interrupción, incluso ante situaciones inusuales.

- **Facilidad de uso:** Se asume que el sistema va a ser utilizado por usuarios no expertos. Tanto las interfaces como los mecanismos de interacción deben ser sencillos de usar.

- **Diseño para todos:** Las interfaces de usuario deberán adecuarse a los criterios del diseño para todos, así como a los estándares de accesibilidad.

- **Disponibilidad:** El sistema debe garantizar una disponibilidad total. Un fallo en el mismo va a causar enormes gastos tanto a los consumidores como a los proveedores.

- **Robustez:** El sistema debe ser capaz de responder con rapidez y eficacia a los fallos del sistema para minimizar el *downtime.*

- **Interfaz adaptable:** La interfaz de usuario deberá poder adaptarse a los diferentes dispositivos dónde puede ser visualizada, así como a los diferentes perfiles de usuario. Es deseable que la interfaz de usuario sea gráfica para que puede ser asimilada por éste más fácilmente.

- **Control de acceso:** El sistema debe proveer un método de autenticación mediante el cual sólo los usuarios autorizados podrán acceder al mismo.

- **Gestión de permisos:** Los usuarios deben tener una lista de permisos asociada que les habiliten para realizar operaciones de distinto nivel dentro del sistema, como buscar usuarios o utilizar los servicios creados por otros.

- **No repudio:** El sistema debe garantizar que todas las transacciones que van a ser realizadas son seguras. Ninguna entidad podrá repudiar una operación realizada.

- **Confidencialidad de los datos:** Toda la información enviada y recibida por los actores deberá estar cifrada, de forma que se garantice la privacidad.

- **Integridad de los datos:** El sistema debe garantizar la integridad tanto de los datos como de la información intercambiada.

- **Ubicuidad:** El sistema debe ser inter-operable desde cualquier punto dónde la plataforma esté presente. A través del servicio integral de localización, se podrá acceder a la información del contexto.

- **Utilización como capacidad:** Los servicios pueden disponer de mecanismos que le permitan poder ser utilizados como capacidades. Esos mecanismos pueden proporcionarse integrados en el propio servicio o como un módulo complementario y se ofrecerán al exterior a través de una interfaz.

- **Eficacia:** El sistema deberá ofrecer a los usuarios que lo utilicen las herramientas y los servicios que hayan solicitado y no otros, con la mayor precisión posible.

- **Inteligencia:** El sistema deberá "comprender" las intenciones del usuario y proporcionarle las herramientas y servicios que éstos deseen.

Pensando en la definición del entorno a partir de estos requisitos, el análisis de éstos permite definir un sistema con las siguientes características:

- Necesidad de incorporar una interfaz para la ejecución y creación de servicios de aspecto gráfico, que pueda ser utilizada de manera sencilla por usuarios no expertos y que pueda adaptarse a las distintas formas de visualización.

- La interfaz de usuario debe mostrar un sistema inteligente y eficaz, interoperable desde cualquier punto.

- La plataforma debe destacar por su seguridad, muy importante para obtener la confianza de los usuarios, gestionando los permisos de control de acceso y garantizado la confidencialidad de los datos.

- El sistema debe ser robusto, estar disponible en cualquier momento y garantizar la integridad tanto de los datos como de la información intercambiada.

4.2.2 Requisitos funcionales

Los requisitos funcionales podemos definirlos como el conjunto de operaciones que deben soportarse.

En general son características requeridas del sistema que expresan una capacidad de acción del mismo, una funcionalidad; generalmente expresada en una declaración en forma verbal.

Los requisitos funcionales de la plataforma pueden ser agrupados en los siguientes campos:

1. **Localización**
    a) Encontrar usuarios. El sistema va a ser capaz de localizar a cualquier usuario conectado a la red de la plataforma. Si un usuario pregunta por la posición de otro, el sistema debe ser capaz de responder con la información solicitada.
    b) Mostrar posición. El sistema debe ser capaz de mostrar la posición actual de cualquier usuario si se cumplen los permisos pertinentes.
2. **Gestión de usuarios**
    a) Añadir usuario. El sistema debe ser capaz de dar de alta nuevos usuarios. La información relevante así como el rol del usuario deberá ser introducida

en este proceso para limitar las funciones a las que dicho usuario puede acceder.

b) Eliminar usuario. Cuando un usuario no va a volver a utilizar nunca más la plataforma, deberá ser dado de baja del sistema. También se deberá borrar la información personal del mismo.

c) Listar usuarios. Por razones de gestión, es muy útil poder disponer de una lista completa de los usuarios del sistema.

d) Gestión de perfiles. El sistema deberá poder gestionar los perfiles de los usuarios, acceder a ellos y utilizarlos para ofrecerles la solución que mejor se adapte a sus necesidades.

3. Gestión de servicios

a) Añadir servicios. El sistema debe permitir la introducción de nuevos servicios en la plataforma. Tanto la descripción del servicio como los datos relevantes al mismo, deberán ser proporcionados para permitir el acceso al mismo.

b) Eliminar servicios. Cuando un servicio no vaya a ser utilizado deberá ser eliminado del sistema.

c) Listar servicios. Se debe garantizar el acceso a una lista con todos los servicios disponibles en un determinado momento.

d) Modificar servicios. El proveedor de servicios podrá modificar o actualizar los servicios que oferta.

e) Encontrar servicios. Un buen sistema de búsqueda de servicios debe ser introducido a la plataforma. Además, se deben poder buscar servicios de diferentes formas.

f) Acceder a servicios. Los servicios ofrecidos por el sistema deben facilitar su acceso a través de una interfaz sencilla.

4. Gestión de servicios adaptados

a) Crear servicios adaptados. El sistema debe ser capaz de permitir a los usuarios crear sus propios servicios, a través de la composición de los que ya se ofertan en la plataforma.

b) Borrar servicios adaptados. Cuando un servicio adaptado no vaya a ser utilizado deberá ser eliminado del sistema.

c) Modificar servicios adaptados. Debido a la naturaleza de la plataforma (nuevos servicios disponibles, servicios dados de baja), el usuario debe poder modificar la composición de su servicio adaptado.

d) Consumir servicios adaptados. Los servicios adaptados ofrecidos por el sistema deben facilitar su acceso a través de una interfaz sencilla.

5. Gestión de contenidos

a) Añadir nuevos contenidos. El sistema debe permitir añadir nuevos contenidos para mejorar la calidad de un servicio. Además del contenido en sí, se pueden introducir otros datos útiles como pueden ser permisos de acceso, número de identificación, etc.

b) Eliminar contenidos. El sistema debe permitir el borrado de contenidos obsoletos de cara a mantener la información actualizada.

c) Listar contenidos. El sistema debe ser capaz de listar los contenidos de un determinado servicio, si el usuario tiene permiso para ello.

d) Encontrar contenidos. El sistema debe ser capaz de buscar contenidos dentro de un servicio. De esta forma, el usuario podrá acceder a un determinado contenido, sin necesidad de conocer todo los demás.

e) Acceder a contenidos. El usuario debe poder acceder a los contenidos de un servicio.

f) Modificar contenidos. La posibilidad de modificar un contenido debe ser proporcionada por el sistema, de cara a actualizar los contenidos obsoletos.

6. Monitorización de la plataforma

a) Monitorizar la eficiencia del sistema. El sistema debe proveer los mecanismos necesarios para garantizar la eficiencia.

b) Gestionar los parámetros del sistema. Los parámetros que van a ser monitorizados deben representarse con sus valores límite. Si en algún momento un valor sobrepasa un límite, se deberán accionar los sistemas de notificación.

c) Gestionar las alarmas. El sistema debe implementar un sistema de notificación que permita avisar y responder ante cualquier fallo.

d) Activar acciones de prevención. Si la eficiencia del sistema se está viendo mermada debido algún problema capaz de ser monitorizado, se deben desencadenar las acciones pertinentes para minimizarla.

7. Gestión de la seguridad

a) Configurar servicios de auditoría. El administrador del sistema deberá poder establecer servicios de auditoría para controlar las políticas de seguridad que están siendo aplicadas.

b) Acceder a los datos de auditoría. El sistema permitirá que el administrador acceda a las incidencias generadas por los servicios de auditoría.

c) Configurar el control de acceso. El administrador debe poder establecer grupos de usuarios, perfiles, permisos específicos y mapas de recursos.

d) Configurar los servicios criptográficos. El administrador debe poder establecer el despliegue y la selección de los mecanismos de cifrado.

8. Gestión de los proveedores de servicios

a) Añadir proveedor de servicio. El sistema debe ser capaz de dar de alta nuevos proveedores de servicio. De esta forma, podrán ofrecer sus servicios.

b) Eliminar proveedor de servicio. Cuando un proveedor de servicios no va a volver a ofrecer un servicio, deberá ser dado de baja del sistema.

c) Listar proveedores de servicio. Por razones de gestión, es muy útil poder disponer de una lista completa de los proveedores de servicio.

d) Modificar proveedores de servicio. El sistema debe permitir al administrador modificar los datos del proveedor de servicios.

4.3 Definición de servicio en el contexto del nuevo Universo Inteligente

Definimos un servicio del Universo Inteligente como una entidad software sensible al contexto con interfaz de usuario que se ejecuta en un servidor residente en el móvil o en la red, y proporciona valor al usuario. Además, estos servicios tienen una serie de características:

- Están destinado para su utilización por una persona y no una máquina. Por tanto tienen que presentar una interfaz que permita su interacción con el usuario.
- Son creados y gestionados por los propios usuarios, lo que permite que cada uno pueda crear el servicio específico que necesita para cubrir una necesidad personal o una necesidad genérica para otros usuarios.
- Se proveen desde terminales móviles, para permitir que sea el propio usuario el que ofrezca los servicios, convirtiéndose no solo en consumidor, sino en proveedor de servicios; y que además pueda hacerlo esté donde esté a través de su terminal móvil.

4.3.1 Elementos de un servicio

Dentro de la plataforma diseñada en este trabajo, un servicio puede manifestarse de tres formas: plantilla de servicio, instancia de servicio y sesión de servicio, de acuerdo a la siguiente Figura.

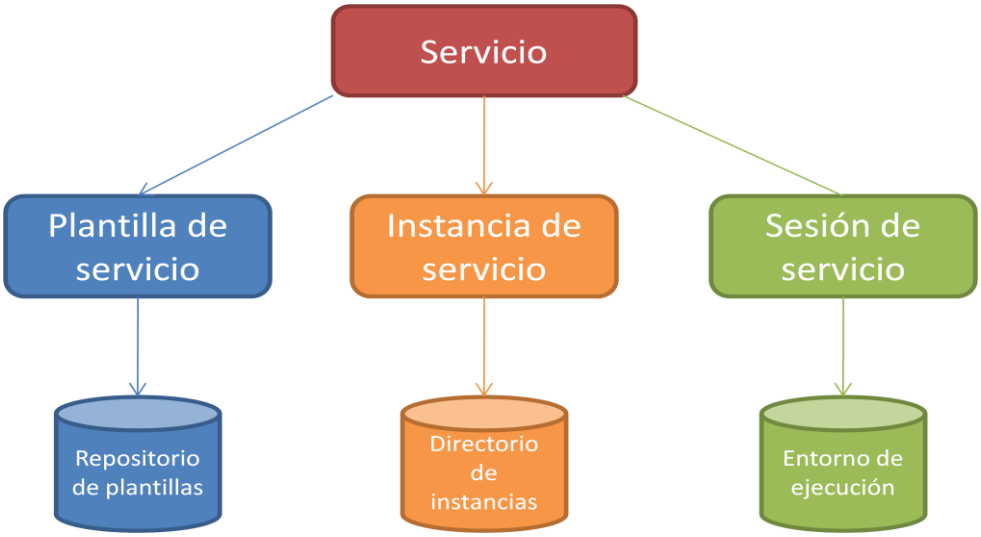

Figura 13. Elementos de un servicio

La primera manifestación de un servicio es la **plantilla de servicio**. Se trata de un documento que contiene toda la información relativa al servicio, necesaria para llevar a cabo todas las operaciones que se van a realizar con él a lo largo de su ciclo de vida dentro de la plataforma. Esta información incluye, por ejemplo:

- La lógica del servicio, es decir, la parte ejecutable del mismo, que será utilizada por el entorno de ejecución.
- Metadatos, que serán utilizados para la clasificación del servicio y su posterior descubrimiento.
- Información de seguridad, con listas de control de acceso, etc.
- Descripción gráfica, que describe el aspecto que tiene el servicio en el editor gráfico de servicios.
- Contenidos estáticos
- Etc.

Esta plantilla se almacena en la plataforma, en el/los repositorios de plantillas, para que pueda ser accedida por los diferentes elementos de la misma, y los usuarios, siempre que se necesite cualquier información relativa a dicho servicio.

Una vez que un usuario encuentra una plantilla de un servicio que quiere ofrecer, tiene que personalizarla si el creador ha introducido parámetros configurables. La plantilla puede especificar un servicio de forma genérica, como por ejemplo "servicio que ofrece la localización de un usuario en Google Maps", pero el usuario A, que quiere ofrecer ese servicio, tiene que introducir información de configuración en esa plantilla para que ofrezca la localización del usuario A y no de otro. Cuando a una plantilla se añade información de configuración y se instala en un servidor, de forma que en un *endpoint* determinado se aceptan peticiones para ese servicio, esa plantilla se convierte en una **instancia de servicio**: una plantilla particularizada con información de configuración y que se ofrece en un *endpoint* determinado de la plataforma mIO! Esa instancia se lista en el/los repositorios de instancias, que almacenan la información de configuración, el endpoint en el que se puede acceder, etc.

Eso significa que, mientras que la plantilla de un servicio es única, las instancias pueden ser múltiples. El usuario A y el usuario B pueden tener cada uno una instancia diferente del mismo servicio de localización. La instancia del usuario A ofrecerá la localización del usuario A y la instancia del usuario B ofrecerá la localización del usuario B.

Una instancia representa por tanto un servicio que ya se está ofreciendo. Para cada una de las peticiones de servicio que llegan para esa instancia de cada uno de sus

clientes, se genera una **sesión de servicio**: un hilo de ejecución asignado a un cliente determinado que está corriendo dentro del entorno de ejecución. Si un servicio tiene asociada una y sólo una plantilla, y una plantilla puede tener asociada varias instancias, una instancia puede tener asociada varias sesiones, una por cada cliente que esté consumiendo un servicio en un momento dado. Si los usuarios C y D son amigos del usuario A y consumen su instancia de servicio de localización, y el usuario E es amigo del usuario B y consume su instancia de servicio de localización, habrá dos sesiones de la instancia del usuario A (una para el usuario C y otra para el usuario D) y solo una de la instancia del usuario B (para el usuario E).

Cabe señalar que la introducción de la información de configuración puede venir impuesta desde la fase de creación de la plantilla a través de parámetros que hacen referencia al contexto del proveedor, del cliente o de otros usuarios. En el ejemplo del servicio de localización, en tiempo de diseño podría especificarse que la localización sea un parámetro que se obtenga del contexto del proveedor.

4.3.2 Actores y roles de un servicio

Desde el punto de vista del entono *prosumer* los diferentes actores de un servicio son en todo caso **usuarios** (servicios en movilidad prestados por usuarios y consumidos por usuario).

Por este motivo, en este apartado se plantean los diferentes roles que un usuario puede desempeñar en el acceso a servicios en este entorno. En este sentido, se estudian dos vertientes: el **usuario como consumidor de servicios** y el **usuario como proveedor de servicios.**

Cuando el usuario actúa como mero consumidor de servicios, ya sean éstos prestados por parte de otros usuarios o de entidades, su comportamiento será el mismo, por lo que éste será el rol único que desempeñará: rol consumidor .

Dada la estructura jerárquica de roles, se mantiene no obstante abierta la posibilidad de definir niveles inferiores en cuanto a roles de usuario como consumidor de servicios, aunque en este primer análisis no se han detectado necesidades para dicha diferenciación.

Desde el punto de vista de entorno *prosumer*, un usuario del sistema puede crear y proveer servicios que serán consumidos por parte de otros usuarios. En base a la estructura jerárquica de roles contemplada, existirá un **rol proveedor**, pero se detectan asimismo diferentes roles asociados que el usuario puede desempeñar cuando actúa como proveedor de servicios.

- Un usuario es considerado creador de un servicio cuando decide crear la lógica de servicio y su plantilla, subiendo esta al repositorio central. El usuario, por tanto, no ha instanciado el servicio, tan solo ha creado una plantilla que ha subido al repositorio. En este caso, un usuario jugará el **rol creador** cuando se disponga a crear un servicio.

- Un usuario actúa como aprovisionador cuando está ofreciendo el servicio desde su terminal (ha instanciado una plantilla de servicio y está ejecutándolo). En este caso, un usuario tendrá activo su **rol aprovisionador.** Dado que un usuario puede descargarse plantillas creadas por otros, es posible ser aprovisionador de un servicio sin ser su creador.

- Un usuario proveedor puede actuar como gestor o manager cuando puede configurar parámetros de un servicio que decide ofrecer. En este caso, un usuario tendrá activo su **rol gestor** cuando configure parámetros concretos de los servicios que vaya a proveer. También sería gestor ya que decidirá en todo momento si quiere para la ejecución del servicio o controlar el acceso a su servicio por parte de otros usuarios.

La estructura jerárquica de roles propuesta para cubrir las necesidades detectadas en el acceso a servicios en el entorno *prosumer* sería la siguiente:

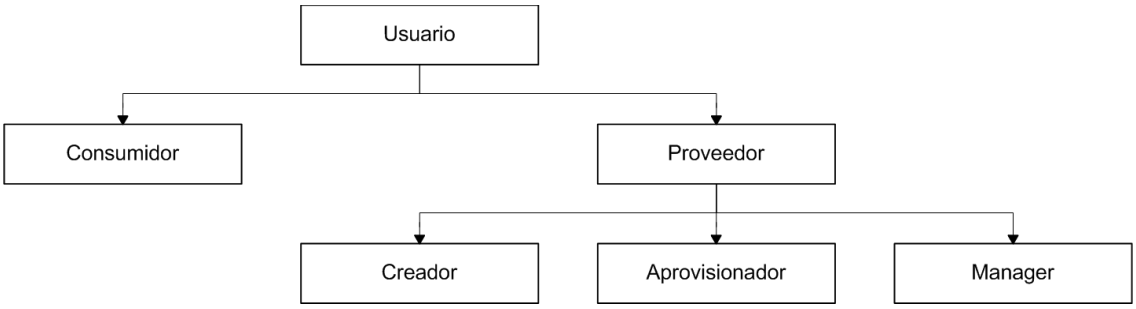

Figura 14. Estructura jerárquica de roles para el entorno *prosumer*

4.3.3 Estructura lógica de un servicio

Podemos definir un servicio de muchas formas, dependiendo del estado del ciclo de vida en el que se encuentra. La estructura lógica de un servicio ha sido dividida en tres niveles que corresponden con el nivel de usuario, el nivel de creación y en nivel de ejecución. En la siguiente figura puede verse un ejemplo de servicio (Sport Tracker) creado por un usuario que pretende visualizar en un mapa su localización e información sobre su pulso cardiaco cuando sale a correr. Siguiendo el paradigma del universo inteligente el usuario debe tener un sistema para crear servicios de forma fácil desde su móvil donde, con sólo agregar los componentes de Localización, Mapa y Pulso cardiaco y algunas opciones de configuración, este servicio pueda ser generado automáticamente por la plataforma. En tiempo de ejecución el servicio accederá a las capacidades disponibles en el entorno que le permitirán resolver estos componentes de

la mejor manera posible. Una vez que el servicio está creado el usuario puede publicarlo para que otras personas lo puedan ejecutar y puedan beneficiarse de la información que proporciona.

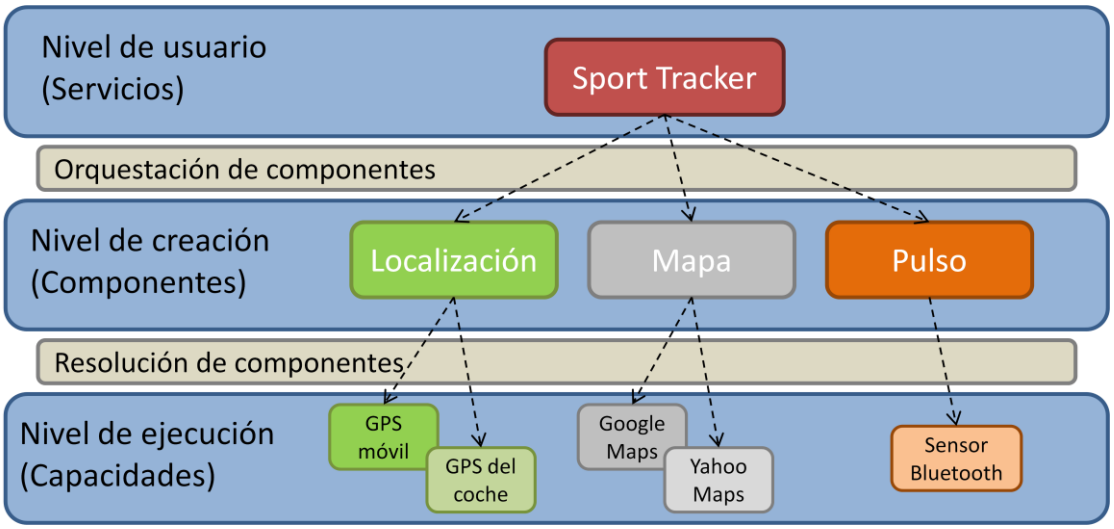

Figura 15. Estructura lógica del servicio Sport Tracker

Desde el punto de vista del usuario (nivel de usuario) el servicio puede verse como una estructura monolítica donde los componentes permanecen ocultos. Esta visión corresponde a un modelo de caja negra [12] donde el usuario que consume el servicio puede permanecer ajeno a la funcionalidad interna del mismo mientras conozca el resultado de ejecutarlo. El elemento más importante en este nivel es la interfaz gráfica de servicio, que contiene la lógica necesaria para procesar las acciones del usuario y mostrar una respuesta apropiada.

