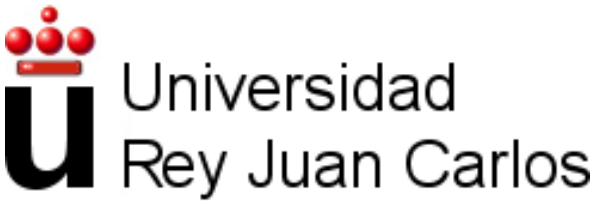

Escuela Técnica Superior de Ingeniería Informática

INGENIERÍA INFORMÁTICA – LICENCIATURA EN ADMINISTRACIÓN Y DIRECCIÓN DE EMPRESAS

Curso Académico 2010/2011

Proyecto de Fin de Carrera

Asistente Inteligente para recomendación de servicios adaptados a contexto en dispositivos móviles

Autor: Helena Cebrián Benito

Tutor: Rubén Ortiz Martín

Resumen

Este proyecto fin de carrera se enmarca dentro del proyecto mIO! en el que colabora el grupo de Inteligencia Artificial de la Universidad Rey Juan Carlos junto con otras instituciones.

El objetivo de este proyecto es el diseño e implementación de una arquitectura de agente asistente inteligente capaz de recomendar de forma proactiva servicios de valor para un usuario, haciendo uso de la información percibida de su contexto.

La revolución de las nuevas tecnologías ha propiciado la creación de este proyecto, basándose en la idea de que el usuario, a través de su dispositivo móvil, puede estar conectado en todo momento y tener por tanto acceso al ecosistema de servicios. El proyecto mIO! se basa en entornos ubicuos en los que el agente percibe el entorno del usuario como información semántica, que tras ser procesada, permite llevar a cabo la recomendación de servicios web semánticos adaptados a las necesidades detectadas para dicho usuario.

Para ello, se parte de una arquitectura dada que se compone de cuatro fases: búsqueda de objetivos, planificación de acciones, búsqueda de servicios y ejecución. Como resultado, obtenemos una implementación de referencia de la arquitectura del agente asistente inteligente para plataformas en entornos de sobremesa, así como para dispositivos móviles en la plataforma Android, que pese a su reciente aparición se postula como una de las grandes plataformas para móviles.

Índice

1. Introducción 1
1.1. Contexto del proyecto mIO! 1
1.1.1. Descripción del proyecto mIO! 1
1.1.2. Agente inteligente asistente mIO! 2
1.1.3. Gestión de contexto 3
1.1.4. Servicios mIO! 5
1.2. Arquitectura de servicios REST 7
1.3. Modelos de confianza 8
1.4. Dispositivos móviles 9
1.4.1. Android 9

2. Objetivos 11
2.1. Caso de estudio 11
2.2. Objetivos del proyecto 12

3. Descripción Informática 15
3.1. Metodología empleada 15
3.2. Arquitectura de agentes inteligentes de recomendación 17
3.2.1. Agenda 17
3.2.2. Arquitectura propuesta 19
3.3. Análisis 23
3.3.1. Casos de uso generales del sistema 23
3.3.2. Diagramas de estado para cada caso de uso 26
3.4. Diseño 34
3.4.1. Diseño de la arquitectura general 34
3.4.2. Interoperabilidad del agente con componentes externos 57
3.4.3. Agenda de acciones 66
3.4.4. Diseño de la interfaz en dispositivos móviles 77
3.5. Implementación 82
3.5.1. Tecnologías utilizadas 82
3.5.2. Diagrama de despliegue 84
3.5.3. Detalles de implementación 86
3.6. Evaluación 95