Continuando con el nivel de creación el servicio puede dividirse en unidades lógicas denominadas componentes, los cuales están interconectados mediante conectores y una lógica que describe la funcionalidad del servicio. Definimos un **componente** como la **unidad básica funcional de un servicio**. Es una **abstracción de alto nivel que se implementa con una capacidad encontrada**, dependiendo de las condiciones de ejecución de servicio. Los componentes son utilizados por el sistema para hacer un proceso de creación más sencillo y para proporcionar el nivel de abstracción adecuado para que un usuario pueda comprender su funcionamiento pero los detalles de implementación permanezcan ocultos. Es importante recordar que esta propuesta tiene como restricciones las condiciones impuestas por el universo inteligente en el que un usuario no experto puede crear servicios de un forma sencilla.

Llamamos **orquestación de componentes al proceso por el cual se define un flujo resultado de la combinación de diversos componentes**. Este flujo es en general controlado por una entidad externa a los componentes que determina el orden de ejecución de los mismos y realiza las correspondientes invocaciones, al contrario que el

proceso de coreografía, en el que los componentes son conscientes del lugar que ocupan dentro del flujo e invocan al siguiente componente en orden, una vez que han terminado de ofrecer su funcionalidad.

Desde el punto de vista del nivel de ejecución, un servicio es un conjunto de capacidades que ofrecen las funcionalidades que el servicio pretende cumplir. Una **capacidad** es la **implementación de un componente utilizado en el entorno de creación**. Las capacidades se encuentran en repositorios locales, cercanos o remotos y pueden ser dispositivos como una impresora, un monitor o recursos como un servidor web, etc. La división entre los modelos de componente y capacidad se ha realizado por dos razones:

El empleo de componentes, más allá de la importancia que tienen en la reutilización de código, supone una enorme ventaja para el usuario en el proceso de creación ya que una generación de servicios basada en el empleo de workflows de componentes es una buena alternativa para un usuario no experto a la hora de crear servicios simples.

Las capacidades ocultan características particulares del elemento al que están accediendo proporcionando un middleware con métodos conocidos que son compatibles con la gestión de componentes. De esta forma, **un componente puede ser resuelto con diferentes capacidades dependiendo de las condiciones de ejecución del servicio y de las preferencias proporcionadas por el usuario en el proceso de creación**. Como las capacidades proporcionan un conjunto común de métodos para acceder a los recursos, los componentes pueden utilizar los nombres de los métodos desacoplando así la funcionalidad proporcionada de la implementación particular de esta funcionalidad.

Llamamos **resolución** al **proceso por el cual una capacidad se asigna a un componente en tiempo de ejecución**. Según lo que se ha comentado sobre componentes y capacidades se necesita identificar un componente con una capacidad que sepa implementarlo. El proceso de decisión para encontrar la capacidad óptima para un componente dado se denomina proceso de armonización de capacidades. Encontrar la capacidad óptima depende de múltiples factores, por ejemplo, de las opciones de configuración que el usuario ha establecido en tiempo de creación y que se convertirán en restricciones utilizadas en el proceso de armonización de capacidades para seleccionar la capacidad apropiada. En la siguiente sección se describe con detalle este proceso y cómo se transforman las preferencias de usuario a restricciones. En la Figura 15, el componente del servicio Sport Tracker, denominado “Pulso cardiaco”, siempre accede a la capacidad “sensor Bluetooth” mientras que el componente de “Mapa” puede ser resuelto por distintos proveedores de mapas (Google o Yahoo) dependiendo de preferencias de usuario o restricciones en el servicio.

4.4 Conclusiones

En esta sección se ha expuesto un ejemplo de escenario de entre los analizados en [8] y [9] que describe las características que se quieren introducir en el diseño de la plataforma de provisión de servicios en movilidad. A través de las conclusiones del análisis se han generado una serie de requisitos de usuario que determinan las necesidades en el diseño e implementación del sistema.

Estos requisitos se han dividido en funcionales (conjunto de operaciones que deben soportarse) y no funcionales (conjunto de funciones o especificaciones adicionales), y contribuyen a realizar una definición de servicio más precisa.

Finalmente se ha definido un **servicio en el universo inteligente como una entidad software con interfaz de usuario que se ejecuta en el móvil o en la red y que está gestionada y es utilizada por los propios usuarios de la plataforma, desde la cual es provista**. Tras describir los elementos de un servicio y los actores que lo utilizan se perfila su estructura lógica con tres niveles: Nivel de usuario, en el que el servicio se representa como una unidad monolítica, nivel de creación, en el que el servicio está compuesto por un conjunto de componentes interrelacionados y por último el nivel de ejecución donde se define el concepto de capacidad, como implementación funcional de un componente.

5 Definición de arquitectura genérica para la provisión de servicios

5.1 Metodología seguida

En el marco de la definición del entorno se ha seguido una metodología para alcanzar la definición y la implementación de una plataforma de provisión de servicios basada en fases, que comentamos a continuación:

El primer paso que se realizó fue materializar el **concepto de Universo Inteligente**, definiendo sus características, sus ventajas con respecto a los paradigmas actuales y el cambio de enfoque, situando al usuario en el centro del entorno, y a su terminal móvil, como su puerta de enlace a todas las capacidades que lo rodean y que pueden ser utilizadas por los servicios existentes. Estos servicios son creados por los propios usuarios y, por supuesto, consumidos por estos.

El segundo paso fue realizar una revisión del **estado del arte** en tecnologías para prestar servicios en movilidad. Concretamente, se han analizado los lenguajes de descripción de servicios, las nuevas arquitecturas de servicios, herramientas para el desarrollo móvil, los mecanismos de búsqueda, descubrimiento, publicación y registro de servicios que se consideran con mayor impacto y, por último, las herramientas para creación de servicios en terminales móviles.

El tercer paso fue generar unos escenarios que reflejaran el comportamiento del usuario en este nuevo Universo Inteligente. De entre los escenarios generados se ha analizado en este trabajo el escenario de "Creación y Uso de Sport Tracker", en el que se identifican situaciones sencillas de utilización de la plataforma de creación y provisión de servicios.

Posteriormente se extrajeron los requisitos y restricciones de este caso de uso para definir formalmente los conceptos utilizados (servicio, componente, capacidad, plantilla, instancia) y los módulos y sistemas que intervienen en los distintos procesos (creación, publicación, búsqueda y descubrimiento, ejecución, resolución e interoperabilidad).

En esta sección se diseña la arquitectura de este sistema a partir de los requisitos generados en las secciones previas, descomponiendo las funcionalidades descritas en los escenarios en bloques atómicos que realicen una función básica.

5.2 Ciclo de vida de un servicio

Se define *ciclo de vida de un servicio* como la colección de estados en los que éste se puede encontrar dentro de la plataforma, desde que se crea hasta que se destruye y desaparece, y a las operaciones que se pueden llevar a cabo con él en cada uno de esos estados, y que pueden provocar la transición hacia otro estado diferente.

A grandes rasgos, el ciclo de vida de un servicio puede verse esquematizado en la Figura 16.

El servicio comienza su existencia en manos del creador del servicio. Éste utiliza las herramientas de la plataforma para crear su servicio y publicar la plantilla correspondiente.

A continuación, un proveedor de servicio descubre dicha plantilla y decide que quiere proveerla. Para ello, la configura y la instancia, desplegándola y publicándola en la arquitectura del sistema.

Finalmente, un usuario descubre dicha instancia y decide utilizarla, convirtiéndose en consumidor de la misma, para lo cual se crea una sesión dentro del entorno de ejecución correspondiente.

Figura 16. Ciclo de vida de un servicio

El proceso de borrado o eliminación puede llevarse a cabo en diferentes niveles. Un consumidor de servicio podrá gestionar y eliminar sus sesiones; un proveedor, sus instancias; y finalmente, un creador, sus plantillas.

A lo largo de todo el ciclo de vida se lleva a cabo un proceso transversal de recogida de *feedback* por parte del operador de la plataforma. La información relevante de cada uno de los pasos es almacenada en el perfil de cada uno de los usuarios y lo enriquece. Esto aumentará el conocimiento que de cada usuario tiene la plataforma, facilitando procesos como la recomendación o la personalización.

A continuación se definen en detalle cada uno de los pasos del ciclo de vida de un servicio.

5.2.1 Creación de un servicio

Se trata de la primera etapa del ciclo de vida de cualquier servicio, que determina el inicio de su existencia dentro de la plataforma. Para empezar, un usuario determina que tiene que crear un servicio nuevo, probablemente porque no ha encontrado ningún servicio que satisfaga la necesidad concreta que tiene en un momento dado.

La creación de un servicio comienza cuando el usuario que asume el rol de creador del servicio decide abrir el editor de servicios. Existen tres puntos de partida posibles para iniciar la creación de un servicio:

- Creación desde cero: El creador inicializa una plantilla completamente vacía.
- Creación a partir de otra plantilla: El creador utiliza otra plantilla ya existente y va modificando sus características para dar lugar a un nuevo servicio diferente del primero. Para ello, el creador ha tenido que llevar un proceso de búsqueda de plantillas y descubrir aquélla sobre la que quiere trabajar.
- Creación a partir de patrones: La plataforma ofrece una serie de "recetas" habituales para la creación de servicios, en forma de guías paso a paso utilizando un asistente, conocidas como patrones. Un creador de servicio puede utilizar este asistente para crear un servicio siguiendo los pasos de un patrón determinado.

De cualquier manera, utilizando el editor de servicios, el creador puede definir la lógica e introducir todos los datos que son necesarios para constituir el servicio.

Con la información introducida por el usuario y otros datos que se encuentran directamente en posesión de la plataforma (como la fecha o el identificador del usuario creador y otros datos sensibles), una serie de herramientas automatizadas generarán la plantilla del servicio. Por ejemplo, un traductor podría convertir la definición gráfica de la lógica en un *script* o programa ejecutable, o un razonador semántico utilizar toda la información del servicio para categorizarlo adecuadamente o construir su interfaz. En todo caso, como resultado se genera la plantilla del servicio con todos los campos obligatorios necesarios para su correcto funcionamiento dentro de la plataforma.

Antes de pasar a la fase de publicación, el creador del servicio puede querer probarlo. La plataforma debería ofrecer herramientas que permitan que el creador esté seguro de que el servicio recién diseñado funciona como él desea y que la plataforma compruebe que el servicio no es dañino o malicioso. Estas herramientas pueden incluir un depurador que permita ver la ejecución simulada de un servicio paso a paso; y/o un entorno sintético tipo "caja de arena" [58] que permita la ejecución del servicio sobre recursos simulados; y/o una herramienta de comprobación inteligente que detecte errores en la lógica (como bucles infinitos) y código malicioso.

Este paso de prueba resulta clave tanto en la plataforma como en cualquier entorno abierto en el que usuarios no expertos u hostiles tengan derechos de creación, y es necesario para proteger a los usuarios de sí mismos (por ejemplo, para que no se publiquen servicios con bucles infinitos que envíen SMSs con el consiguiente gasto monetario), o de otros (como en el caso de un usuario malicioso que cree un servicio para recopilar datos sensibles de los demás).

5.2.2 Publicación de plantillas

El objetivo de la etapa de publicación de plantillas es que la plantilla que acaba de ser generada esté disponible a través de la plataforma para su descubrimiento por usuarios potenciales.

Para ello, el editor de servicios que acaba de generar la plantilla tiene primero que descubrir el lugar más adecuado para su publicación, el/los repositorios de plantillas dentro de la arquitectura, y a continuación realizar la transferencia de los datos, dando además los valores adecuados para su indexación y/o categorización dentro de dicho repositorio/s. Estos repositorios podrán estar total o parcialmente categorizados y semantizados para facilitar su descubrimiento y la búsqueda de los servicios que contienen.

5.2.3 Publicación de instancias y despliegue

En esta etapa del ciclo de vida del servicio entra en escena el proveedor del servicio, que será un usuario que desee ofrecer un servicio (y en el caso de servicios de tipo local, también consumirlo) particularizándolo para sus necesidades.

Lo primero que el proveedor tiene que hacer es descubrir la plantilla de un servicio que desee ofrecer. Para ello tendrá que utilizar las herramientas de descubrimiento y publicidad de la plataforma.

Para que esto sea posible, será necesario llevar a cabo dos operaciones bien diferenciadas:

- *Configuración*: Por medio de un asistente, el proveedor del servicio introduce toda la información necesaria para personalizar el servicio que desea ofrecer, si es necesario. Por ejemplo, si una asociación de comerciantes pone a disposición de sus miembros un servicio de acceso remoto a los catálogos, el dueño de una tienda en concreto tiene que configurar dicho servicio para que acceda a la base de datos de su catálogo y no al de ningún otro.

- *Instanciación y despliegue*: Es el proceso por el cual se genera la instancia del servicio, se inicializa el servidor que va a escuchar peticiones de servicio para dicha instancia y se publica en un repositorio de instancias para que pueda ser descubierta por sus potenciales usuarios. Para ello, primero se almacena en el servidor de la instancia toda la información de configuración introducida en el paso anterior, y a continuación se configura el mismo para que escuche peticiones de servicio. Finalmente, se publica la instancia dentro de la arquitectura, incluyendo la información necesaria para su correcto descubrimiento y la dirección del *endpoint* en la que se encuentra, de forma que los clientes que deseen utilizarla sean capaces de encontrarla.

El resultado de todo este proceso es la instancia, un servicio que se está ofreciendo en un *endpoint* determinado y que mantiene de forma conjunta la información sobre la plantilla en la que se basa y los datos de configuración particulares para esta instancia.

5.2.4 Búsqueda, Recomendación y Publicidad de servicios

Para permitir un archivado y una categorización dinámica de los servicios, la arquitectura contempla que la información sobre los mismos, ya sean instancias o plantillas, esté recogida en una red de repositorios. Para que un usuario adquiera conocimiento sobre la existencia de un determinado servicio primero tiene que *descubrir* el repositorio adecuado y a continuación encontrar un servicio determinado dentro de dicho repositorio.

Para no complicar esta tarea de forma adicional, el **descubrimiento de repositorios** se lleva a cabo de manera automática. El terminal debe gestionar una serie de procesos de preguntas y escuchas para descubrir los repositorios, sus direcciones y cómo comunicarse con ellos. Además, es necesario que la red de repositorios esté coordinada adecuadamente de forma que una búsqueda pueda ser redirigida hasta que encuentre el repositorio más adecuado.

Para encontrar un servicio se contemplan tres procesos diferentes: búsqueda, recomendación y publicidad, dependiendo de quién desempeñe el rol activo:

- Búsqueda: En este procedimiento, es el usuario el que desempeña el rol activo. Es él el que ejecuta una búsqueda introduciendo los parámetros de su interés, que servirán para discriminar los servicios útiles de entre toda la paleta disponible, en un buscador.

- Recomendación: En este procedimiento es la propia plataforma la que, aprovechando el conocimiento que tiene del usuario y agentes inteligentes que analicen dicho conocimiento, confecciona una búsqueda personalizada que arroje los resultados que más se ajusten a un perfil o a un contexto determinado, recibiendo el usuario una lista de los mismos, quizás incluso ordenada según sus preferencias.

- Publicidad: En este procedimiento es la persona que publica el servicio la que decide lanzar una notificación a una lista de usuarios objetivo potencialmente interesados. Dichos usuarios objetivo reciben entonces un mensaje informándoles de la existencia y características del nuevo servicio disponible.

Como ya se ha visto, el descubrimiento de repositorios y la búsqueda de servicios puede llevarse a cabo en cuatro niveles:

- Nivel de plantillas: Se puede utilizar para dos propósitos. Primero, para que los potenciales creadores de servicios busquen plantillas ya existentes para modificarlas y crear servicios nuevos; y segundo, para que los potenciales proveedores de servicios busquen plantillas que puedan configurar como instancias propias.

- Nivel de componentes: Usado para que los usuarios creadores de servicios accedan a los repositorios de componentes externos y puedan incluirlos en el diseño del servicio en tiempo de creación.

- Nivel de instancias: Se utiliza para que los potenciales usuarios busquen las instancias que desean consumir.

- Nivel de capacidades: Usado en tiempo de ejecución para determinar las capacidades disponibles en las que se mapean los componentes.

Cabe destacar que los métodos y herramientas de búsqueda, recomendación y publicidad pueden ser los mismos para plantillas, componentes e instancias, pero rastreando en cada caso los repositorios adecuados, es decir, de plantillas, componentes e instancias respectivamente.

5.2.5 Ejecución de servicios

En esta etapa se efectúan las acciones de que consta el servicio, acciones que se realizan tanto en el consumidor como en el proveedor. Es decir, que la ejecución tiene lugar en ambas partes (aunque en el caso de un servicio local, el usuario es a la vez proveedor y cliente). Esta etapa se inicia cuando un usuario ha descubierto una instancia de servicio que desea consumir y lanza una petición al *endpoint* adecuado, puesto que en la información obtenida durante el proceso de descubrimiento se le ha informado de la dirección de dicho *endpoint*.

El servidor gestiona dicha petición y en ese momento se genera una *sesión de servicio* dentro de un entorno de ejecución, que en principio podría estar localizado en

cualquier parte de la arquitectura, incluyendo la propia red, el terminal servidor o el terminal cliente. En cualquiera de los casos, la información de la lógica de servicio obtenida a partir de su platilla se carga en dicho entorno de ejecución, junto con todos los datos sensibles necesarios para su correcto funcionamiento, y específicamente el identificador del cliente. En este momento debe producirse una resolución de las dependencias necesarias para que el servicio funciones. Si alguna dependencia no está disponible se buscará en el terminal del usuario cliente o en la red y si, por algún motivo, no se encuentra el arranque del servicio se detendrá.

Durante la ejecución el servicio lleva a cabo la funcionalidad para la que ha sido diseñado. Además, a lo largo del proceso de ejecución se efectúan también diferentes operaciones de tarificación, dependiendo del modelo de tarificación aplicable en cada caso (que puede depender del operador o del dueño del servicio o de la instancia). La cantidad de modelos aplicables es muy grande y depende del operador, que puede imponer un modelo para todos los servicios, o del dueño del servicio y la instancia, si el operador permite elegir uno diferente para cada servicio. Es posible imponer costes a los proveedores por ofrecer servicios o a los clientes por usarlos. Es posible que los beneficios vayan a parar al operador, al creador del servicio o a su proveedor, o que se repartan entre los tres. Es posible que se efectúe tarificación a nivel de sesión de servicio, cargando costes por cada sesión iniciada, o más detallados, dependiendo de los eventos que sucedan en cada sesión y los recursos consumidos. Por ejemplo, se podría cobrar cada vez que se inicie una sesión de servicio, sea éste el que sea, o que la ejecución no esté tarificada, pero cada vez que dicho servicio haga uso de un recurso (una videollamada, un acceso a una base de datos), se cargue dicho acceso.

En definitiva, las posibilidades son casi infinitas, y el modelo empleado finalmente dependerá del modelo de negocio del operador y, en su caso, del creador y proveedor del servicio.

En todo caso, durante la etapa de ejecución se efectuará también una monitorización adecuada que permita llevar a cabo una correcta tarificación.

5.2.6 Gestión de servicios

Tanto los creadores de servicios sobre sus plantillas, como los proveedores sobre sus instancias y los usuarios sobre sus sesiones, tendrán derechos de gestión. Esto implica que pueden llevar a cabo dos operaciones: modificación y eliminación.

En todo caso, la modificación/borrado de una sesión solo atañe a su usuario. Sin embargo, la modificación/borrado de una instancia afectará también a las sesiones que dependen de ella. Y la modificación/borrado de una plantilla, podría, al menos potencialmente, afectar a sus instancias y a las sesiones de estas instancias.

Es por tanto necesario que la infraestructura disponga de mecanismos para garantizar la coherencia de plantillas, instancias y sesiones. Las aproximaciones a este problema son múltiples, y dependen de factores de diseño de bajo nivel. Una opción sencilla para implementar la eliminación, aunque no muy adecuada desde el punto de vista del usuario, sería que el eliminar una plantilla purgue el sistema de todas las instancias y sesiones dependientes, forzando la terminación de todas las que estén activas. Sin embargo, también es posible que en el mismo caso, aunque las instancias se eliminen, las sesiones activas acaben su ejecución. Si se considera en cambio que un creador no tiene derecho a eliminar las instancias que dependen de sus plantillas, el procedimiento de eliminación de éstas se complica, pues habría que llevar a cabo un control de coherencia, como por ejemplo, marcar la plantilla como "en eliminación" mientras que todas las instancias que de ella dependen finalizan, y al mismo tiempo, llevar a cabo un proceso de "recogida de basura" para asegurarse que ninguna de las instancias que permanecen activas durante mucho tiempo hayan sido simplemente olvidadas ahí.

El caso de la modificación es aún más complicado. La aproximación más sencilla es que la modificación de una plantilla o instancia sea equivalente a la eliminación de la versión antigua y la creación de una nueva. Sin embargo en ese caso será imposible relacionar ambas versiones, y los usuarios de la versión antigua no tendrán conocimiento automático sobre la nueva versión.

En el lado opuesto del espectro se encontraría un sistema de "actualización en caliente" (*hot updating*), que permitiera una transición suave entre dos versiones. Una forma de implementar esta funcionalidad sería permitir que las sesiones de la versión antigua siguieran existiendo hasta que terminaran de forma natural. Esto implicaría mantener de alguna manera una duplicidad de los datos (el sistema tendría que conservar la plantilla antigua para referencia de las sesiones antiguas y la nueva plantilla para las nuevas sesiones que se vayan iniciando) y disponer de un sistema de "recogida de basura" que identifique el momento en el que la versión antigua ha dejado de usarse y la elimine definitivamente. Otra forma de implementar esta "actualización caliente" sería intentar transformar las sesiones de la versión antigua en sesiones de la versión nueva, copiando de forma inteligente los estados y los datos de cada una de ellas y adaptándolos a la nueva versión, pero resulta evidente que esto necesita de grandes dosis de inteligencia en el sistema y que no será posible en todos los casos.

La conclusión es que la problemática generada por esta etapa del servicio es complicada, pero en general la plataforma tiene que disponer de herramientas adecuadas para garantizar la coherencia de las versiones de los servicios.

Desde el punto de vista de los usuarios, es necesario también ofrecer interfaces adecuados para que todos los actores de la plataforma puedan gestionar los elementos de los que son dueños. Esto incluye controles para modificar y eliminar dichos elementos, así como herramientas de monitorización:

- Los usuarios querrán consultar la información de las sesiones que se están ejecutando en cada momento y poder detenerlas en consecuencia.

- Los proveedores querrán ver información sobre los usuarios que están consumiendo una instancia en un momento dado, y poder invocar de nuevo a la interfaz de configuración para poder modificar dicha instancia, o directamente eliminarla.

- Los creadores querrán consultar qué instancias se corresponden con cada servicio, y poder invocar al entorno de creación para cambiar parámetros de la plantilla o la propia lógica del servicio, y también, por supuesto, eliminar completamente un servicio.

Y en todos los casos, también debe ser posible consultar la información de tarificación.

5.2.7 Interoperación de servicios

Como ya se ha especificado al comienzo del apartado 4.3 Definición de servicio en el contexto del nuevo Universo Inteligente , una de las características fundamentales de un servicio es que está destinado al consumo humano. Sin embargo, para facilitar la composición de servicios más complicados, la plataforma desarrollada puede permitir que un servicio haga uso de otros que ya existen previamente.

Para ello, durante la ejecución de un servicio pueden lanzarse invocaciones a otros. En todo caso, la etapa de interoperación de servicios implicará una comunicación entre servicios a través del entorno de ejecución y de interoperabilidad que ha de ser transparente al usuario.

5.3 Identificación de elementos funcionales del entorno

La arquitectura funcional diseñada en este trabajo se compone de distintos sistemas que interactúan entre sí (creación, ejecución, búsqueda y interoperabilidad) y de los actores que interviene en esta plataforma).

La siguiente figura propone una arquitectura general para esta plataforma:

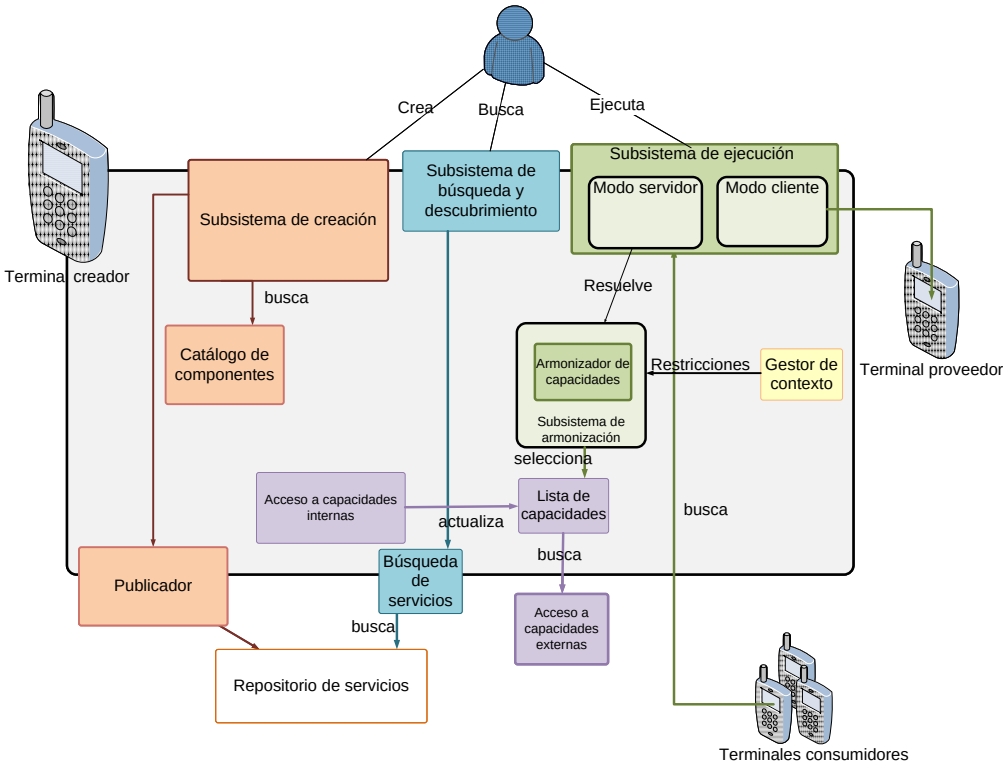

Figura 17. Arquitectura genérica del entorno

Subsistema de creación: proporciona una interfaz para el usuario no experto que resulta fácil de utilizar, permitiéndole crear sus propios servicios utilizando componentes como elementos atómicos del proceso de creación.

Subsistema de publicación: una vez que el usuario ha creado sus servicios necesitará publicarlos en repositorios para que otros usuarios los encuentren.

Subsistema de ejecución: el objetivo de este sistema es servir como entorno de ejecución de los servicios. Puede funcionar en modo cliente, servidor o ambos.

Subsistema de búsqueda y descubrimiento: incluye un conjunto de herramientas que facilitan que los usuarios puedan encontrar cualquier servicio publicado (búsqueda) y permite al entorno móvil seleccionar de forma automática el servicio más apropiado para el usuario, basado en preferencias e información contextual (descubrimiento).

Subsistema de armonización: Mientras que el componente es la unidad atómica en el proceso de creación, el término capacidad se relaciona con la implementación de la funcionalidad de este componente. Los componentes manejados por el *prosumer* deben ser fáciles de comprender y utilizar y pueden ser utilizados para la creación de múltiples servicios. Las capacidades identifican recursos que serán utilizados por el componente para cumplir su propósito. En este subsistema se produce la elección de la capacidad más apropiada para el componente que la requiera. Por ejemplo, si el usuario quiere crear un servicio para ver su localización en un mapa, en el subsistema

de armonización se buscará la capacidad de localización más apropiada, por ejemplo, la capacidad GPS interna del móvil o un dispositivo GPS Bluetooth que el usuario lleva conectado.

5.4 Desarrollo tecnológico del trabajo

Durante el diseño de la arquitectura del sistema hemos identificado algunos problemas tecnológicos que describimos a continuación incluyendo la solución hallada para cada caso.

El primer problema que se ha tratado es cómo hacer que la **interacción entre subsistemas** funcione. Los servicios creados deben ser publicados antes de que sean requeridos por otros usuarios y ejecutados en el entorno de ejecución. Por eso, mecanismos como la publicación, que mantiene el ciclo de vida del servicio, han sido integrados en la plataforma.

Para que exista una interacción entre subsistemas es necesario definir un Lenguaje de Descripción de Servicios (SDL) que describa al servicio a lo largo de su ciclo de vida. Definimos el SDL como un conjunto de reglas gramaticales y sintácticas que permiten la descripción de los diferentes aspectos de un servicio, como la funcionalidad que proporciona, aspectos de seguridad, autenticación, capacidades no-funcionales e incluso las posibles interacciones con otros servicios. Para el diseño del SDL hemos tenido en cuenta que la plataforma se integra en dispositivos móviles por lo que se necesita un SDL ligero y no muy complejo.

Existen muchos lenguajes estándar que pueden utilizarse como BPEL [26], que emerge como la solución general para la composición de servicios Web en los procesos de negocio. BPEL requiere una interfaz de descripción bastante rígida [59] en la que se deben proporcionar datos como los nombres para los tipos de puerto, nombres de operación para servicios Web, etc. Además ni el estándar BPEL ni las actuales extensiones soportan características de reutilización [60]. En otras palabras, necesitamos un mecanismo más modular y granular para definir y describir fragmentos reutilizables de modelos de proceso de negocio.

La iniciativa OWL-S [61] se centra en la descripción semántica de servicios Web y en la automatización de un conjunto de tareas como el descubrimiento de servicios Web y su composición. Desde un punto de vista técnico OWL-S constituye una buena alternativa a considerar gracias a su enfoque directo. Sin embargo existen algunos puntos y responsabilidades que no están claros desde el punto de vista de los modelos de negocio, como la falta de monitorización y el manejo de errores [62]. Otros aspectos como la escalabilidad de la tecnología y la necesidad de intervención manual para la

gestión de servicios Web convierte a la tecnología OWL-S en una opción no muy viable para entorno de ejecución en tiempo real.

Teniendo en cuenta las ventajas de BPEL y de OWL-S se decidió diseñar un lenguaje SDL propio siguiendo las siguientes premisas:

- El documento SDL está dividido en distintas vistas, que contienen la información proporcionada por todos los subsistemas.

- El SDL está basado en XML (eXtensive Markup Language) [18]. La utilización de este lenguaje proporciona modularidad y mecanismos estándar para modificar el documento.

- Cada vista SDL se define utilizando distintos lenguajes basados en XML, por lo que será posible encontrar una vista semántica en OWL o RDF, una vista de ejecución descrita en alguna variante de BPEL o SCA [34] o una vista de presentación expresada en XHTML y JavaScript.

- Este SDL actúa como mecanismo de enlace entre los distintos subsistemas de la plataforma y contendrá requisitos especificados en uno de los subsistemas que actuarán como restricciones en otro de ellos (por ejemplo los requisitos impuestos en creación pueden ser decisivos en el proceso de armonización de capacidades).

Otro problema que ha sido considerado es **como gestionar los componentes que se utilizan para construir un servicio compuesto**. Los componentes son arrastrados en el entorno de creación y son interconectados. En el entorno de ejecución estos componentes son procesados y ejecutados siguiendo las conexiones definidas, las cuales establecen el flujo de ejecución. Para producir una interacción entre componentes sencilla ha sido necesario proporcionar mecanismos de gestión de componentes que permitan que se intercambien datos entre componentes conectados. En esta sección se analiza una tecnología que hace posible esta gestión:

La tecnología OSGi [42]**¡Error! No se encuentra el origen de la referencia.** es decuada en esta tarea. Sin embargo, esta tecnología, que permite implementar arquitecturas basadas en componentes no es adecuada para gestionar componentes que no existían previamente en la plataforma [63] [64]. Se necesita extender la plataforma OSGi para que gestione dependencias dinámicas más allá del modelo de servicios declarativos (DS), derivado del proyecto Service Binder [65] [66].

La especificación DS permite a los desarrolladores especificar dependencias de componentes en un archivo XML. Nuestra propuesta incluye un nuevo bundle que gestiona las vinculaciones de cada uno de los bundles que componen un servicio. Este nuevo bundle asigna una instancia de un gestor de bundles (bundleManager) a cada componente en su proceso de arranque para que este gestor resuelva sus dependencias

en tiempo de ejecución utilizando peticiones de servicio OSGi. La pérdida de una vinculación (producida por un desregistro de servicio, fallo en el proceso de negociación/adaptación, etc) produce un evento que es capturado por el gestor de componentes, el cual intenta encontrar otro proveedor de servicio o, si la búsqueda no resulta fructífera, desactivar el componente.

Otro elemento que ha sido resuelto es el **intercambio de datos entre los componentes**. Los componentes utilizados para crear servicios son piezas de código compiladas que pueden verse en el entorno de creación como cajas negras, con entradas, salidas y algunos parámetros de configuración. El tipo de datos intercambiado entre componentes se describe en el documento SDL del componente con la particularidad de que, en algunas conexiones, la información aceptada por un componente a su entrada no corresponde con la información que otro componente envía. Para resolver este problema se pueden utilizar dos enfoques distintos.

Una solución radica en la utilización de mecanismos de adaptación entre componentes. La adaptación se define como la transformación del formato de los datos de salida de un componente para que se ajuste al formato aceptado a la entrada de otro componente. Para que la adaptación se lleve a cabo se necesita un componente adaptador para cada punto de comunicación de los componentes que se sitúe en medio de los componentes a adaptar. El principal inconveniente de esta propuesta es que se necesita un gran catálogo de adaptadores que sean capaces de transformar el formato de salida de un componente a otros muchos formatos de entrada.

Otra solución es definir un proceso de negociación entre componentes. En este caso un componente contempla más de un tipo de datos en sus puntos de comunicación. El bundle especial que vincula una instancia de *bundleManager* a cada bundle en su proceso de arranque pasa a denominarse ***mediador***. Cuando el usuario intenta conectar dos componentes, el mediador permite que esos componentes interactúen y que determinen cual es el mejor formato y tipo de dato que debe ser intercambiado.

El proceso de negociación introduce las siguientes ventajas:

- No son necesarios nuevos componentes que actúen como adaptadores sino que las transformaciones están integradas en la implementación del componente.

- La adaptación podría considerarse un mecanismo fácil para realizar conexiones pero al no existir un proceso de negociación entre los datos intercambiados (codificación, resolución, bitrate) en el proceso de adaptación quizás este mecanismo no sea el más adecuado desde el punto de vista de la flexibilidad y el rendimiento desde el punto de vista de la ejecución.

El último elemento que se ha definido es el mecanismo utilizado para **desacoplar la implementación del componente con el mecanismo de acceso a la capacidad**.

Considerando un recurso que proporciona una funcionalidad que ha sido representada como una capacidad dentro de la plataforma el primer paso es realizar un envoltorio que rodee el acceso al recurso de una manera uniforme para que los componentes puedan acceder a todos los recursos de la misma forma. Por ejemplo, si se desea utilizar la funcionalidad GPS del teléfono móvil para esta plataforma el recurso GPS se envuelve como una capacidad genérica de GPS para que pueda ser utilizado como una capacidad estándar por el módulo de gestión de capacidades. Internamente, la envoltura realiza una conversión del lenguaje y tipo de datos utilizado en la plataforma (para tratar capacidades genéricas) al formato utilizado por el sistema operativo del terminal móvil para acceder físicamente al módulo GPS. Al tratar cada capacidad utilizando la misma sintaxis es posible lograr la armonización de las mismas mientras el usuario está en movilidad y las capacidades que lo rodean cambian constantemente.

La siguiente figura muestra el proceso de resolución de capacidades que hemos definido, realizado por el subsistema de resolución:

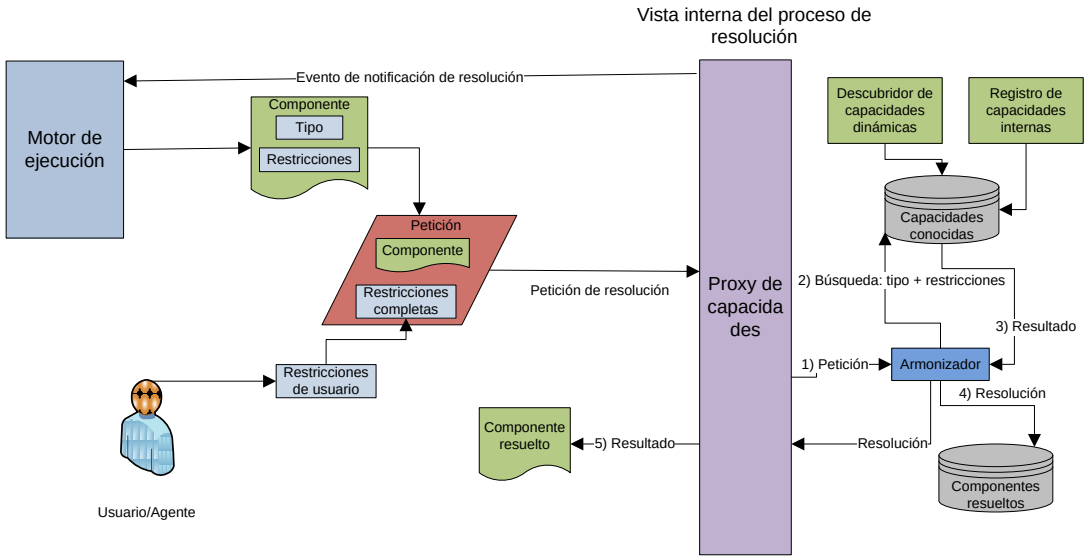

Figura 18. Proceso de resolución de capacidades

Para llevar a cabo la resolución de componentes se necesita realizar una petición al proxy de capacidades, proporcionando información como el componente a resolver y las restricciones impuestas por el usuario (1). El modulo de armonización de capacidades inicia una búsqueda por distintos repositorios conocidos por la plataforma (2). Como resultado de la búsqueda se seleccionan un conjunto de capacidades (3).

Dentro de este conjunto de capacidades, compatibles con las restricciones proporcionadas, la elección de la capacidad óptima se delega al usuario o a un agente inteligente que toma este tipo de decisiones por el usuario (4). Cuando el componente ha sido resuelto se puede proceder a su ejecución como parte del proceso de ejecución.

Como puede verse, el subsistema de armonización de capacidades funciona como una plataforma middleware orientada a componentes que logra el desacoplamiento entre lógica de proceso y la implementación funcional de un componente dado.

5.5 Evaluación y validación

La sección de introducción presentaba los problemas de la creación y el consumo de servicios en el marco del universo inteligente. Esta propuesta intenta definir los mecanismos de creación, ejecución y armonización para que un usuario, sin conocimientos de programación y con una experiencia limitada en informática, pueda diseñar, implementar y, por supuesto, consumir sus servicios siguiendo la filosofía del usuario *prosumer*.

Estos mecanismos han sido analizados para ser fáciles de implementar, utilizando una plataforma de despliegue de servicios que considera al componente como la unidad funcional de la plataforma y participa en los procesos de creación, publicación, búsqueda y ejecución. Según lo revisado, la tecnología OSGi supone una buena candidata para la implementación del sistema, manteniendo el modelo de arquitectura basado en componentes definido en este trabajo.

La utilización de una arquitectura basada en componentes proporciona una ventaja clara en el entorno de creación. Las posibilidades de creación para una usuario no experto crecen de considerablemente cuando manejan componentes que tiene una funcionalidad fácil de entender como provisión de localización y mapas, temperatura, transmisión de video, impresión, etc. Además, tratar directamente con servicios completos no ofrece la flexibilidad y las posibilidades de configuración que pueden lograrse con la inclusión de patrones de combinación de componentes y asistentes de creación de servicios. Estos mecanismos están separados de la lógica de servicio y pueden ser provistos por la interfaz gráfica del subsistema de creación, sin afectar al funcionamiento del motor de creación.

En un nivel inferior se establecen relaciones y correspondencias entre componentes a través de conectores, de forma que, cuando un usuario enlaza dos componentes se realiza automáticamente una comprobación de compatibilidad que determina si los componentes enlazados son capaces de interactuar. Se produce, para ello, una

negociación entre las entradas y las salidas de los componentes y el resultado de esa negociación será un acuerdo sobre el tipo de datos intercambiados entre esos dos componentes [67]. Este acuerdo queda reflejado en un contrato localizado en un documento y referenciado desde el SDL del servicio. En el subsistema de ejecución existe un elemento que controla que los componentes siguen el contrato firmado.

Toda la información generada durante el proceso de creación (componentes utilizados, personalización de los mismos, conexiones entre componentes, contratos, etc) se almacena en el documento SDL que se utiliza para que la interacción entre subsistemas funcione (ver siguiente Figura). Esta información estará constituida por un conjunto de referencias a la colección de contratos, las propiedades de los componentes y los bundles OSGi. Estos tres elementos son publicados junto con el documento SDL para constituir un servicio completo.

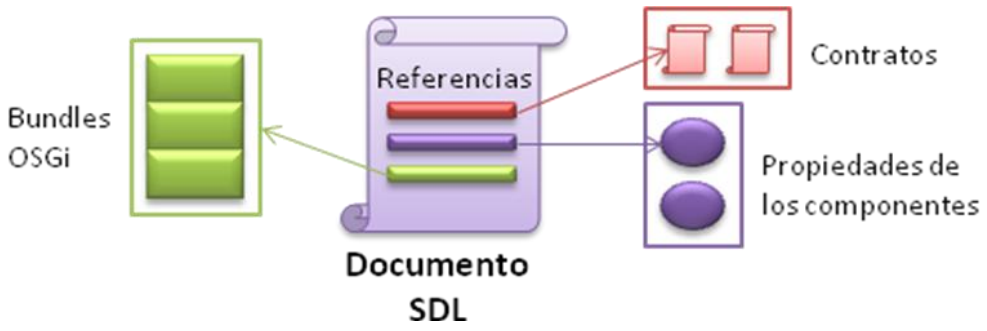

Figura 19. Proceso de resolución de capacidades

En el subsistema de ejecución se proporciona un motor que interpreta el documento SDL y configura los componentes OSGi. A este subsistema se le confía también la tarea de transmitir al subsistema de armonización de capacidades las propiedades de los componentes que actúan como restricciones para que en el proceso de resolución se selecciones la capacidad óptima para cada componente. El gestor de capacidades proporciona acceso uniforme a las capacidades locales (situadas en el dispositivo móvil), próximas (en el entorno cercano) y remotas.

A modo de validación de la solución propuesta en esta sección se ha desarrollado un activo experimental que, siguiendo el escenario de creación del servicio "Sport Tracker" estudiado anteriormente, incorpora una interfaz para el entorno de ejecución a través de la cual un usuario puede elegir el documento SDL a ejecutar y el entorno comprueba que el SDL se ajusta al SDL Schema definido y actúa como elemento orquestador de la ejecución del servicio basado en la interrelación de componentes.

Las siguientes figuras muestran la interfaz gráfica del entorno de ejecución:

Figura 20. Activo experimental de ejecución de servicios

6 Desarrollo de una arquitectura avanzada para la composición, provisión y consumo de servicios en movilidad

A partir de la arquitectura descrita en el apartado anterior se desarrollan los principales subsistemas, creando unidades más complejas con tareas más globales que simplificarán la definición del entorno y el entendimiento de su modo de proceder.