4. Conclusiones 99
4.1. Líneas Futuras 101

5. Bibliografía 103

Índice de Figuras

FIGURA 1 - ARQUITECTURA DE GESTOR DE CONTEXTO ____ 4
FIGURA 2 - COMPARACIÓN DE CUOTAS DE MERCADO DE SISTEMAS OPERATIVOS MÓVILES PERÍODO 2008 - 2010 ____ 10
FIGURA 3 - FUNCIONALIDAD GENERAL DEL AGENTE INTELIGENTE MIO! ____ 17
FIGURA 4 - RELACIÓN DEL ASISTENTE INTELIGENTE CON LA ARQUITECTURA DE AGENTES INTELIGENTES ____ 19
FIGURA 5 - DETALLE DE ARQUITECTURA DE AGENTES INTELIGENTES ____ 20
FIGURA 6 - DIAGRAMA DE CASOS DE USO ____ 24
FIGURA 7 - DIAGRAMA DE ESTADOS DE "RECOMENDACIÓN" ____ 27
FIGURA 8 - DIAGRAMA DE ESTADOS DE "DESCUBRIR ACCIONES" ____ 28
FIGURA 9 - DIAGRAMA DE ESTADOS DE "EJECUCIÓN DE UNA ACCIÓN" ____ 29
FIGURA 10 - DIAGRAMA DE ESTADOS DE "REFINAR ACCIÓN" ____ 30
FIGURA 11 - DIAGRAMA DE ESTADOS DE AGENDA "AÑADIR ACCIÓN A LA AGENDA" ____ 31
FIGURA 12 - DIAGRAMA DE ESTADOS DE AGENDA DE "BORRAR ACCIÓN DE LA AGENDA" ____ 31
FIGURA 13 - DIAGRAMA DE ESTADOS DE "REFINAR PROPUESTA" ____ 32
FIGURA 14 - DIAGRAMA DE ESTADOS DE AGENDA DE "VER DETALLE DE UNA ACCIÓN" ____ 32
FIGURA 15 - DIAGRAMA DE ESTADOS DE AGENDA DE "EJECUTAR ACCIÓN" ____ 33
FIGURA 16 - DIAGRAMA DE CLASES GENERAL ____ 35
FIGURA 17 - DIAGRAMA DE SECUENCIA DE "DESCUBRIR ACCIONES" ____ 42
FIGURA 18 - DIAGRAMA DE SECUENCIA DE "SELECCIONAR ACCIÓN Y EJECUTAR" ____ 48
FIGURA 19 - DIAGRAMA DE SECUENCIA DE "GENERALIZAR ACCIÓN" ____ 55
FIGURA 20 - API DEL GESTOR DE CONTEXTO ____ 58
FIGURA 21 - DETALLE DE BÚSQUEDA DE PLANTILLAS EN EL DIRECTORIO DE SERVICIOS ____ 60
FIGURA 22 - FLUJO DE INFORMACIÓN ENTRE EL ASISTENTE Y EL DIRECTORIO DE SERVICIOS ____ 60
FIGURA 23 - FLUJO DE INFORMACIÓN ENTRE EL ASISTENTE Y EL DIRECTORIO DE SERVICIOS DE INSTANCIAS ____ 62
FIGURA 24 - FLUJO DE PETICIONES REST ENTRE EL ASISTENTE Y EL DIRECTORIO DE SERVICIOS DE INSTANCIAS ____ 63
FIGURA 25 - DETALLE DE INTERACCIÓN AGENTE CON MÓDULOS EXTERNOS ____ 64
FIGURA 26 - DETALLE DE FUNCIONAMIENTO DEL MÓDULO DE CONFIANZA ____ 65
FIGURA 27 - DETALLE DE FUENTES DE INFORMACIÓN PARA EL MÓDULO DE CONFIANZA ____ 66
FIGURA 28 - DIAGRAMA DE CLASES DE LA AGENDA ____ 67
FIGURA 29 - DIAGRAMA DE SECUENCIA DE "EJECUTAR ACCIÓN" ____ 68
FIGURA 30 - DIAGRAMA DE SECUENCIA DE "PARTICULAR ACCIÓN" ____ 70
FIGURA 31 - DIAGRAMA DE SECUENCIA "GENERALIZAR ACCIÓN" ____ 72
FIGURA 32 - DIAGRAMA DE SECUENCIA DE "BORRAR ACCIÓN" ____ 74
FIGURA 33 - DIAGRAMA DE SECUENCIA DE "AÑADIR ACCIÓN A LA AGENDA" ____ 75
FIGURA 34 - DIAGRAMA DE SECUENCIA DE "VER DETALLE DE ACCIÓN" ____ 76
FIGURA 35 - DIAGRAMA DE CLASES DE LAS ACTIVIDADES DE LA INTERFAZ ____ 78