El desarrollo de esta arquitectura está basado en el proyecto mIO! [13], que estudia, define y desarrolla tecnologías para prestar servicios en movilidad en el futuro universo inteligente. El grupo de investigación al que pertenece el alumno ha liderado la tarea del diseño de la arquitectura para una plataforma de composición, publicación y consumo de servicios en movilidad, y, en especial, ha profundizado en el desarrollo del subsistema de creación. Por ello, y a modo de validación de la arquitectura propuesta el alumno ha realizado la implementación de un activo experimental de un entorno de creación para el caso de uso analizado en este trabajo en la sección 4.1.1: Sport Tracker.

6.1 Arquitectura avanzada del entorno

En el esquema aparecen los componentes que definen la arquitectura coloreados según la funcionalidad que ofrecen.

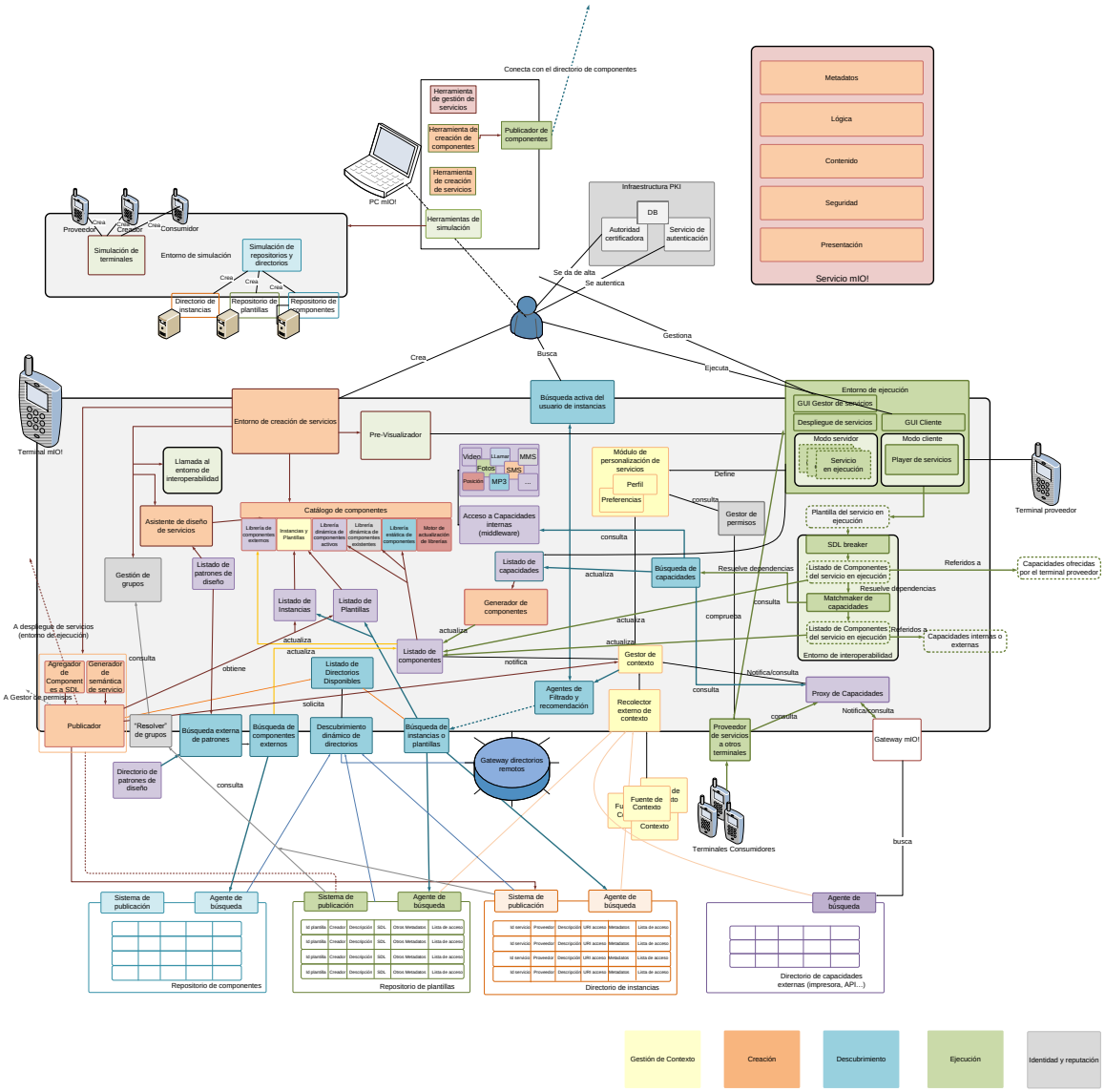

Figura 21. Arquitectura avanzada de entorno

Además de la infraestructura móvil, se ha tenido en consideración la posibilidad de que el usuario quiera crear servicios avanzados, utilizando alguna herramienta de creación y composición de servicios sustancialmente más avanzada, que incorpore características de simulación de terminales y también de simulación de repositorios de plantillas y componentes y también de directorios de instancias.

6.2 Identificación y descripción de elementos funcionales del entorno

6.2.1 Entorno PC

Se realizará una descripción de los elementos funcionales existentes dentro del entorno PC de mIO!

Entorno de simulación completo: Se contempla un entorno mIO! en PC, que incluirá un simulador completo del sistema, incluyendo terminales consumidores, terminales proveedores, repositorios y directorios, todos ellos simulados, para que el usuario pueda crear servicios y probar cómo se comportarán estos servicios en relación a todas las entidades del sistema, antes de la publicación.

Herramientas para la gestión de servicios: Permite la incorporación de herramientas más sofisticadas para la gestión de servicios que las que podemos tener en un terminal móvil. Informarán sobre estadísticas de consumo de recursos, utilización de los servicios además de proporcionar una interfaz para permitir la actualización, parada, rearranque, instalación y desinstalación de servicios.

Herramientas para la creación de componentes: Uno de los elementos fundamentales en la arquitectura mIO! del entorno PC es la inclusión de herramientas que permitan programar componentes. Estos componentes pueden estar relacionados con alguna capacidad como por ejemplo un componente que devuelva una imagen de un mapa al introducir una dirección (debe conectarse con la API de Google Maps, que, desde el entorno mIO! es vista como una capacidad) o pueden ser componentes sin ninguna capacidad asociada como codificadores (MP3, JPG, MPEG), sistemas de transmisión (video, audio), implementaciones de protocolos, que aunque no constituyan servicios completos, deben integrarse como parte de los servicios a crear.

Publicador de componentes: Conecta con los repositorios de componentes y envía los componentes creados mediante las herramientas de creación para que otros usuarios puedan incorporarlas a sus servicios en sus entornos de creación.

6.2.2 Entorno móvil

Clasificaremos los elementos según si pertenecen a los mecanismos de gestión de contexto, creación, descubrimiento, ejecución o identidad y reputación.

6.2.2.1 Elementos de gestión de contexto (pertenecientes a otras actividades del proyecto mIO!)

Módulo de personalización de servicios: Este módulo se compone de un elemento que gestiona el perfil del usuario y otro elemento que gestiona las preferencias tomando decisiones sobre la información almacenada en ellos. En general la función de

este módulo es la de proporcionar al entorno de ejecución información sobre el perfil y las preferencias del usuario indicando las acciones a personalizar del servicio y de la propia ejecución. Además es consultado por el gestor de permisos para extraer las preferencias de ejecución.

Perfil: Almacena la información del perfil (restricción de la definición de un sujeto que juega un determinado rol en unas determinadas circunstancias) del usuario e incorpora mecanismos para gestionar ese perfil, como la modificación y lectura de los datos, permisos de acceso al perfil, etc.

Gestor de contexto: Contiene toda aquella información relevante que sirve para caracterizar la situación de un usuario final con el fin de seleccionar los servicios más apropiados a dicha situación, así como adecuar su funcionalidad.

Recolector externo de contexto: Recoge de forma dinámica la información de contexto procedente de las fuentes de contexto y la envía al módulo gestor de contexto.

Fuente de contexto: Entidad externa al terminal capaz de proporcionar información de contexto al sistema local. Puede ser otro terminal móvil, entidades software y otros dispositivos físicos y objetos del entorno.

6.2.2.2 Elementos de creación

Entorno de creación de servicios: Constituye el medio donde tendrán lugar los mecanismos de creación de servicios. Proporciona una interfaz gráfica muy intuitiva para que el usuario pueda acceder a los elementos de creación adicionales. El resultado de la invocación al entorno de creación es un conjunto de componentes conectados de forma ordenada que constituirán la plantilla del servicio. El entorno de creación estará conectado con los elementos explicados a continuación.

Asistente de diseño de servicios: Asistente que ayudará al usuario en la tarea de creación de servicios. Sugerirá posibilidades e intentará anticiparse a los deseos del creador. Este asistente interpretará patrones de diseño para guiar al usuario a la hora de crear servicios intentando analizar qué patrón es el más adecuado para implementar un servicio determinado.

Listado de patrones de diseño: Los patrones de diseño serán proporcionados por el asistente de diseño de servicios el entorno de creación y consistirán en "recetas" para crear servicios comunes. Se intentará que la gran mayoría de los servicios que deseen crear los usuarios estén basados en patrones de diseño.

Búsqueda externa de patrones: Elemento de búsqueda pero muy relacionado con la creación de servicios. Este módulo conectará con el directorio externo de patrones de diseño y actualizará el listado de patrones que reside en el terminal.

Pre-Visualizador: Módulo que permite la función de visualizar el comportamiento de un servicio antes de publicarlo. Esta opción estará disponible dentro del entorno de creación de servicios.

Agregador de componentes a SDL: Módulo que se encarga de generar un SDL conjunto del servicio creado a partir de los SDLs de los componentes utilizados y de otro tipo de información expresada en metadatos.

Generador de semántica de servicio: Nutre al servicio creado de información semántica útil para su publicación y descubrimiento.

Publicador: El módulo publicador contacta con los sistemas de publicación del repositorio de plantillas y el directorio de instancias para incluir en ellos las plantillas y las referencias a las instancias que se desea publicar. El publicador se conecta al módulo de despliegue de servicios y le proporciona la plantilla del servicio a publicar para que en el módulo de despliegue instancie esa plantilla y le devuelva la dirección del *endpoint* donde se encuentra la instancia de forma que el publicador pueda incluir esa información en el directorio de instancias para que los clientes que deseen utilizarla sean capaces de encontrarla.

En este módulo se utilizan la información de contexto y la semántica de servicio junto con la información proporcionada por el usuario en tiempo de publicación (definición de la lista de acceso al servicio, restricciones de acceso, etc.) para añadir información al servicio a publicar.

El contexto actual del usuario publicador debe ser tenido en cuenta a la hora de filtrar las búsquedas de otros usuarios hacia ese servicio. La adición de esta información contextual en la publicación debe ser opcional, a pesar de que esto pueda complicar el creador, ya que en algunas ocasiones tendrá sentido añadir esta información y en otras no (unos servicios son *context-aware* y otros no).

Por otro lado, en el caso de servicios impulsivos, pequeños, con contenidos muy personales y ligados al usuario, esta información contextual puede ser en ocasiones tan importante como el propio servicio.

Catálogo de componentes: Conjunto de repositorios donde se encuentran los componentes utilizables para crear un servicio, clasificados en: componentes externos, instancias y plantillas (instancias para ser utilizadas como componentes y plantillas para ser enviadas al entorno de interoperabilidad), componentes activos, dinámicos y

estáticos. También existe el motor de actualización de librerías que busca iterativamente en los listados de componentes, plantillas y estancias si existen elementos nuevos.

Generador de componentes: Utiliza el listado de capacidades internas y externas para generar componentes que puedan ser utilizados en el entorno de creación de servicios.

6.2.2.3 Elementos de descubrimiento y búsqueda

Incluye todos los módulos de búsqueda y listados:

Listado de componentes: Contiene los componentes que pueden utilizarse en el entorno de creación.

Búsqueda de componentes externos: Se conecta al directorio de componentes y actualiza el listado de componentes con los componentes externos al terminal móvil.

Listado de Instancias y Plantillas: Contienen instancias que serán utilizadas como componentes y plantillas que pueden ser enviadas al entorno de interoperabilidad para ser descompuestas en sus elementos básicos (componentes).

Búsqueda de instancias o plantillas: Se conecta a los directorios de instancias y plantillas y actualiza los listados correspondientes.

Listado de capacidades: Contiene información sobre las capacidades disponibles en el sistema y que el terminal conoce o tiene en memoria en este momento.

Acceso a capacidades internas: Este módulo controla el acceso a las capacidades que proporciona el terminal móvil del usuario como la posibilidad de realizar llamadas, realizar fotografías y reproducir archivos MP3.

Proxy de capacidades: Utiliza el *Gateway* mIO! para acceder al directorio de capacidades externas y suministrar esa información al módulo de contexto y al de búsqueda de capacidades. Actualiza el listado de capacidades actual.

Gateway mIO!: Es utilizado como puerta de enlace hacia los elementos con los que el terminal puede interactuar como marquesinas, dispositivos cercanos, otros elementos disponibles en la red del operador, etc.

Búsqueda de capacidades: Accede a los módulos de capacidades internas y al *proxy* de capacidades para actualizar la información que contiene el listado de capacidades.

Descubrimiento dinámico de directorios: Se encarga de localizar la dirección de los directorios de componentes, capacidades externas, instancias, plantillas y patrones y proporciona un sistema de conexión a éstos.

Búsqueda activa: Utilizado por el usuario para buscar de forma activa instancias de servicio para su posterior ejecución.

Agentes de filtrado y recomendación: Se basan en la información proporcionada por el contexto y los perfiles para seleccionar qué servicios se adaptan mejor a las exigencias del usuario.

Repositorios: Pueden clasificarse en repositorio de componentes, capacidades, de plantillas y directorio de instancias.

- Repositorio de componentes: Incorpora mecanismos de búsqueda y publicación de componentes para ser adquiridos por el módulo de búsqueda de componentes externos e incorporarlos al listado de componentes del terminal móvil. Estos componentes, como se ha comentado arriba, son utilidades como codificadores, sistemas de transmisión, implementaciones de protocolos, que aunque no constituyan servicios completos, deben integrarse como parte de los servicios a crear.

- Repositorio de plantillas: Incorpora mecanismos de búsqueda y publicación de plantillas de servicio. Estas plantillas son adquiridas por el módulo de búsqueda de instancias o plantillas para pasar a formar parte del listado local de plantillas y poder ser utilizadas en el entorno de creación de servicios.

- Directorio de instancias: Incorpora mecanismos de búsqueda y publicación de información referente a instancias de servicio. La instancia no se encuentra en el repositorio sino que permanece en el sistema del usuario que provee el servicio. Sin embargo este repositorio contiene información para poder localizarla y otra información adicional en forma de metadatos como la descripción del servicio, información del creador y una lista de permisos de acceso.

- Directorio de capacidades externas: Accesible mediante el *Gateway* mIO! reúne dinámicamente información sobre los elementos disponibles en un entorno cercano al usuario, o proporcionados por la red del operador.

- Directorio de patrones de diseño: Directorio externo que contiene un conjunto de patrones clasificados según la función del servicio que es mantenido por los desarrolladores del sistema mIO! añadiendo nuevos patrones que se vayan desarrollando a lo largo del periodo de utilización del sistema mIO!

Gateway de directorios remotos: Gestiona el conjunto de directorios desplegados en el entorno mIO! Aunque en el esquema de diseño del entorno aparezca un solo repositorio de instancias, por ejemplo, deben existir muchos directorios de instancias, ubicados en distintos lugares y accesibles desde distintas redes. El *Gateway* de

directorios remotos puede encargarse de gestionar los elementos presentes en los repositorios moviéndolos de ubicación o clasificándolos por temática si resulta conveniente.

6.2.2.4 Elementos de ejecución

Entorno de ejecución: El entorno de ejecución tiene dos partes bien diferenciadas para conseguir una separación (al menos lógica) entre la ejecución como proveedor y la ejecución como consumidor.

La parte del entorno que actúa en **modo servidor tiene como función** ejecutar los servicios provistos desde el terminal, procesando las peticiones recibidas mediante la lógica de servidor de dichos servicios. Para ello alberga una plataforma de despliegue de servicios provistos por el módulo "despliegue de servicios" que contiene todas las instancias publicadas por el sistema que están preparadas para empezar a ejecutarse junto con todas las instancias que ya se encuentran en ejecución.

Despliegue de servicios: Este módulo es alimentado con plantillas que transforma a instancias de servicios. En el proceso de transformación a instancia este módulo devuelve una dirección del *endpoint* en la que se encuentra la instancia de forma que el sistema publicador pueda incluir esa información en el directorio de instancias para que los clientes que deseen utilizarla sean capaces de encontrarla. Este módulo utilizará tanto la información contenida en el perfil del usuario como el acceso a capacidades internas y externas para resolver las dependencias de los componentes que formen el servicio que se está instanciando.

GUI Gestor de servicios: La llamada a este módulo se encuentra en el menú principal de la interfaz gráfica del sistema mIO! presentada al usuario. Este módulo accede al listado de instancias y ofrece una descripción detallada de los servicios que han sido publicados, los servicios ejecutándose en ese momento, consumo de memoria y procesador, así como estadísticas de utilización del servicio (última fecha de utilización, ejecuciones al día, usuarios que lo ejecutan, etc…).

Además de la información a desplegar el usuario necesita unas opciones de gestión de servicios como:

- Detener un servicio en ejecución: Debido al consumo de recursos un usuario puede decidir dejar de promover un servicio temporalmente. Con esta opción la instancia del servicio deja de estar disponible.

- Reactivar un servicio parado: Si se quiere volver a ofrecer el servicio parado.

- Eliminar un servicio: Elimina un servicio del sistema. Si está arrancado primero lo detendrá y lo despublicará.

Debemos estudiar las implicaciones de poder arrancar y detener servicios sobre la publicación, ya que, si un servicio está publicado debería poderse acceder a él.

Dentro del entorno de ejecución tenemos una parte que actúa en **modo cliente**. El módulo principal es el ***player* de servicios** que arrancará cuando un usuario quiera ejecutar un servicio cuya instancia ha buscado o quiera previsualizar en servicio que ha creado o está creando. Este módulo se encarga de realizar peticiones a los terminales proveedores de servicios para arrancar la ejecución de éstos (en los terminales proveedores).

GUI Cliente: Este módulo es accesible desde el menú principal de la interfaz gráfica del sistema mIO! presentada al usuario y es el que ejecuta el *player* de servicios.

Entorno de interoperabilidad: Este entorno es llamado en dos situaciones:

- En el entorno de ejecución: Cuando un usuario está ejecutando un servicio, este servicio puede estar combinado con alguno ya en funcionamiento. Para ello se llamará al módulo de interoperabilidad que gestione la comunicación con el servicio combinado.

- En el entorno de creación: Un usuario puede requerir algunos componentes que no se encuentran en su listado y pretender buscar la plantilla de algún servicio que le proporcione la funcionalidad buscada.

En cualquier caso, una vez adquirido el componente del servicio con el que interoperar se llamará al entorno de interoperabilidad que descompondrá el servicio en componentes y con ellos actualizará el listado de componentes del sistema local.

SDL *breaker*: Este módulo permite descomponer una plantilla de un servicio en un listado de componentes. Algunos de estos componentes controlarán capacidades ofrecidas por el terminal proveedor del servicio.

***Matchmaker* de capacidades**: Se encarga de resolver las dependencias de los componentes del servicio referidos a capacidades con capacidades reales que satisfacen las restricciones impuestas por los componentes y que se encuentran accesibles en el contexto de ejecución. Este módulo permite convertir un componente creado para satisfacer las necesidades de un usuario en otro cuya finalidad sea satisfacer las necesidades del sistema local de ejecución. Un ejemplo de esta funcionalidad viene explicado en el apartado 8.3.2 discusión de la propuesta en el párrafo de resolución de dudas sobre el entorno.

6.2.2.5 Elementos de identidad y reputación

Incluiremos dentro de este apartado los elementos que proporcionan mecanismos de seguridad en la identidad de los usuarios y la autorización para realizar distintas tareas.

Infraestructura PKI: Aunque está descrita y analizada en el apartado de discusión podemos explicar brevemente que será la infraestructura utilizada para proporcionar mecanismos de registro y autenticación en la plataforma. Como componentes básicos podemos observar una base de datos donde se guarda la información de autenticación del usuario, una autoridad certificadora que entrega certificados de clave pública a los usuarios que lo solicitan y un servicio de autenticación que proporciona *tokens* que habilitan al usuario para realizar distintas tareas dentro de la plataforma, como la publicación, búsqueda y ejecución de servicios.

Gestión de grupos: Accesible desde el entorno de creación de servicios permite gestionar los grupos entre los que se encuentra el usuario. Gracias a este módulo se pueden crear nuevos grupos y asignar usuarios a dichos grupos.

"Resolver" de grupos: Cuando se accede a la información sobre la plantilla/instancia de un servicio con intención de utilizarla en tiempo de creación/ejecución se debe resolver en la lista de acceso a ese servicio qué usuarios pertenecen al grupo definido por el usuario creador. Ésa es la función de este módulo.

Gestión de permisos: Establece los tipos de permiso de acceso a los servicios publicados en modo local. También es llamado por el módulo publicador para establecer los permisos de acceso por grupos en las plantillas e instancias a publicar. El módulo de gestión de permisos debe estar en relación directa con el de gestión de grupos ya que los permisos de acceso a plantillas e instancias vendrán definidos sobre grupos de usuarios o usuarios individuales. Dentro de los repositorios también existe la gestión de permisos de acceso por medio de listas de acceso.

Lista de acceso en repositorios: Propiedad de los servicios desplegados en repositorios de instancias y plantillas. Se establece una lista de acceso por cada servicio publicado que contiene los identificadores de los usuarios o grupos que pueden acceder al servicio.

Proveedor de identidad: Permite identificarse en el sistema. Una vez identificado se podrá acceder a los agentes de búsqueda de los repositorios de componentes, plantillas e instancias.

6.3 Relación del escenario Sport Tracker con los elementos del entorno

En este apartado se analiza el escenario descrito en la sección 4.1.1 Sport Tracker desde el punto de vista de esta nueva arquitectura.

6.3.1 Creación de servicios desde cero

En la primera parte del escenario, Raúl crea su servicio "Entrenamiento de Raúl" desde su entorno de creación.

Para ello accederá al **Catalogo de componentes** que le proveerá los componentes de "mapa, posición, velocidad y pulso cardíaco". El catalogo no dispone de ellos constantemente sino que necesita que el **Listado de componentes** le provea y éste a su vez lo hace buscándolos a través de **Búsqueda de componentes externos** que accede al **Repositorio de componentes**.

Cuando Raúl concluye la creación de su servicio deberá guardarlo. En este momento el usuario podrá testear que la composición es correcta, las uniones de componentes están bien y el servicio funciona correctamente. Para ello el **Entorno de creación de servicios** accederá al **Pre-visualizador** que mostrará una ejecución del servicio virtual.

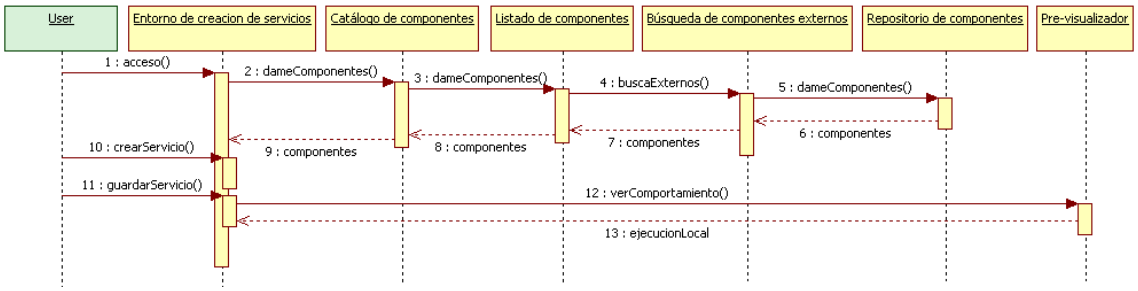

Figura 22. Creación desde cero

6.3.2 Publicación de servicios

Una vez que Raúl termina de crear su servicio lo publica para que pueda ser utilizado por otras personas.

Una vez la instancia de "Sport Tracker" está creada se enviará al bloque **Agregador de componentes a SDL** como al **Generador de semántica de servicio** que le añadirán la información necesaria para su correcto funcionamiento y búsqueda posterior y la enviarán al **Publicador**. Éste, la desplegará enviándola al **Entorno de ejecución** y éste al **Proveedor de servicio de otros terminales** que le devolverá el *endpoint* donde estará la instancia para después mandarla al **Directorio de instancias** donde se almacenará.

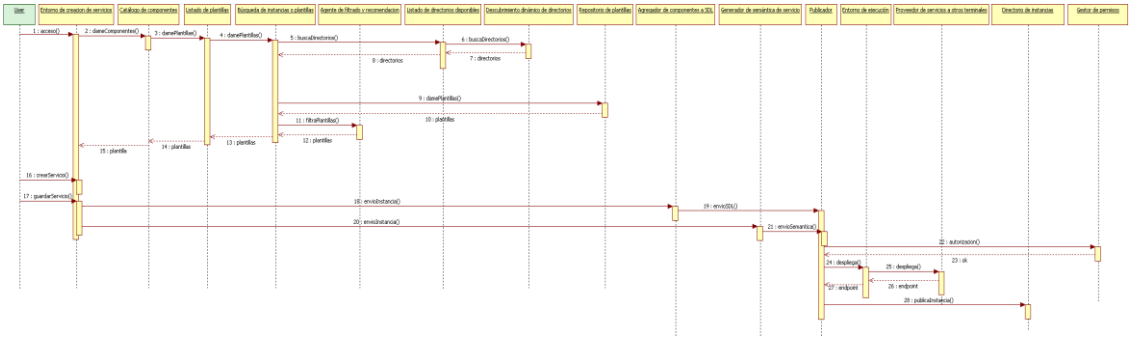

Figura 23. Publicación de instancias

6.3.3 Ejecución de servicios

Raúl decide salir a correr y ejecuta su servicio de Sport Tracker.

Para ello accederá al **Entorno de ejecución** donde lo indicará. El **Entorno de ejecución** de su terminal se comunicará con el del proveedor para obtener la lógica del servicio a través del **Proveedor de servicios a otros terminales.** Una vez el cliente dispone de toda esta lógica busca las capacidades necesarias para la correcta ejecución del servicio. Para ello el **Entorno de ejecución** accederá al **Entorno de interoperabilidad** y más concretamente al **Matchmaker de capacidades** que a través del bloque **Búsqueda de capacidades** obtendrá tanto las capacidades del propio terminal (**Acceso a capacidades internas**) como las externas al mismo (a través del ***Proxy*** **de Capacidades**).

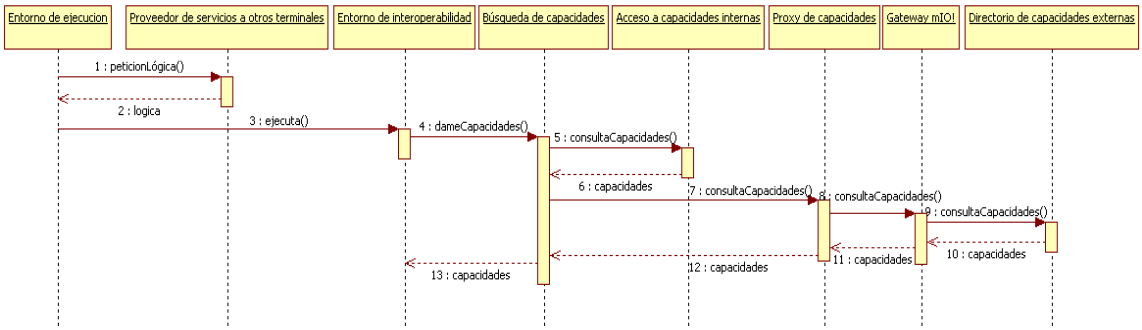

Figura 24. Ejecución de servicios

6.3.4 Búsqueda de servicios

Por último, Cristina, amiga de Raúl recibe una recomendación de ejecución del servicio publicado por Raúl, por lo que, intrigada por este nuevo servicio, decide ver sus detalles para convertirse en cliente del mismo.

Para buscar una instancia el usuario accederá al bloque **Busqueda activa del usuario de instancias**. Este bloque irá al **Agente de filtrado y recomendación** que, a través de la información del contexto y el perfil del usuario filtrará todas las instancias que ha obtenido el bloque **Búsqueda de instancias o plantillas**.

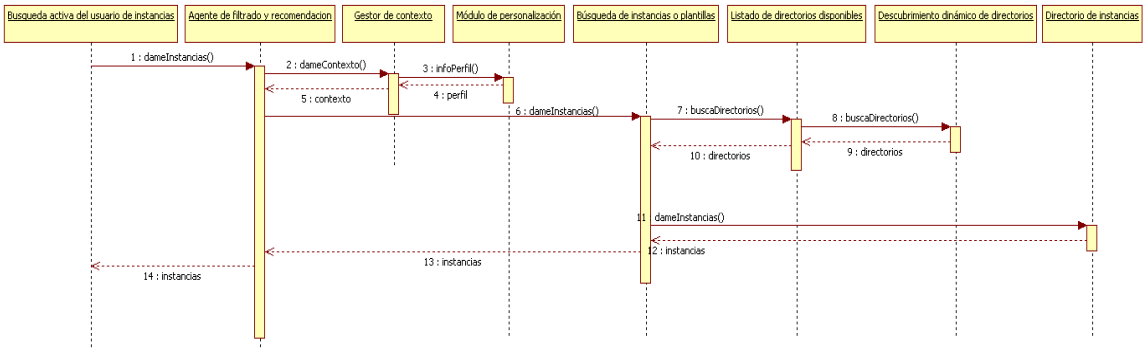

Figura 25. Búsqueda de servicios

6.4 Evaluación y validación

A modo de validación de la arquitectura propuesta se ha realizado un activo experimental con los conceptos estudiados para el sistema de creación dentro del entorno tecnológico integral para el usuario en movilidad.

En el activo se ha pretendido realizar un prototipo limitado del entorno de creación que permitiera al usuario crear un servicio equivalente al estudiado en el caso de uso Sport Tracker

El objetivo del activo es el desarrollo de un entorno de creación que permita al usuario, a través de una interfaz gráfica, generar un servicio de una forma intuitiva. Además, este servicio tiene que ser almacenable en el sistema y deben proporcionarse los mecanismos adecuados para que el entorno de ejecución pueda interpretar el resultado final del servicio para proceder a su ejecución.

Desde un punto de vista funcional el activo puede dividirse en tres partes:

- La primera parte consiste en la elaboración en un motor de generación de vista SDL de descripción, que llamaremos Motor SDLd. Este motor pertenecerá al

módulo "Entorno de creación" que aparece en la Figura 21. Arquitectura avanzada de entorno. El motor SDLd constituye el núcleo del subsistema de creación y describe el conjunto de funciones que permiten representar en memoria el árbol de componentes de un servicio, reflejando de manera precisa todas las acciones realizadas por el usuario desde la interfaz gráfica del entorno de creación. La vista SDL de descripción se define y se analiza en la sección 7.2 Modelo de vistas.

- Elaboración de un motor de generación de vista SDL interpretable, al que llamaremos Motor SDLi, que no es sino un conjunto de funciones que permiten la transformación de una plantilla (como servicio guardado por un usuario) a una instancia, mediante la creación de una nueva vista SDL llamada vista interpretable, a partir de las vistas interpretables de los distintos componentes del servicio. Esta vista interpretable contendrá toda la información necesario para que el servicio pueda ser ejecutado, como una lista de los tipos de capacidades a las que se pretende acceder, un conjunto de restricciones de acceso a esas capacidades, que serán utilizadas por el sistema armonizador de capacidades para seleccionar la capacidad óptima en cada momento y también un conjunto de opciones de configuración, especificadas por el usuario que se traducirán a parámetros utilizados en el acceso a las capacidades. La vista SDL interpretable se define y se analiza en la sección 7.2 Modelo de vistas.

- Desarrollo de una interfaz gráfica para el entorno de creación basada en "workflow" gracias a la cual el usuario puede crear un nuevo servicio o cargar uno ya existente y configurarlo a su antojo, agregando o eliminando componentes y también modificando la configuración de los mismos. Estas acciones se transformarán en funciones comprensibles por el motor SDLd a través del API que separa estos dos niveles. El objetivo del desarrollo de esta interfaz es proporcionar mecanismos para la validación de los motores SDLd y SDLi basándonos en el caso de uso de Sport Tracker.

En el activo experimental se ha implementado una arquitectura para el sistema de creación de tres niveles:

- En el nivel superior se ha definido una interfaz de creación que constituye el elemento de interacción con el usuario y mediante el cual se obtienen las preferencias del servicio y su composición interna.

- En el nivel medio se mantiene un árbol lógico del servicio cuyos nodos son componentes, enlaces, preferencias, etc, que son actualizadas por el nivel superior a través de un API diseñado para tal fin. Las posibilidades que ofrece este API van desde la gestión y configuración de componentes hasta la generación de vistas SDL, pasando por la interconexión de componentes y el diseño de la vista de presentación del servicio.

- En el nivel inferior intervienen los motores generadores de SDL (SDLd y SDLi) que, como su propio nombre indica, se encargan de plasmar en un documento toda la información generada en el entorno de creación para que sirva como nexo de unión con otros sistemas (sistema publicador y entorno de ejecución).

Tomando como base esta arquitectura se ha desarrollado un entorno de creación utilizando la tecnología J2ME que sirve como validación de la misma.

En las siguientes figuras podemos ver el funcionamiento del activo experimental. En la primera figura vemos la pantalla principal del entorno, utilizando el emulador Java ME Platform SDL 3.0 para Windows 7. En la segunda figura se muestra le catálogo de componentes que existen en el repositorio de componentes del teléfono móvil. En la tercera figura puede observarse el servicio "Sport Tracker" ya creado con los componentes localización, mapa, pulso cardiaco y velocidad, en el que, según el diagrama de workflow el componente de localización está conectado con el componente de mapa por una flecha.

Figura 27. Entorno de creación mIO!

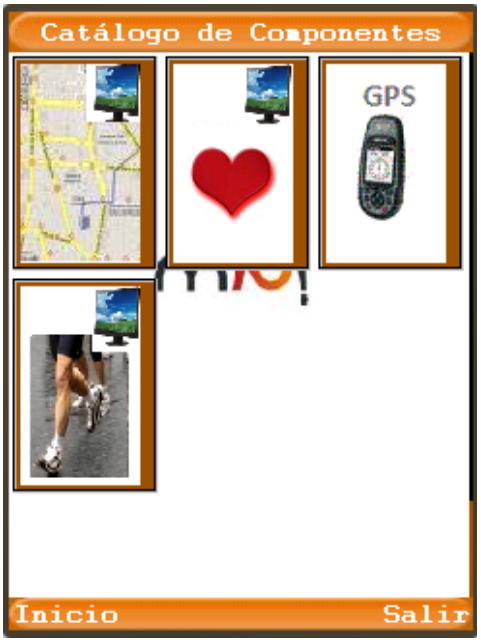

Figura 26. Catálogo de componentes

Figura 28. Servicio "Sport Tracker" en mIO!

Si bien es cierto que el activo experimental, una vez probado y revisado, tiene algunas carencias como la falta de gestión de permisos y seguridad, algunos problemas de fiabilidad y robustez y la no incorporación de algunos mecanismos complejos de optimización (estructuras complejas, mejoras en la presentación y en la vista gráfica de los componentes) podemos afirmar que el entorno cumple los objetivos propuestos y funciona como se espera para el caso de uso que ha servido como base (Sport Tracker). Prueba de esto es la generación correcta del documento SDL que coincide en características con el documento esperado, tanto en la vista de descripción (usada para la reapertura y modificación del servicio en el entorno de creación) como en la vista lógica y la de presentación (utilizadas en el entorno de ejecución para la provisión del servicio).

También relacionado con el proyecto mIO!, desde nuestro grupo de investigación se ha colaborado en el diseño e implementación de otra interfaz de creación de servicios, que se ejecuta sobre el API descrito en esta sección y que sigue el paradigma de la creación de servicios mediante Lenguaje natural escrito.

Las siguientes figuras muestran la aplicación desarrollada, escrita en Flash Lite:

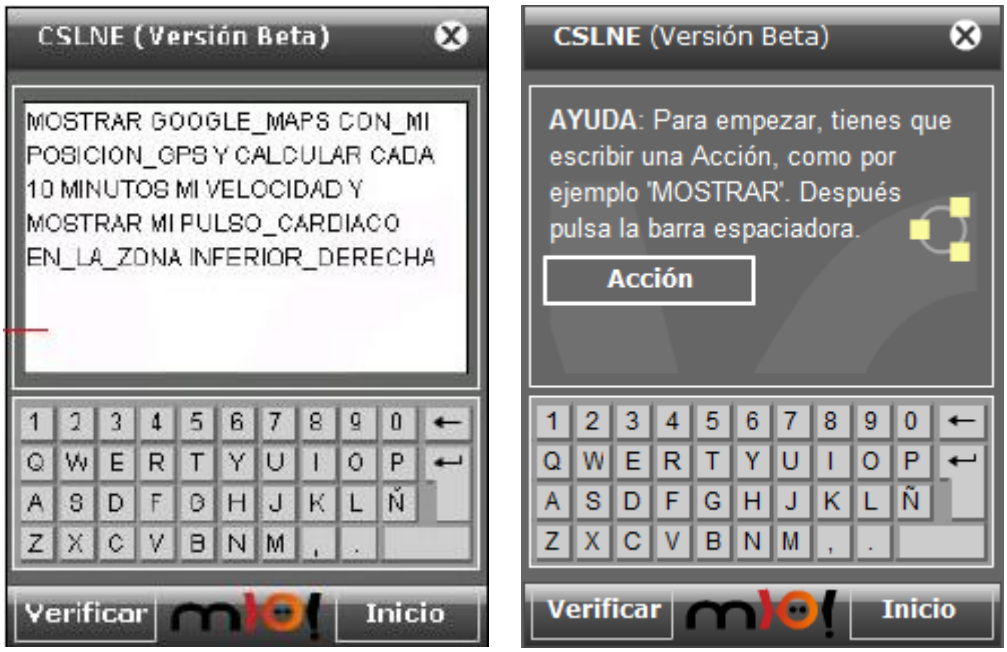

Figura 29. Interfaz de creación de servicios mediante lenguaje natural

7 Identificación del lenguaje de descripción de servicios

Llamamos lenguaje de descripción de servicios (a partir de ahora SDL) a la especificación utilizada para definir qué contiene y cómo se comporta un servicio. En concreto, es el documento que contiene la especificación del servicio escrita que se genera en el entorno de creación y que se utiliza a lo largo del ciclo de vida del servicio.

El lenguaje SDL que se describe en esta sección corresponde con el utilizado por la arquitectura desarrollada en la sección 6, relativa al proyecto mIO!. Este lenguaje está condicionado por los requisitos que se imponen desde el entorno *prosumer*, el escenario analizado en la sección 4.1, las restricciones tecnológicas analizadas en la sección 3.8 y la definición de los elementos que intervienen en la arquitectura avanzada para la composición, provisión y consumo de servicios en movilidad. Estos condicionantes se analizan de forma estructurada en la sección 7.1 Requisitos necesarios impuestos al SDL.

El lenguaje SDL debe definirse según un modelo de vistas, para que consiga definir a los servicios en todos los estados en los que se pueden encontrar dentro de la plataforma. En la sección 6.4 Evaluación y validación se mencionaban las vistas de descripción e interpretable, que contienen los datos del servicio a ser consumidos por los entornos de creación ejecución respectivamente. En la sección 7.2 Modelo de vistas se realizará un estudio pormenorizado de las diferentes visiones que se necesitan para que un servicio y un componente queden completamente definidos.

Estas vistas están relacionadas con distintos procesos dentro de la arquitectura. Por ejemplo, la vista interpretable de un servicio se genera en tiempo de creación pero se consume en tiempo de ejecución. En la sección 7.3 se define el proceso de generación de vistas.

Este modelo para el lenguaje de descripción de servicios se ha validado mediante la definición de un lenguaje de descripción basado en XML para las vistas de componente y de servicio. En la sección 7.4 y en el Anexo (B1 y B2) se describe este lenguaje mediante la notación BNF (Backus-Naur form).

7.1 Requisitos necesarios impuestos al SDL

El proceso de creación, indudablemente, impone una serie de requisitos que debe cumplir el documento SDL que define el servicio.

7.1.1 Requisitos de complejidad y procesamiento

El documento SDL atraviesa todas las fases del ciclo de vida de un servicio y recibe modificaciones en tiempo de creación y publicación mientras que en ejecución es interpretado.

En tiempo de creación el documento SDL es generado a partir de la descripción realizada sobre los componentes que integran el servicio añadiendo la información de visualización de dichos componentes en el entorno de creación y los metadatos asociados al proceso de creación.

En tiempo de publicación se produce la generación de la información que será utilizada para localizar la plantilla o la instancia y se efectúa la transferencia de la plantilla a un repositorio de plantillas externo y de la información de localización de la instancia a un directorio de instancias.

En tiempo de ejecución se produce el análisis del documento SDL con el fin de extraer código que sea directamente interpretable por el entorno de ejecución. Es en este proceso cuando se produce la instanciación de un servicio. Es posible que el código requiera un intérprete que transforme el lenguaje de descripción de servicios utilizado a lenguaje ejecutable sobre la plataforma concreta en que se ejecute.

Desde el punto de vista de las limitaciones de procesamiento la fase en la que se produce una sobrecarga debería ser la menos restrictiva en otras funciones por lo que parece lógico **intentar liberar dicha carga en la medida de lo posible del proceso de ejecución**. La creación permite recibir una carga extra de procesamiento por tres motivos:

- En tiempo de creación se pueden permitir retardos producidos por el procesamiento ya que no es necesaria ninguna ejecución de servicios en tiempo real.

- La creación se realizaría una vez frente a las "n" veces que el servicio sería consumido.

- La creación puede realizarse en equipos con mayores prestaciones (PC) o incluso si se realiza desde terminal móvil, sería posible contar con algún equipo en red al que se transfiriese la carga de realizar la traducción en el momento de la publicación, mientras que la ejecución se realizará siempre en el dispositivo móvil.

La **diferenciación de usuario experto y usuario no experto** es una opción muy interesante a la hora de separar los listados de componentes a mostrar según el tipo de creación que se requiera. El problema de esta medida es que añade también

complejidad a la tarea de creación y a la carga que el terminal pudiese tener durante la misma.

7.1.2 Restricciones impuestas por el modelo

El **modelo** que se sugiere para la definición del lenguaje de descripción de servicios es el **Model-view-controller (MVC)**:

Este modelo constituye un patrón de arquitectura utilizado en ingeniería del software. La utilización de este patrón aísla la lógica de la interfaz de usuario resultando una aplicación en la que es fácil de modificar no sólo la apariencia de la aplicación sino también la lógica de negocio sin afectar la una a la otra.

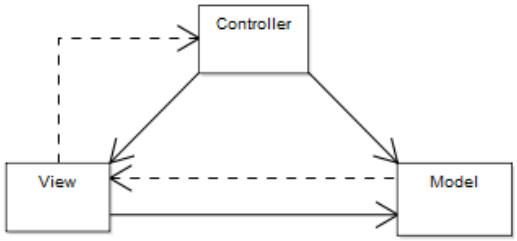

Figura 30. Modelo MVC

Los tres elementos que intervienen en este patrón son:

- Modelo: Representación de la información con la que una aplicación opera en un dominio específico. Muchas aplicaciones utilizan un mecanismo de almacenamiento persistente como una base de datos para almacenar la información. La capa de acceso a este tipo de información está incluida en el modelo.

- Vista: Transforma el modelo para que sea apropiado para la interacción. Típicamente constituye un elemento de interfaz de usuario. Pueden existir múltiples vistas de un mismo modelo para distintos propósitos.

- Controlador: Procesa y responde a eventos (generalmente acciones del usuario) y pueden invocar cambios en el modelo de forma indirecta.

Siguiendo este modelo se separará por un lado la información almacenada en los componentes (el modelo) por otro la capa de presentación que se le ofrece al usuario (vista) y por último el conjunto de operaciones que se realizan en tiempo de ejecución que generan el resultado a presentar (controlador).

En tiempo de creación se produce el almacenamiento de la representación gráfica de un servicio en un archivo mediante una descripción textual. Esta descripción, que pertenece a la lógica de una plantilla, define la ubicación de cada componente en el

entorno de creación y las conexiones entre estos componentes. La descripción consistirá en un árbol de bloques XML en donde cada bloque será un componente.

La **vista ejecutable o interpretable** de un servicio se generará a partir de la vista de descripción y describe el modo en el que los diversos componentes se relacionan entre sí. En el **módulo de lógica se ha realizado una división para indicar la parte de la lógica que se ejecuta en el servidor y la que se ejecuta en el cliente.**

Los **componentes descritos en la vista gráfica** como componentes sencillos, cuya funcionalidad es entendida por un usuario no experto y que representan una acción concreta **son descompuestos en funciones más sencillas** para que sean entendibles por el motor de ejecución. Así, por ejemplo, un componente "localización" descrito en la vista gráfica como una caja de un determinado tamaño y color que se conecta a su salida con otro componente de mapas de esta forma:

```
<componente1 id=localizacion>
      <set name=localización color=#FFFFF loc=27,65>
      <link id=out1 type=dataLink compID=mapas>
</componente1>
```

Figura 31. Ejemplo SDL de diseño de componente Localización

Puede convertirse a:

```
<servidor>
        <estructuras_de_datos>
        <struct nombre=”locationmsg” >
<campo identificador=”timestamp” tipo_de_campo=”DateTime” />
        <campo identificador=”gpsposition”
tipo_de_campo=”String”/>
        </struct>
        </estructuras_de_datos>
        <variable identificador=”location_info”
tipo_de_dato=”locationmsg” />
        <variable identificador="posicion" tipo_dato="string" />
        <lista_de_procedimientos>
        <procedimiento identificador="getMyCurrentLocation">
<resultado_proc tipo_de_dato="void" />
<lista_parametros/>
<bloque>
<wait tiempo = "1" unidades = "segundo">
<invoke identificador="get.GPSLocation"
capacidad=”Internet_Localizacion” datos_semanticos="">
<result_invoke identificador="respuesta" tipo_dato="String"/>
</invoke>
</wait> …
<cliente>  …  <cliente/>
```

Figura 32. Ejemplo SDL de ejecución de componente Localización

En el cuadro anterior puede verse como la lógica del componente "localización" se transforma en dos partes, una que se interpretará en el sistema del usuario proveedor del servicio (parte servidora) y la otra en el sistema del usuario consumidor (parte cliente), por lo tanto la vista gráfica no se ejecutará directamente sino que debe ser transformada luego a otro código que deberá ser ejecutado.

7.1.3 Limitaciones de creación móvil

Para la elección de un lenguaje de descripción de servicios que describa la forma gráfica del servicio en el entorno de creación se deben tener en cuenta las limitaciones producidas por el procesamiento de este lenguaje en un dispositivo móvil.

Como se ha planteado en el apartado anterior, la plantilla SDL del servicio consistirá en un árbol de bloques XML en donde cada bloque constituiría un componente. Para una correcta generación de un árbol XML se necesita que **el dispositivo móvil soporte funciones de *parsing* de documentos XML**.

Si se decide utilizar un lenguaje de descripción de servicios disponible en el mercado debe limitarse su búsqueda a las restricciones aplicables a la naturaleza móvil del terminal. Así pues, un lenguaje de modelado de servicios como BPMN no ofrece ninguna herramienta de modelado para sistemas móviles ni se cree posible que en un futuro próximo esta situación cambie.

Por tanto se deben analizar con especial cuidado las propuestas de lenguajes de descripción de servicios con el fin de determinar si son compatibles el entorno móvil propuesto en este trabajo.

7.1.4 Requisitos impuestos por el lenguaje y sus implementaciones

La elección del lenguaje a utilizar para el modelo de plantilla del servicio impondrá también una serie de requisitos. De esta forma, se seleccionará el lenguaje a utilizar para describir el servicio que mejor se adapte a las especificaciones planteadas.

En el caso de BPMN, como puede verse en sus conclusiones no se ha encontrado ninguna herramienta que ofrezca compatibilidad en BPMN sobre terminales móviles por lo que esta fuerte restricción hace poco probable que se vaya a utilizar esta notación como lenguaje SDL.

De esta notación se extraerá información acerca del diseño de servicios, como la división entre conectores de flujo de secuencia y flujo de mensaje. Los requisitos que impone la utilización de esta información va en la línea de asegurar que está complejidad añadida sea tratada con cautela en el trato con usuarios no expertos.

Otro paradigma extraído del estudio de la notación BPMN es la utilización de subprocesos en distintos niveles de abstracción para garantizar la ausencia de complejidad en la interacción con usuarios no expertos, lo cual, no genera ningún requisito sobre la definición de servicios.

Mediante el estudio de BPEL se ha llegado a la conclusión de que su utilización en la vista interpretable del servicio tendría muchas ventajas. Además, al poseer una implementación de motor BPEL para móviles (Sliver) se demuestra que es posible utilizar este lenguaje de tipo Workflow en dispositivos con recursos escasos. Como desventaja se puede citar que **Sliver sólo consiste en un motor BPEL** pero no contiene ningún editor ni generador de código BPEL por lo que su aplicabilidad es limitada,

Sin embargo se ha observado que el lenguaje BPEL ofrece una versatilidad que difícilmente pueda exprimirse en aplicaciones creadas por usuarios no expertos por lo que **se considerará la utilización de una versión recortada de este lenguaje.**

Finalmente, tras el estudio de la arquitectura SCA, se concluye que ésta es una opción más que recomendable para formar parte de la vista lógica o de descripción de un servicio. El **modelo de componentes visto en SCA debe adaptarse a la definición de servicio mIO!** y a las capacidades de interacción del usuario con la herramienta de creación.

Se propone, por tanto, utilizar los conceptos aprendidos tras la revisión de SCA para la **Vista Lógica** del servicio en la que aparecerá un lenguaje orientado a descripción en el que estén definidas instrucciones sencillas de interoperabilidad entre componentes y que soporte que los componentes estén descritos en un lenguaje de descripción orientado a ejecución como BPEL.

7.1.5 Restricciones por creación avanzada

Este tipo de requisitos que debe cumplir el documento SDL debe verse como un tipo de restricciones sobre los componentes utilizados desde el entorno de creación. Para una correcta comprensión de este concepto consideraremos un proceso de creación a nivel básico, es decir, uno en el que interviene un usuario no experto y otro proceso avanzado, dirigido en este caso por un usuario conocedor del entorno de creación de servicios.

En un proceso básico de creación se incorpora un componente de creación como el que se observa en la siguiente imagen:

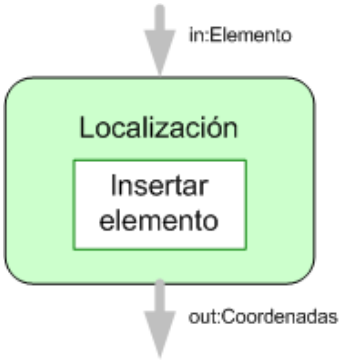

Figura 33. Componente básico de localización

Este componente deberá llevar asociado una descripción en lenguaje SDL que, al finalizar el tiempo de creación será integrada en el documento SDL que define el servicio completo. En el SDL del componente podemos observar todos estos datos:

- Entradas: Se intenta que las **entradas y las salidas sean lo más genéricas posibles** y en caso de que exista necesidad de realizar una conversión se utilizarán las capacidades asociadas u otras funcionalidades que posea el componente.
- Salidas
- Funcionalidad: Describe la función que desempeña el componente mediante lenguaje formal describiendo las capacidades que tiene asociadas.
- Metadatos: Creados durante el proceso de creación del componente se utilizan para las funciones de búsqueda y descubrimiento además de proporcionar información relevante acerca del componente como el nombre del autor, fecha de creación, restricciones de uso, etc.

En un proceso de creación avanzada un usuario experto puede definir qué tipo de capacidad desea utilizar, seleccionar los proveedores de información y encapsular distinta lógica dentro de componentes propios. En este caso el usuario incorpora al entorno de creación este componente:

Figura 34. Componente avanzado de localización

La descripción de este componente, además de contener los parámetros que se describen en el caso anterior incorpora la descripción de restricciones de mapeo a capacidades.

Las restricciones de mapeo a capacidades se imponen por las decisiones avanzadas de usuario ya que, en este caso, preferirá, diga lo que diga su perfil o la información almacenada en las preferencias de usuario la utilización de localización GPS. Esta información existirá dentro del SDL a generar durante el proceso de creación por lo que se puede considerar un requisito a tener en cuenta para el diseño de este lenguaje.

7.2 Modelo de vistas

El SDL consta de un documento maestro con la información básica sobre el servicio, y que además contiene una serie de "vistas del servicio". Cada una de estas vistas guarda información sobre un aspecto determinado del servicio. La idea es que estas vistas sean independientes, y permitan a cada uno de los módulos de la arquitectura poder acceder tan sólo a la información que necesita respecto al servicio sin necesidad de tener que transferir y *parsear* el documento completo.

El lenguaje de descripción de servicios aplicable al proyecto mIO! debe considerar un modelo de vistas como el descrito a continuación:

Modelo de un Componente

Un componente fue definido previamente en el apartado **¡Error! No se encuentra el origen de la referencia. ¡Error! No se encuentra el origen de la referencia.** como la unidad básica y funcional de un servicio. Por lo tanto un componente consta dentro de sí de cuatro elementos básicos: Metadatos, Vista de Descripción, Vista de Presentación y Vista Interpretable. A continuación se describe el modelo propuesto en la Figura 35. Vista de un componente.

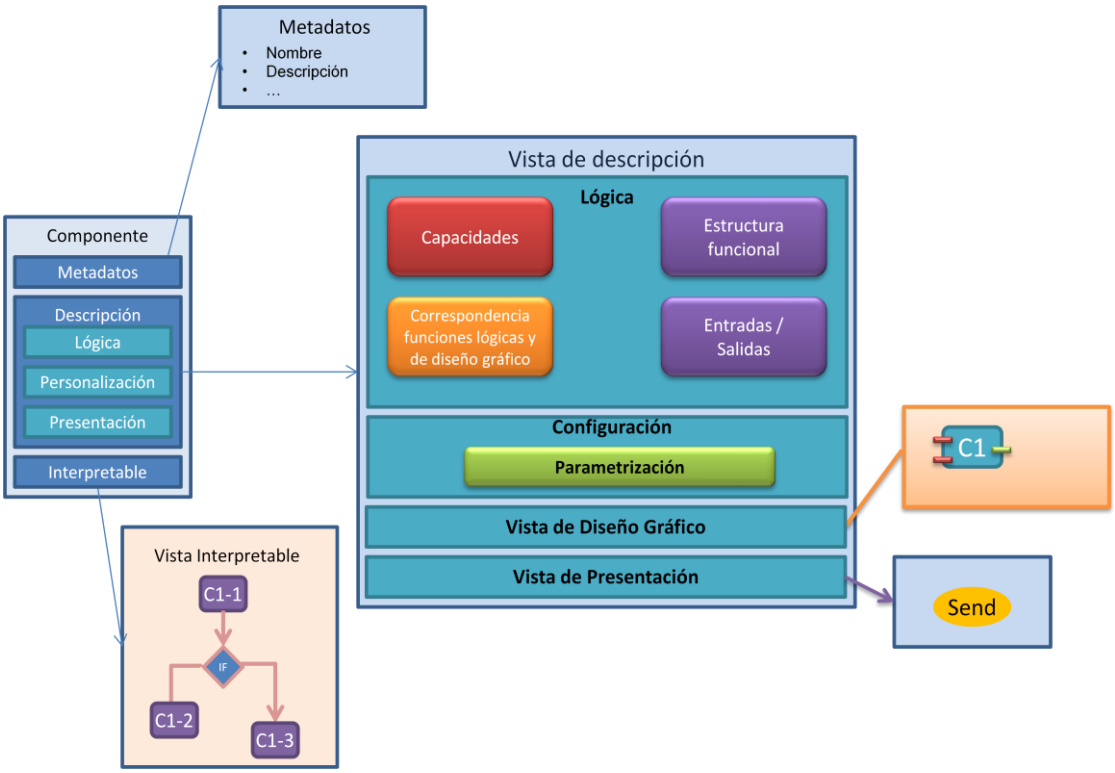

Figura 35. Vista de un componente

Metadatos: son anotaciones semánticas que describen el funcionamiento del servicio, cómo tiene que ser usado, quién puede usarlo y otras características inherentes que permiten una descripción más completa del componente al ser un elemento básico. Los metadatos tienen la función de proveer información semántica que será utilizada posteriormente en la búsqueda y clasificación.

Vista de Descripción: La vista de descripción está compuesta por elementos que son denominados *lógica y configuración*.

1) Lógica: La lógica tiene relación directa con la descripción del funcionamiento del componente.

   a) Capacidades: Dentro del modelo de componente, podrá existir información específica sobre el acceso a las capacidades, según los dos enfoques propuestos: capacidades explícitas e implícitas. En un principio, se definiría en base criterios como: nombre, tipo, referencias y restricciones que haya podido imponer el creador del servicio. Por ejemplo en el entorno de ejecución un componente que requiera tener acceso a una capacidad podrá hacer su petición en base a estos criterios; en este caso el matchmaker que es el encargado de verificar la disponibilidad tomará la decisión en base a los recursos disponibles y los posibles SLA, para luego informar al componente cual es el tipo de capacidad óptima que podrá utilizar en ese momento, dependiendo del tipo de terminal, el costo, método de acceso u otros factores que serán posteriormente definidos.

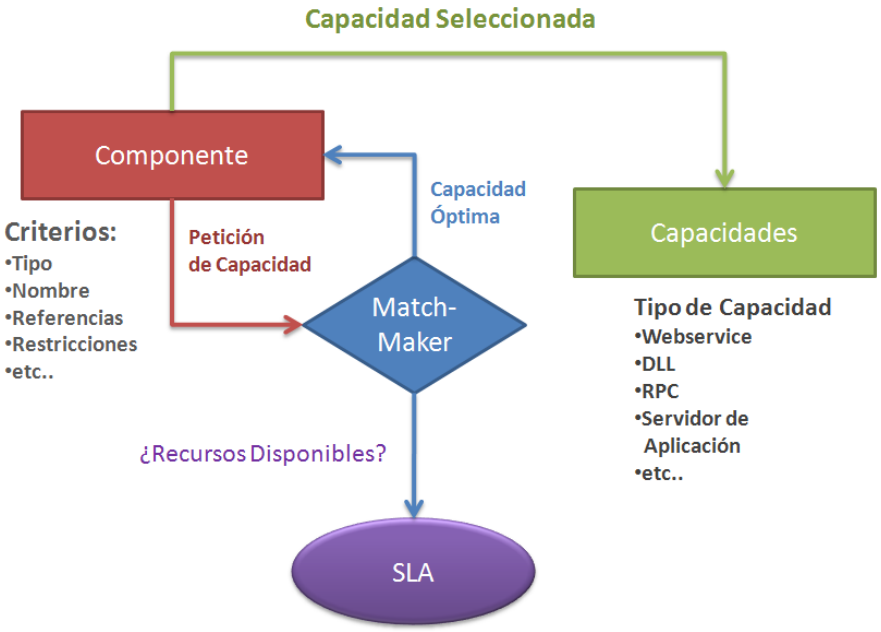

Figura 36. Acceso de un componente a una capacidad

b) Estructura Funcional: Se ha especificado como una lista de componentes que forman parte de otro más complejo. Se opina en este caso que si un componente es finalmente una anidación de otros, no tendrá una estructura visible por lo que se dificulta una descripción más profunda, adicional a la vista de interpretación, que es la encargada en este caso de la descripción lógica y funcional.

c) Correspondencia entre funciones lógicas y de diseño gráfico: Están basadas en una lista de asociaciones entre las acciones o los eventos ocurridos en el entorno gráfico y su relación con los comandos o instrucciones a definir en el entorno de creación. Esta lista de asociación se podría imaginar como "verbos". Ej. Agregar, borrar, conectar etc.

d) Entradas y Salidas: Se refiere a los puntos de comunicación que el componente tendrá para asegurar una interacción con otros componentes. Las entradas y salidas pueden ser de tipo: sólo entrada, sólo salida o entrada-salida dependiendo del flujo de información que haya sido definido para el componente en el entorno. Pueden o no existir varios puntos de comunicación que tienen un identificador y el tipo de datos que será transmitido.

2) Configuración: se refiere a la parametrización y personalización orientada hacia los usuarios finales. Son todos los elementos que afecten tanto a la lógica, a los tipos de entradas/salidas o al formato, etc. y que podrán ser modificados o adaptados ya sea en las capacidades o en el mismo componente

a) Parametrización: La parametrización se refiere a todos los elementos que son modificables por el usuario en el entorno de creación pero que no tienen que ver con la vista gráfica.

i) Preferencias de capacidades: Las preferencias de capacidades se refieren a referencias o parámetros específicos que podrían ser manipuladas al

acceder una capacidad. Estas referencias podrían ser por ejemplo en un servicio de localización, la fuente desde el usuario querrá tener la posición: desde su propio móvil, desde la red, o desde otro dispositivo.

ii) Preferencias avanzadas: Se han definido como parámetros de configuración orientados a los usuarios avanzados y que tienen su destino el componente en sí.

iii) Preferencias básicas: Las preferencias básicas se refieren a aspectos básicos que pueden ser cambiando por un usuario no técnico por ejemplo: la medida de distancia (m, km cm), parámetros genéricos (alto, medio, bajo) aplicados a cualquier característica.

3) Vista de Diseño Gráfico: La vista de diseño gráfica depende directamente del lenguaje utilizado para describirlo y no necesariamente debe ser entendido por el entorno de ejecución

4) Vista de Presentación: La vista de presentación se ha definido como cadenas de caracteres o dígitos (String) que contienen la representación gráfica del componente utilizando el lenguaje de programación gráfico y que no es necesariamente es entendido por el entorno de creación.

Vista Interpretable: es la vista ejecutable del componente, consumida por el entorno en ejecución. Esta vista a su vez se ha descompuesto en estructuras de datos y estructuras de componentes que permiten separar la lógica interna del componente de los tipos de datos que son manejados. La vista interpretable contiene estructuras de datos.

La estructura de datos se refiere a todas las estructuras, variables, métodos y lógicas que serán definidas en el componente para manipular la información. Todos los datos procesados serán tratados como estructuras independientemente de la cantidad campos que se tendrán, de esta manera será mucho más fácil su manipulación entre distintas entidades (otro componente, cliente o servidor). Acerca de las variables estas tienen similitud con las variables de los lenguajes de programación comunes en donde éstas cambian su contenido a lo largo de la ejecución del programa, pero en este caso aplicadas a los componentes. Los métodos se utilizan para proveer de una funcionalidad lógica al componente, se ha agregado inicialmente IF, CASE Y WHILE con una sintaxis común a los lenguajes de programación

Modelo de un Servicio

A continuación se expone el modelo propuesto para un servicio en la siguiente figura. Como puede verse el modelo es muy similar al utilizado para un componente.

Las diferencias pueden encontrarse en la aparición de información semántica que defina al servicio y a la nueva vista de patrones.

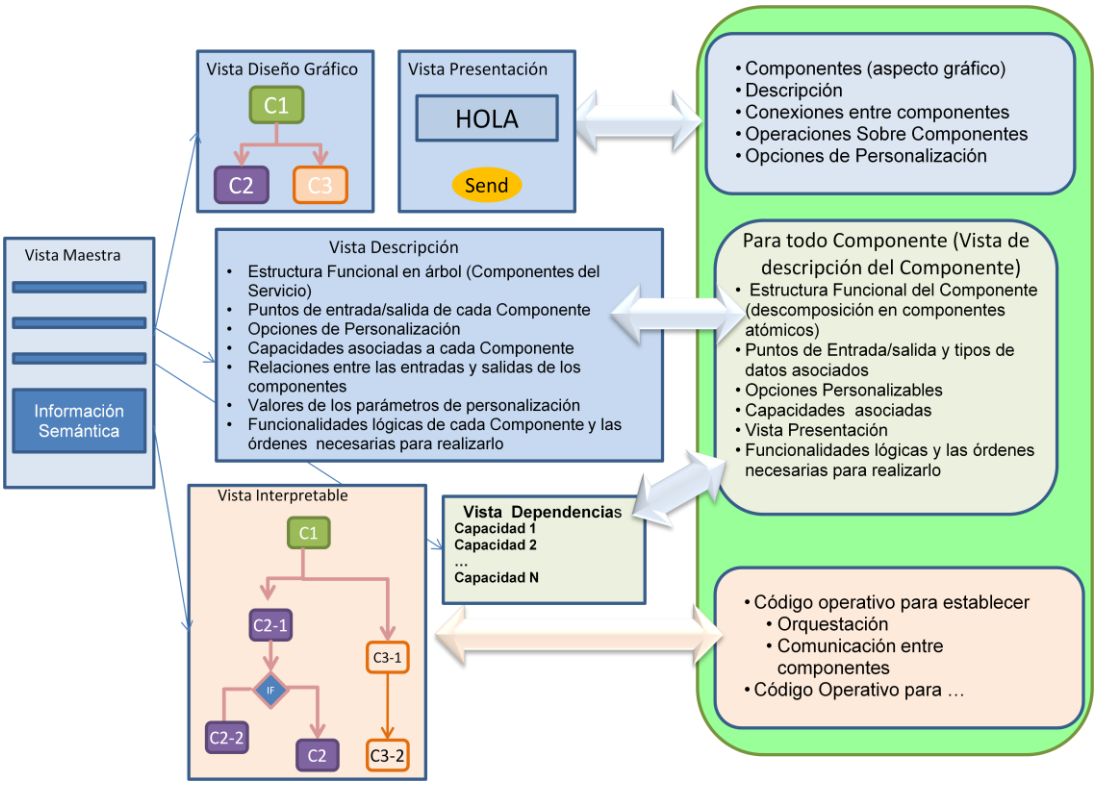

Figura 37. Estructura en vistas de un servicio

Vista de Patrones

La vista de Patrones define a un lenguaje de alto nivel que permite la descripción de los distintos elementos involucrados en la solución de un problema donde los distintos elementos, dentro del entorno mIO!, son los patrones. Una vista de patrones facilita el diseño a un nivel de abstracción mucho más elevado dentro del modelo de composición de un servicio, por lo tanto una de las metas es poder describir y darle solución a la problemática de la creación de servicios mediante la descripción de los componentes relacionados, de manera que el enfoque tomado pueda ser reutilizado dentro del sistema.

Un patrón consiste en cuatro componentes principales cuya descripción es readaptada cuando se desea describir un lenguaje que describa esta vista de patrones, y especialmente una sintaxis que permita utilizarlo dentro de la arquitectura del sistema.

- Nombre: es un identificador que describe el patrón que se está utilizando

- Descripción: al igual que el nombre es un identificador me provee información sobre el caso de uso, características de personalización y modificación.

- Soluciones: es el componente más complejo ya que provee un esquema general del servicio, al ofrecer una especie de plantilla adaptable a un problema específico. Adicionalmente a esto las soluciones deben mencionar las restricciones impuestas por el uso del patrón, lo cual tiene relación directa con el tipo de componentes o capacidades que pudieran utilizarse o no.

- Consecuencias: son una serie de identificadores que contienen información sobre las consecuencias de la utilización del patrón elegido.

7.3 Proceso de generación de vistas

Este apartado describe el proceso por el cual se van creando, modificando y enriqueciendo las vistas de un servicio en las diferentes fases que ocurren en la plataforma.

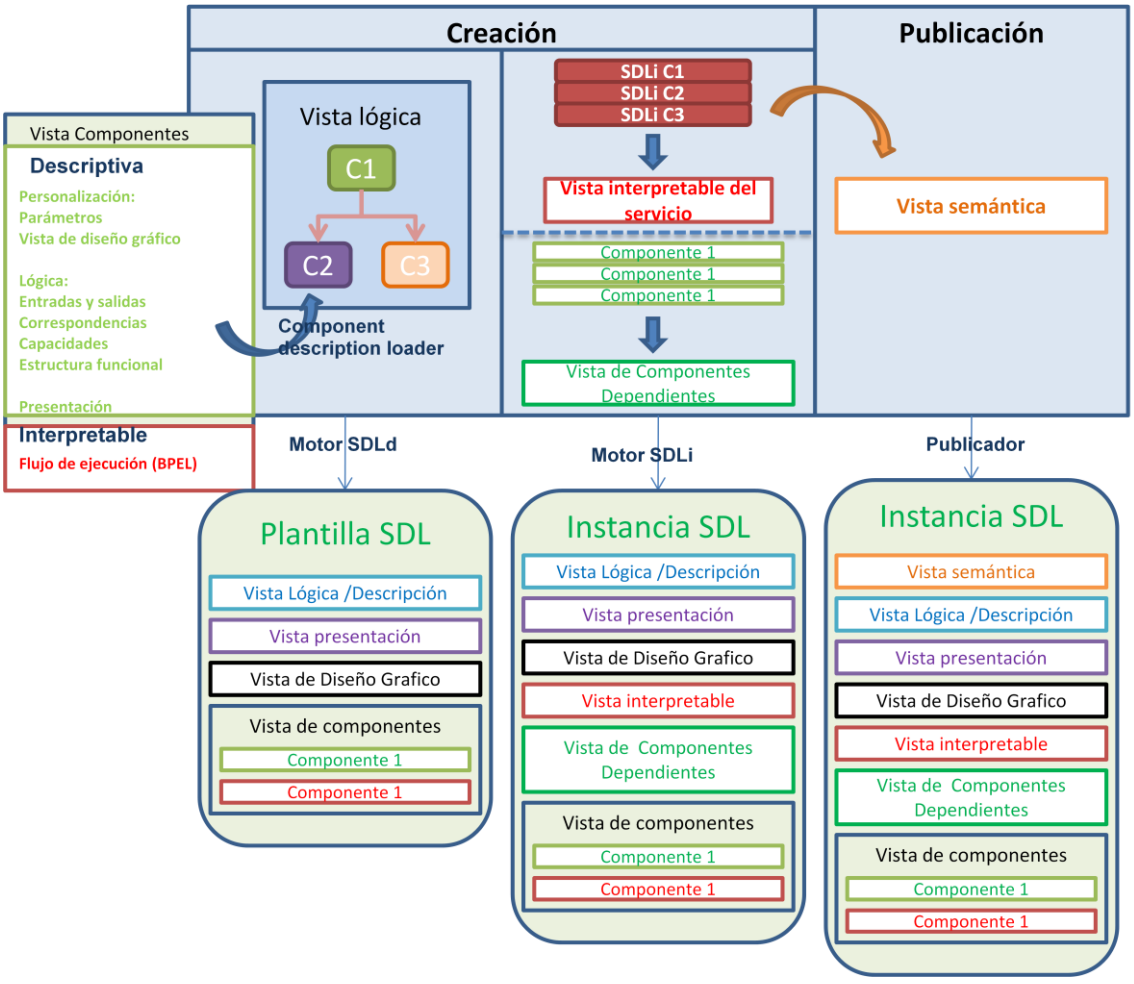

Figura 38. Generación y evolución de las vistas en un servicio

Como puede verse en la figura el documento SDL de los componentes es introducido en el entorno de creación, en el que se generan como resultado una plantilla y una instancia SDL. Existe además un proceso de publicación donde se genera una instancia SDL que incorpora información de publicación.

Si se presta atención al SDL de los componentes pueden identificarse las dos vistas fundamentales que posee:

Vista Lógica: Contiene los parámetros de personalización, lógica y presentación del componente.

Vista interpretable: Contiene el código que será interpretado por el motor de ejecución.

Una vez que el entorno de creación tiene la información de los componentes genera un árbol en memoria con los componentes necesarios para forma el servicio y mediante el motor SDLd se genera una plantilla SDL, que contiene lo siguiente:

1. Vista lógica o de Descripción: Contiene los siguientes parámetros:
    - Estructura funcional del servicio: Describe el número de componentes que intervienen en el servicio y como están conectados entre sí.
    - Puntos de e/s: Puntos de entrada y salida utilizados para cada componente. Pueden aparecer puntos de entrada o salida vacíos, bien porque ese componentes tenga algunos puntos opcionales de conexión o bien porque el servicio aún no esté completado y se requiera una vuelta al entorno de creación para finalizar su creación.
    - Opciones de personalización: Recoge las opciones especificadas por el usuario en tiempo de creación. Pueden ser básicas o avanzadas según la experiencia del usuario.
    - Relaciones entre e/s: Pueden existir otro tipo de relaciones en el conexionado de componentes además del señalado en el apartado de "puntos de e/s" como asociaciones semánticas, basadas en ontologías, etc.

2. Vista de presentación: Contiene la información sobre como representar el servicio en el entorno de ejecución. Está formada por la combinación de vistas de presentación de los componentes que integran el servicio. La vista de presentación de los componentes está incluida en la vista descriptiva de los mismos.

3. Vista de diseño gráfico: Se presenta como la combinación de las vistas de diseño gráfico de los componentes y de las relaciones entre e/s para formar una vista global de la interfaz gráfica del servicio en el entorno de creación.

4. Vista de componentes: Contiene una lista de componentes con sus respectivas vistas. El objetivo de incluir esta vista en la plantilla y la instancia de un servicio es

dotar al servicio creado de reversibilidad. De esta forma, aunque el servicio se haya personalizado y modificado, en la vista de componentes existirán los SDL originales de los componentes. Además, de esta forma se permite extraer los componentes de un servicio para poder ser utilizados en la creación de otros servicios. Lo único necesario en este caso es acceder a la vista de componentes del servicio y seleccionar los componentes a extraer (ya que toda la información sobre el componente está representada en esta vista).

El proceso de creación tiene dos fases. Tras la generación de la plantilla SDL el motor SDLi genera otras dos nuevas vistas del servicio:

5. Vista de componentes dependientes: Contiene información sobre los componentes que intervienen en el servicio y las capacidades a las que tiene que acceder. Formada con la información procedente de las vistas descriptivas de los componentes esta vista es útil en tiempo de ejecución para determinar antes de que comience el proceso si se reúnen las condiciones (existen las capacidades a las que el servicio accederá) para que el servicio se ejecute correctamente.

6. Vista Interpretable: Formada por la combinación de las vistas interpretables de los componentes presentes. En la vista interpretable aparecerán las modificaciones que el usuario ha realizado sobre los componentes, como los parámetros de personalización y las opciones de conexionado, elección de la capacidad preferente asociada a un componente determinado, etc.

Por último, durante el proceso de publicación se genera una nueva vista, que se incorpora al SDL del servicio, llamada vista semántica. El contenido de esta vista consistirá en información que describe el servicio y lo categoriza en base a ontologías para permitir su búsqueda mediante los mecanismos de búsqueda y descubrimiento de los que dispone la plataforma.

El resultado del proceso de publicación es una instancia que describe el servicio completo, ubicada en el directorio de instancias.

7.4 Evaluación y validación

Según los requisitos estudiados en el apartado 7.1 que debe cumplir un lenguaje de descripción de servicios para satisfacer las condiciones del entorno definido en este trabajo se ha trabajado en la definición de un lenguaje de descripción basado en XML para la vistas de componente y de servicio.

Se ha elegido la notación BNF (Backus-Naur form [14]) para definir formalmente las estructuras del lenguaje de descripción de servicios. El lenguaje BNF permite de una manera sencilla la definición de la sintaxis de un lenguaje de programación, además de utilizar un lenguaje natural que puede ser entendible a simple vista por un humano. Además, BNF se utiliza extensamente como notación para las gramáticas de

los lenguajes de programación, de los sistemas de comando y de los protocolos de comunicaciones, así como una notación para representar partes de las gramáticas de la lengua natural.

En el anexo (B.1 y B.2) se describe este lenguaje de descripción de servicios en notación BNF para la definición de un componente y, posteriormente para la definición de un servicio.

8 Conclusiones y trabajos futuros

El objetivo prioritario de este trabajo ha sido ofrecer una visión de los nuevos paradigmas de creación, provisión y consumo de servicios para la creación de una nueva generación de servicios en movilidad, orientados al usuario *prosumer* a través de su terminal móvil.

El entorno *prosumer* en movilidad es un entorno nuevo, influido por aspectos tecnológicos, por los servicios existentes y por los condicionantes impuestos por el uso que les pueden dar los usuarios. El entorno *prosumer* en movilidad modifica o adapta las visiones y definiciones tradicionales del mundo de servicios, por lo que es necesaria una reformulación de ciertos conceptos y una revisión de la aplicabilidad de tecnologías para esta nueva visión.

En lo referente a aspectos del servicio:

1. Los servicios descritos en este trabajo, dada la peculiaridad del dominio en el que se inscriben, no podrán ser definidos completamente por los lenguajes de descripción de servicios existentes y se necesitará una combinación de varios lenguajes de distintos propósitos, además de una extensión de algunos de ellos para poder satisfacer las necesidades identificadas.

2. Finalmente, en lo tocante a aspectos del usuario: Para que un usuario pueda abstraerse en la creación de servicios de sus complicaciones técnicas será necesaria la definición de componentes comprensibles por los usuarios que puedan ser transformados en elementos ejecutables en otras fases del ciclo de vida de un servicio.

En relación con la arquitectura genérica desarrollada en la sección 5 se han realizado pruebas de concepto sobre una plataforma OSGi y un lenguaje SDL reducido y se han validado mediante la difusión de este trabajo en congresos nacionales e internacionales (ver A. Publicaciones Relacionadas).

La arquitectura avanzada desarrollada en la sección 6 identifica un extenso catálogo de módulos y componentes necesarios para desarrollar una plataforma de composición, provisión y consumo de servicios. Además, en esta sección se exponen diagramas que reflejan la relación entre los distintos módulos y el procedimiento definido para soportar los mecanismos de creación, publicación, ejecución y búsqueda de servicios.

Se ha realizado una validación de la arquitectura respecto al subsistema de creación, comprobándose como el motor de creación genera de forma satisfactoria vista SDL de descripción de un servicio y consigue transformarla a la vista ejecutable. Las interfaces

basadas en workflow y en el paradigma de composición de servicios mediante la utilización de lenguaje natural.

El lenguaje de descripción de servicios definido en este trabajo considera todos los requisitos y restricciones impuestas por el entorno, por la definición del servicio y por las tecnologías disponibles. Está dividido en distintas vistas, que describen los estados en los que un servicio puede encontrarse y se describe en su totalidad en el Anexo.

8.1 Trabajos futuros

La experiencia y los resultados procedentes del desarrollo de este trabajo serán enfocados a progresar en unas líneas de investigación bien definidas. Para describir las líneas de investigación y la evolución que se pretende alcanzar a través de trabajos futuros conviene volver a mencionar, de forma resumida, las principales contribuciones de este Trabajo Fin de Máster y, relacionado con cada una de ellas, indicar los objetivos propuestos para los próximos trabajos en sus respectivos campos.

- Diseño y desarrollo de una arquitectura para la provisión de servicios para el usuario *prosumer* en movilidad.
- Diseño de plataforma orientada a componentes

La línea de investigación asociada busca la convergencia entre las arquitecturas SOA y las basadas en componentes y busca en la tecnología OSGi la solución que integra estos dos tipos de arquitecturas.

Los futuros trabajos en esta línea de investigación irán sobre la provisión de semántica en la plataforma OSGi. También se pretende aplicar los conceptos desarrollados en este trabajo, relativos a las arquitecturas de provisión de servicios, a otros entornos, como los hospitalarios, de e-health y rurales. Ya hemos desarrollado algunas propuestas de plataformas para entornos hospitalarios y urbanos, como puede apreciarse en A. Publicaciones Relacionadas, concretamente en [2] [7] [4] [8] y [5] respectivamente.

- Diseño de mecanismos de composición de servicios.
- Definición de flujos de ejecución y provisión de arquitectura de control para controlar el proceso de ejecución de los componentes.

La línea de investigación asociada busca profundizar en las arquitecturas basadas en componentes y en las ventajas de utilizar la composición de servicios en el proceso de creación.

Los futuros trabajos en esta línea se basarán en el desarrollo de funciones de orquestación, coreografía y composición de servicios. Además, queremos avanzar en la línea de dotar a los componentes de funciones de adaptabilidad, a través de la plataforma, para que modifiquen su comportamiento según cambios en el contexto o en las preferencias de usuario.

- Diseño de un lenguaje SDL para la definición de servicios, componentes y capacidades.

La línea de investigación se centra en la definición de los elementos que interviene en el entorno del usuario. Esta idea se aproxima a la visión de "La Internet de las Cosas" que defiende el acceso y el uso de los recursos de Internet de manera sencilla e inmediata.

Los futuros trabajos en esta área consistirán en la definición formal de servicios y en la provisión de características semánticas a los elementos del entorno.

A. Publicaciones relacionadas

Internacionales

[1] **Towards the convergence between IMS and social networks.** Ramon Alcarria, Tomas Robles, Gonzalo Camarillo, ICWMC 2010, Valencia, Spain, 2010.

Este trabajo describe una solución para la provisión de convergencia entre las redes de nueva generación, como la arquitectura IMS y las redes sociales. En este trabajo se refleja el concepto de API para la composición de servicios por *enablers* (habilitadores) y funcionalidades de Facebook.

[2] **Communication Architecture for Smart Services in Hospital Environments**. Augusto Morales, Tomas Robles, Ramon Alcarria, David Alonso, Silvia Platas, II International Workshop on Ambient Assisted Living, Valencia, Spain, 2010.

En este trabajo se presenta una plataforma de despliegue de servicios utilizando la tecnología OSGi en entornos hospitalarios. Esta publicación está basada en el proyecto CARDINEA [11], y constituye una forma de validación de la arquitectura genérica de provisión de servicios definida en la sección 5.

[3] **Converged Service Provision in Multidomain Environments for the Future Internet.** Sergio Gonzalez-Miranda; Tomás Robles; Augusto Morales; Ramón Alcarria; Eduardo Pico Hernandez, ICIN 2009.

En este trabajo se exponen mecanismos de provisión de servicios que garantizan en control adecuado de perfiles de usuario y la composición de servicios en entornos convergentes a través de las capacidades que ofrecen los *enablers* IMS.

[4] **CARDEA: Service platform for monitoring patients and medicines based on SIP-OSGi and RFID technologies in hospital environment**. Saúl Navarro, Ramón Alcarria, Juan A. Botía, Silvia Platas and Tomás Robles, OpenHealth Spain, 2009

Este trabajo describe el proyecto CARDEA [10], que define una infraestructura abierta de provisión de servicios en entornos hospitalarios basada en OSGi, RFID y SIP.

Se exponen los resultados finales del proyecto como prueba de validación de la arquitectura propuesta.

Nacionales

[5] **Plataforma de composición, provisión y consumo de servicios para el nuevo universo inteligente**. Ramon Alcarria, Tomas Robles, Augusto Morales, Sergio Gonzalez-Miranda. JITEL 2010, Aceptado, Pendiente de publicación.

Este trabajo representa la evaluación y validación de la arquitectura para la composición, provisión y consumo de servicios en movilidad. Está basado en el proyecto mIO! [13] y describe los principales retos tecnológicos a los que hay que enfrentarse para el diseño e implementación de una plataforma de provisión de servicios para el Universo Inteligente.

[6] **Arquitectura para Provisión de Servicios Ubicuos en redes IMS-P2P**. Augusto Morales, Tomas Robles, Ramon Alcarria, Sergio Gonzalez-Miranda. JITEL 2010, Aceptado, Pendiente de publicación.

Este trabajo desarrolla el concepto de usuario *prosumer* para redes convergentes y P2P y propone una integración de las infraestructuras de estos dos tipos de redes para unificar los mecanismos de provisión y consumo de servicios. Para ello se propone utilizar el servidor XDMS de IMS como repositorio de documentos SDL de servicios.

[7] **Desarrollo de una Arquitectura de Comunicaciones para Transmisión de Señales Médicas en Entorno Hospitalario.** David Alonso, Tomas Robles, Augusto Morales, Ramon Alcarria, JITEL 2010, Aceptado, Pendiente de publicación.

Este trabajo desarrolla una arquitectura de comunicaciones similar a la arquitectura genérica tratada en este trabajo. En este artículo se desarrolla el paradigma de Internet de la cosas, como facilitador de la proliferación de servicios que acceden a capacidades en el entorno, lo que conecta con el paradigma de Universo Inteligente.

[8] **Cardea. Una plataforma OSGi para Servicios Hospitalarios.** Saúl Navarro, Silvia Platas y Ramón Alcarria, JITEL 2009

Este trabajo está basado en el proyecto CARDEA [10]. Los servicios son representados en la arquitectura como bundles, lo cual facilita a la plataforma OSGi intervenir en su ciclo de vida (arranque, publicación, búsqueda y detención).

B. Anexos

1. Definición en BNF de la Vista de un Componente

En esta sección del anexo se explica de manera detallada el lenguaje de descripción de servicios diseñado para este trabajo en notación BNF.

Dentro de esta definición se ha identificado con el texto ":: soporte de información semántica", aquellos elementos sintácticos que esperamos sirvan para estructurar la información semántica asociada al componente. Esta información semántica será recabada durante la fase de publicación y añadida a estos elementos sintácticos para su posterior utilización.

```
//Estructura general del Modelo de Componente

<componente> ::= <metadatos> <vista_de_descripción> <vista interpretable>

//Metadatos

<metadatos> ::= <id_componente> <descripcion> <autor> <compañia> <version>
<coste> <lista_de_metadatos> ";"

<id_componente>   ::= <identificador> <datos_semanticos>  // Nombre del
Componente

<descripcion>     ::= <string> // descripción informal de las
características y funcionamiento del componente ::  soporte de información
semántica

<autor>     ::= <apellido> "," <nombre>

<compañía>  ::=  <nombre>

<versión>   ::=  <digito>

<coste>     ::=  <digito> "." <digito> | <string>

<tipo_componente> ::= <string>

<lista_de_metadatos> ::=  BL_METADATOS <metadato> "," + EL_METADATOS // ::
soporte de información semántica

<metadato> ::=  <tipo_metadatos> <identificador> "=" <valor>|
identificador "="  "(" lista de metadatos ")" // ::  soporte de información
semántica


<vista_de_descripción> ::= <logica> <personalizacion>
<vista_diseno_grafico>  <vista_presentacion>
```

<logica> ::= [<capacidades>] <correspondencia_funciones_logicas_y_diseño gráfico> <entradas/salidas>

<capacidades>::= BL_CAPACIDADES <capacidad> * “,” EL_CAPACIDADES

<capacidad> :== <tipo_capacidad> <id_capacidad> [<referencia>] [<descripción>] [<valor>] <datos_semanticos>

<tipo_capacidad> ::= <valor>

<id_capacidad> ::= <string>

//Las referencias son los identificadores que apuntan a las capacidades y permiten conocer de dónde van a suministradas. Las referencias son los identificadores que apuntan a las capacidades Por ejemplo un URI.

<descripción> ::= “string”

//La descripción permite al matchmaker localizar la información adecuada si no se ha proporcionado una capacidad explícita mediante la referencia

<restricciones> ::= <nombre> “=” <valor>

//Se define la asociación entre las funciones lógicas que aparecen en el interfaz gráfico de diseño del servicio y la secuencia de comandos sobre el API del motor de generación que es necesario realizar para implementar cada una de las funciones lógicas

<correspondencia_funciones_lógicas_y_diseño_grafico> ::= <lista de asociaciones>

<lista de asociaciones> ::= <datos_semanticos> :: <asociacion> * “,”

<asociacion> ::= <accion> “=” <llamada_procedimiento>

<accion> ::= <string>

//Enumeración, Identificación y descripción de cada uno de los puntos de entrada y salida del componente.

<entradas_y_salidas> ::= <entradas> <salidas> <entradas-salidas>

<entradas> ::= ENTRADAS <lista_de_puntos_de_comunicación>

<salidas> ::= SALIDAS <lista_de_puntos_de_comunicación>

<duales> ::= DUALES <lista_de_puntos_de comunicación>

<lista_de_puntos_de_Comunicación> ::= <punto_de_comunicación> * “,”

<punto_de_comunicacion> ::= <identificador> <tipo_dato> <datos_semanticos>

//Inicio de los datos parametrizables y personalizables

<personalizacion> ::= <parametrización>

<parametrizacion> ::= <preferencias_de_capacidades> <preferencias_avanzadas> <preferencias básicas>

<preferencias_de_capacidades> ::= <proveedor_de_capacidades> <datos_semanticos>

<proveedor_de_capacidades> ::= <nombre> "=" <valor> // De quién quiere el usuario por ejemplo: la localización, el sensor etc. :: soporte de información semántica

//Se han definido preferencias de ejemplo tanto avanzadas como básicas, así como también listas genéricas

<preferencias_avanzadas> ::= <Cantidad_de_usuarios_concurrentes> <Puerto utilizado> <tipo_de_codecs> <lista_de_p.avanzadas> <datos_semanticos>

<cantidad de usuarios concurrentes> ::= <digito>

<puerto> ::= <digito>

<tipo_de_codecs> ::= <string> + ","

<lista_de_p.avanzadas> ::= <p.avanzadas> "," + //Lista de una o más preferencias separados por comas

<p.avanzadas> ::= <tipo> <identificador> "=" <valor>
| identificador "=" "("<lista_de_p.avanzadas> ")"

<preferencias_basicas> ::= <medidas_utilizadas> <frecuencia> <lista_de_p.basicas> <datos_semanticos>

<medidas_utilizadas> ::= <string>

<lista_de_p.basicas> ::= <p.basicas> "," + //Lista de una o más preferencias separados por comas

<p.basicas> ::= <tipo> <identificador> "=" <valor>
| <identificador> "=" "(" lista de p.basicas ")"

//Vista de Diseño Gráfico

<vista_diseno_grafico> ::= B_VISTAVDG <String> E_VISTAVDG

<vista_de_presentación> ::= B_VP <string> E_VP

//Se representa el componente en el entorno de ejecución, este depende directamente del lenguaje utilizado para describirlo y no necesariamente debe ser entendido por el entorno de ejecución

//Vista Interpretable

```
   //En este punto se diferencia entre el código a ejecutar por el cliente y
del servidor

   <vista_interpretable> ::= [<cliente>] [<servidor>]

   <cliente> ::= <estructuras_de_datos>

   <servidor> ::= <estructuras_de_datos>

   <estructuras_de_datos> ::= <estructuras> <variables> <lista_procedimientos>

   //Las estructuras tienen un nombre el cual está definido por <nombre del
campo> y un <tipo de campo> que es un identificador.

   <estructura> ::= B_STRUCT  <identificador> <lista_de_campos> E_STRUCT

   <lista de campos> ::=  B_LISTA <campo> + “,”  E_LISTA

   <campo> ::= <identificador> “:” <tipo_de_campo>

    <tipo_de_campo>        ::= <identificador>

   //Definición de las variables

   <variable> ::= <estructura> | <identificador> <tipo_dato>  “=” <valor>

   //Definición de los procedimientos dentro del componente/servicio

   <lista_procedimientos> ::=   B_PROCEDIMIENTO <procedimiento> + “,”
E_PROCEDIMIENTO

   <procedimiento> ::= <identificador> <resultado_procedimiento>
<lista_parametros><bloque>[<resultado>]

   <resultado_proc> ::= [<identificador>] <tipo_de_dato>

   <bloque> ::= B_BLOQUE <instruccion> * E_BLOQUE

   <instruccion> ::=  <asignación>
                  | <if>

                  | <case>

                  | <while>

                  | <invoke>

                  | <wait>

                  | <llamada_a_procedimiento>

              // otras posibles instrucciones simples

   <asignacion> ::= B_ASIGNAR  <variable> [campo] | <expresión_escritura> “=”
<expresión_lectura> | <variable> E_ASIGNAR

   <expresión_lectura> ::= <expresión> | <leer_dato> | <result_invoke>
```

<leer_dato> ::= B_LEER <identificador_punto_entrada> E_LEER

<identificador_punto_entrada> ::= <identificador> <tipo_dato>

<identificador_punto_salida> ::= <identificador> <tipo_dato>

<if> :== IF <expresion> THEN <instruccion> * [ELSE <instruccion> *]
END_IF

<case> ::= CASE <expresion> OF<lista de casos> [default] E_CASE

<lista de casos> ::= <caso> + “,”

<caso > ::= B_CASO <lista de valores> “:” <instruccion> * E_CASO

<lista_de_valores> := <valor > + “,”

<while> ::= WHILE <expresión> ::= DO <instruccion> + E_WHILE

<invoke> ::= B_INVOKE <llamada_procedimiento> E_INVOKE

<wait> ::= WAIT <tiempo> <unidades> | E_WAIT

<wait_e> ::= WAITE <evento> <fuente_evento> E_WAITE

<fuente_evento> ::= <punto_comunicacion> | <llamada_procedimiento>

<llamada_procedimiento> ::= B_PROC_CALL <identificador> [<capacidad>] [
“(“ <parámetro> * “,”)”] [resultado_inv] <datos_semanticos> E_PROC_CALL
//llamada_procedimiento INVOKE

<lista_parametros>::= <parámetro> *”,”

<parametro> ::= <identificador> | <variable> “=” <valor_parametro>

//Nota del editor: Este punto está abierto y hay que relacionarlo con la
descripción de los Datos y la negociación en dos fases.

<valor_parametro> ::= <entero> | <real> | <string> | <time>| EXTERNO
<mensaje>

<result_invoke> ::= [<identificador>] <tipo_de_dato>

<bloque_instrucciones> ::= BLOQUE <instrucciones> + “,” E_BLOQUE

//Otros tipos de Datos utilizados durante la definición BNF.

<tipo_de_dato> ::= “boolean” |“int” | “String” | “real” | “time”|
<tipo_mIO> | <void>

<tipo_metadato> ::= <string> | <numero>

<tiempo> ::= <entero>

<unidades> ::= <segundos> <minutos> <horas>

<datos_semanticos> ::= <string> //Información semántica sobre las características del elemento

//En este apartado se definen los elementos básicos del lenguaje que son utilizados para definir los compuestos.

<digito> ::= 1|2|3|4|5|6|7|8|9|0

<letra> ::=a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
|A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

<numero> ::= <digito> +

<identificador> :: = <letra> [<letra> | <digito>] *

<rango> ::= <numero> ENTRE <numero>

<valor> ::= <string> | <numero> | <rango> | <tipo_mIO> |
<nombre> := <string> “;”
<apellido> := <string> “;”
<factor> ::= <string> | (<expresion>)

<multioperador> ::= * | / | %

<agregaroperador> ::= + | -

<termino> ::= <termino> <multioperator> <factor> | <factor>

<expresion> ::= <termino> | <signo><termino> | <expresion> <agregaroperador> <termino>

2. *Definición en BNF de la vista de un servicio*

Al igual que se definió la vista de un Componente se procederá a utilizar la Vista de un Servicio reutilizando la definición de los elementos y agregando solo los específicos del servicio.

<vista_maestra> ::= <vista_semantica> <vista_diseno_grafico_s> <vista_presentación_s> <vista_descripcion_s> <vista_de_componentes> <vista_interpretable_servicio>

//La vista semántica tiene similitud con la definición de los metadatos de los componentes por lo que es sólo se ha colocado a manera representativa, pero en el futuro puede ser unificado.

<vista_semantica> ::= <id_servicio> <descripcion_s> <categoría> <lengua> <autor_s> <compania> <version_s> <coste> <lista_de_metadatos > “;”

<id_servicio> ::= <identificador> // Nombre del Componente :: soporte de información semántica

<descripcion_s> ::= <string> // descripción informal de las características y funcionamiento del servicio :: soporte de información semántica

<categoría> ::= <string>

<lengua> ::= <string>

<autor_s> ::= <apellido> “,” <nombre>

<compañía> ::= <nombre>

<versión_s> ::= <digito>

<coste> ::= <digito> “.” <digito> | <string>

<lista_de_metadatos> ::= BL_METADATOS <metadato> “,” + EL_METADATOS // :: soporte de información semántica

<metadato_s> ::= <tipo_metadatos> <identificador> “=” <valor>| identificador “=” “(“ lista de metadatos “)” // :: soporte de información semántica

<vista_diseno_grafico_s> ::= <lista_de_componentes> <lista_conectores_graficos> <personalización_s>

<lista_de_componentes> ::= <id_componente> + “,” //La lista de componentes que serán representados gráficamente

//La lista de conectores gráficos contiene el id del componente y su respectiva vista gráfica

<lista_conectores_graficos> ::= <conector_graficos> * “,”

<conector_graficos> ::= <id_conector> <vista_dgrafico_conector>

<vista_dgrafico_conector> ::= <valor>

<personalización_s> ::= <valor> //Info de personalización gráfica tanto del servicio como de los componentes

//La vista de diseño gráfico y la vista de presentación se han definido con la misma estructura que la de un componente

//Las vistas de diseño grafico y presentación en un conector representan gráficamente la relación establecida entre dos o más componentes por el entorno gráfico (flechas, puntos etc…) este aspecto no es relevante en el entorno de creación

<vista_presentacion_s> ::= <valor>

//La vista de descripción contiene la lógica del componentes además de los parámetros configurables que personalizan el servicio.

<vista_descripcion_s> ::= <lógica_s> <configuración_s> <vista_componentes_dependientes>

<logica_s> ::= <lista_de_componentes> <lista_conectores>

<lista_de_componentes> ::= <id_componente> + “,” //La Lista de componentes que serán representados lógicamente

<lista_conectores> ::= <conector> + “,”

<conector> ::= <id_conector> <tipo_conector> (<asociación> “,”) <vista_disenografico_c> <vista_presentacion_c> <datos_semanticos>

<asociacion> ::= <Adaptacion_de_formatos> “:” <punto_entrada> “:” <punto_salida>

<punto_entrada> ::=<Id_componente> ”.” <punto_comunicacion> “:” <Id_componente> “.” <punto_comunicacion> + “,”

<punto_salida> ::=<Id_componente> ”.” <punto_comunicacion> “:” <Id_componente> “.” <punto_comunicacion> + “,”

<tipo_conector> ::= <valor> <datos_semanticos>

// Las opciones de configuración (configuración_s) dependen en gran medida del tipo de la plantilla, y la personalización del servicio utilizado así como también de los componentes

<configuracion_s>::= <lista_preferencias_s> <lista_preferencias_componentes>

<lista_de_preferencias> ::= <preferencias> “,” + //Lista de una o más preferencias separados por comas

<preferencias> ::= <tipo> <identificador> “=” <valor> | identificador“=”“(“<lista_de_preferencias> “)” <datos_semanticos>

<lista_preferencias_componentes> ::= <preferencia> , “,”

<preferencia> ::= <id_componente> <parametrizacion>

//Contiene la lista de restricciones de accesos a capacidades y dependen específicamente de cada servicio

<vista_de_componentes_dependientes> ::= <lista_restricciones_acceso>

<lista_restricciones_acceso> ::= <restricciones_capacidades > * “,”

<restricciones_capacidades> ::= <id_componente> <identificador> “=” <valor>

//La vista de componentes contiene todos los componentes vírgenes que serán utilizados en el servicio.

<vista_componentes> ::= <lista_componentes>

<lista_componentes> ::= <componente> + “,”

//La vista interpretable de la plantilla SDL utiliza los mismos componentes lógica que la de componentes y otros adicionales que se describen a continuación

<vista_interpretable_servicio> ::= <lista_vistas_interpretables_c> <cliente> [<servidor>] <lista_variables> <bloque_instrucciones>

<lista_vistas_interpretables_c> ::= <vista_interpretable> + “,”

<vista_interpretable> ::= <id_componente> <vista_interpretable_componente>

//En este punto se diferencia entre el código a ejecutar por el cliente y del servidor

<cliente> ::= <lista_variables> <bloque_instrucciones>

<servidor> ::= <lista_variables <bloque_instrucciones>

<Lista_variable> ::= <variable> + “,”

<bloque_de_instrucciones> ::= <declaración>[<instruccion>] <enviar> + “,”

<enviar> ::= <id_componente> <estructura> “=” <id_componente> <estructura> <adaptación>

<wait_servicio> ::= WAIT <id_componente> <punto_comunicacion> <variable> E_WAIT

Bibliografía

1. Information Society Technologies Advisory Group: http://cordis.europa.eu/fp7/ict/istag/home_en.html
2. Toffler, Alvin. (1980). The Third Wave.
3. Cloutier, Jean. (1975). La communication audio-scripto-visuelle à l'ère des self-media, L'Ere d'Emerec. P.U.M, Montreal.
4. Halteren, A.v. and Pawar P. (2006). Mobile Service Platform:A Middleware for Nomadic Mobile Service Provisioning. 2nd IEEE International Conference On Wireless and Mobile Computing, Networking and Communications (WiMob 2006), Montreal, Canada.
5. D. Martin (ed.): OWL-S: Semantic Markup for Web Services, W3C Member Submission, November 2004, available at http://www.w3.org/Submission/OWL-S/.
6. Web Service Modeling Ontology, http://www.wsmo.org/
7. Iñaki Vázquez, Diego López de Ipiña: Inteligencia Ambiental: la presencia invisible. Revista Sólo Programadores, Núm. 127, pp. 36-42, Julio 2005.
8. Proyecto mIO!, resumen ejecutivo del entregable E3.1.1 http://www.cenitmio.es/images/stories/mio/entregables/anyo1/e3.1.1%2520resumen%2520ejecutivo.pdf
9. Proyecto mIO!, entregable E7.1.1 http://www.cenitmio.es/images/stories/mio/entregables/anyo1/e7.1.1.pdf
10. Proyecto CARDEA http://cardea.germinus.com/ (Web en Español)
11. Proyecto CARDINEA http://cardinea.grupogesfor.com/ (Web en Español)
12. Belevitch, V.: Summary of the history of circuit theory, In: Proceedings of the IRE, vol 50, Iss 5, pp848-855, May 1962.
13. Proyecto mIO! http://www.cenitmio.es/ (Web en Español)
14. Backus, J.W. (1959). "The Syntax and Semantics of the Proposed International Algebraic Language of Zürich ACM-GAMM Conference". *Proceedings of the International Conference on Information Processing*. UNESCO. pp. 125–132
15. OWL-S Specs http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/
16. SWRL: A Semantic Web Rule Language Combining OWL and RuleML http://www.w3.org/Submission/SWRL/
17. WSDL: Web Service Description Language v1.1, http://www.w3.org/TR/wsdl
18. XML: Extensive Markup Language, http://www.w3.org/XML/
19. SAWSDL: Semantic Annotations for WSDL and XML Schema, http://www.w3.org/TR/sawsdl/

20. WSDL 2.0: Web Service Description Language Version 2.0 Core, http://www.w3.org/TR/wsdl20/
21. BPMN homepage: http://www.bpmn.org/
22. BPEL-WS: Business Process Execution Language for Web Services v1.1. http://www.ibm.com/developerworks/library/specification/ws-bpel/
23. Eclipse BPMN Modeler: http://www.eclipse.org/bpmn/
24. Intalio homepage: http://www.intalio.com/
25. Tutorial BPMN Modeler: http://wiki.eclipse.org/STP/BPMN_Component/STP_BPMN_Presentation_Hands_on_tutorial
26. Douglas K. Barry, "Business Process Execution Language (BPEL)". http://www.service-architecture.com/web-services/articles/business_process_execution_language_for_web_services_bpel4ws.html
27. Web Services Flow Language http://xml.coverpages.org/wsfl.html
28. ebPML Website http://www.ebpml.org/xlang.htm
29. Página de La Web Semántica http://www.lawebsemantica.com/contents/webSemantica/procesos/orquestacionCoreografia.html
30. XPath: XML Path Language, http://www.w3.org/TR/xpath/
31. Especificación de OASIS: http://www.oasis-open.org/home/index.php
32. Hackman, G., Haitjema, M., Gill, C., and Roman, G-C. "Sliver: A BPEL Workflow Process Execution Engine for Mobile Devices" ICSOC 2006.
33. Hackmann, G., Gill, C., and Roman, G.-C., "Extending BPEL for Interoperable Pervasive Computing," Proceedings of IEEE International Conference on Pervasive Services 2007 (ICPS 2007).
34. Service Component Architecture http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications
35. SCA Assembly Model V1.0 March 21 2007 http://www.osoa.org/download/attachments/35/SCA_AssemblyModel_V100.pdf?version=1
36. Everware-CBDI Inc, Report - CBDI-SAE TM - CBDI Service Architecture & Engineering TM - A Framework and Methodology for Service Oriented Architecture (SOA). 2006.
37. Grace Lewis, E.M., Liam O'Brien, Dennis Smith, Lutz Wrage, "SMART: The Service-Oriented Migration and Reuse Technique", CMU/SEI-2005-TN-029. 2005.
38. Olaf Zimmermann, P.K., Clive Gee, Elements of Service Oriented Analysis and Design. An interdisciplinary modeling approach for SOA projects. 2004.

39. Gartner, Daryl C. Plumier, “SODA Harnesses Web Services for Process Fusion”. 2003.

40. Arsanjani, A., “Service-oriented modeling and architecture, How to identify, specify, and realize services for your SOA”. 2004.

41. IBM, RUP Plug-In for SOA V1.0”, 10 May 2005,. 2005.
42. OSGi Alliance, http://www.osgi.org/Main/HomePage
43. Thomas Fielding, R. “Architectural Styles and the Design of Network-based Software Architectures”, phd thesis, 2000.
44. "The 3G IP Multimedia Subsystem (IMS): Merging the Internet and the Cellular Worlds" by Gonzalo Camarillo, Miguel-Angel García-Martín (John Wiley & Sons, 2006, ISBN 0-470-01818-6)
45. Project Capuchin developers Blog: http://blogs.sonyericsson.com/developerworld/tag/project-capuchin/
46. Bluetooth Service Discovery Protocol, http://en.wikipedia.org/wiki/Bluetooth_protocols#Service_discovery_protocol_.28SDP.29
47. Simple Service Discovery Protocol, http://quimby.gnus.org/internet-drafts/draft-cai-ssdp-v1-03.txt
48. Jini, http://www.jini.org/wiki/Main_Page
49. Apache-River, http://incubator.apache.org/river/
50. Apache Software Foundation Main Page, http://www.apache.org/
51. UDDI homepage: http://uddi.xml.org/
52. MyGame homepage: http://www.mygame.com/build.jsp
53. Scratch homepage: http://scratch.mit.edu
54. Long Jump homepage: http://www.longjump.com/
55. Zoho homepage: http://www.zoho.com/
56. Mobots homepage: http://mobots.org
57. YahooWidgets: http://widgets.yahoo.com/
58. Olzak, Tom. "Use free sandboxing software to isolate risky behavior". TechRepublic. December 15th, 2008.
59. Yamato, Y., Nakano, Y., Sunaga, H. (2008). Study and Evaluation of Context-Aware Service Composition and Change-Over Using BPEL Engine and Semantic Web Techniques. In: Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE 10-12 Jan. 2008 Page(s):863 – 867
60. Ma, Z., Leymann, F. (2009). BPEL Fragments for Modularized Reuse in Modeling BPEL Processes. Networking and Services. In: ICNS '09. Fifth International Conference on 20-25 April 2009 Page(s):63 – 68
61. Web Ontology Language for Services (W3C Submission) http://www.w3.org/Submission/2004/07/

62. Vaculin, R., Sycara, K. (2008). Semantic Web Services Monitoring: An OWL-S Based Approach. In: Hawaii International Conference on System Sciences, Proceedings of the 41st Annual 7-10 Jan. 2008 Page(s):313 – 313
63. Yin Qin, Hu Hao, Li Jim, Ge Jidong, Lu Jian. (2005). An approach to ensure service behavior consistency in OSGi. In: Software Engineering Conference, 2005. APSEC '05. 12th Asia-Pacific 15-17 Page(s):8
64. Diaz Redondo, R.P., Vilas, A.F., Cabrer, M.R., Pazos Arias, J.J., Marta Rey Lopez. (2007). Enhancing Residential Gateways: OSGi Service Composition. In: Consumer Electronics, IEEE Transactions on Volume 53, Issue 1, February 2007 Page(s):87 – 95.
65. Cervantes, H, S. Hall, R. (2003). Automating Service Dependency Management in a Service-Oriented Component Model, In: Workshop on Component Based Software Engineering, May 2003.
66. Bottaro, A., Gerodolle, A., Lalanda, P. (2007) Pervasive Service Composition in the Home Network. Advanced Information Networking and Applications. In: AINA '07. 21st International Conference on 21-23 May 2007 Page(s):596 – 603.
67. Ben-Shaul, I., Gidron, Y., Holder, O. (1998). A negotiation model for dynamic composition of distributed applications. In: Database and Expert Systems Applications. Proceedings. Ninth International Workshop on 26-28 Aug. 1998 Page(s):820 –825