FIGURA 36 - DIAGRAMA DE INSTANCIAS DE LA INTERFAZ ____ 80
FIGURA 37 - DIAGRAMA DE DESPLIEGUE ____ 84
FIGURA 38 - DIAGRAMA DE DESPLIEGUE NO MÓVIL ____ 85
FIGURA 39 - CAPTURA DE LA PANTALLA PRINCIPAL Y EL LOGGER DE LA APLICACIÓN ____ 89
FIGURA 40 - CAPTURA DE PANTALLA DE CONFIGURACIÓN ____ 89
FIGURA 41 - CAPTURA DE PANTALLA DE LA AGENDA Y DEL MENÚ DESPLEGABLE DE UNA ACCIÓN ____ 90
FIGURA 42 - CAPTURA DE PANTALLA DE DETALLES DE UNA ACCIÓN Y MENÚ DESPLEGABLE DE UNA ACCIÓN ____ 91
FIGURA 43 - CAPTURA DE PANTALLA DE CONFIRMACIÓN DE EJECUCIÓN Y VALORACIÓN DE UNA ACCIÓN ____ 91
FIGURA 44 - CAPTURA DE AGENDA DE ACCIONES EN INTERFAZ SWING ____ 92
FIGURA 45 - CAPTURA DE DETALLES DE ACCIÓN EN SWING ____ 92
FIGURA 46 - CAPTURA DE SIMULACIÓN DE EJECUCIÓN EN INTERFAZ SWING ____ 93
FIGURA 47 - CAPTURA DE VALORACIÓN DE EJECUCIÓN EN INTERFAZ SWING ____ 93
FIGURA 48 - EJECUCIÓN EN DISPOSITIVOS FÍSICOS ____ 94
FIGURA 49 - PANTALLA PRINCIPAL DE LA APLICACIÓN ____ 95
FIGURA 50 - AGENDA DE ACCIONES OFRECIDA A PAULA ____ 95
FIGURA 51 - DETALLE DE LA ACCIÓN SELECCIONADA ____ 96
FIGURA 52 - NUEVAS ACCIONES TRAS PARTICULARIZACIÓN ____ 96
FIGURA 53 - CONFIRMACIÓN EJECUCIÓN DE UNA ACCIÓN ____ 97
FIGURA 54 - VALORACIÓN DEL PROVEEDOR SELECCIONADO ____ 97
FIGURA 55 - LISTA DE ACCIONES TRAS LA ELIMINACIÓN DE LA EJECUTADA ____ 97
FIGURA 56 - ESTIMACIÓN DE ESFUERZOS ____ 100

1. Introducción

Este Proyecto Fin de Carrera se ha desarrollado en el marco del proyecto de investigación mIO! dentro del grupo de Inteligencia Artificial de la Universidad Rey Juan Carlos.

En este apartado introductorio se pretende dar a conocer cuál es el contexto y la motivación que ha llevado a la creación del proyecto mIO! y más concretamente, de este Proyecto Fin de Carrera. Para ello, se expondrá en primer lugar el contexto sobre el que se ha construido este proyecto, haciendo énfasis en qué pretende ofrecer y por qué.

En los siguientes apartados se explicará en qué consiste exactamente el agente inteligente asistente mIO!, así como cada uno de los elementos externos con los que interacciona en el proceso de recomendación. Se explicará qué es un servicio en el contexto del proyecto mIO!, cómo son consumidos estos servicios y por último, la importancia que cobran los mecanismos de confianza en el proceso de recomendación. Todo ello, haciendo énfasis en la heterogeneidad del entorno y la expansión de las tecnologías móviles.

1.1. Contexto del proyecto mIO!

El proyecto mIO! se enmarca dentro del programa CENIT (Consorcios Estratégicos Nacionales en Investigación Técnica) que a través de la creación del consorcio mIO! provee la financiación necesaria para poder llevar a cabo este proyecto.

Dicho consorcio está formado por 10 empresas y un total de 19 Organismos Públicos de Investigación (OPI), distribuidos geográficamente por 12 Comunidades Autónomas diferentes.

Más concretamente, la parte que a este Proyecto Fin de Carrera corresponde, se ha llevado a cabo en el Grupo de Inteligencia Artificial de la Universidad Rey Juan Carlos de Madrid.

1.1.1. Descripción del proyecto mIO!

El proyecto mIO!, tomando como elemento central las tecnologías de dispositivos móviles, ofrece servicios de valor añadido adaptados a los usuarios, sus preferencias y sus situaciones contextuales. Para lo cual propone sacar todo el partido posible a un dispositivo que viaja con el usuario en todo momento y que está siempre conectado, de tal forma que se convierta en una ventana abierta a la información relevante y personalizada que cada usuario requiere.

Se pretende, por tanto, sentar las bases para la creación de una nueva generación de servicios de movilidad, en la que las personas son el centro de la propuesta, pudiendo elegir en cada momento dentro del universo de servicios que se le ofrece, aquel que mejor le convenga en cada momento.

El proyecto mIO! propone la idea de emplear el dispositivo móvil como un agente recomendador para el usuario. Entendiendo por proceso de recomendación aquel a través del cual el agente asesora de forma proactiva, ofreciendo servicios a los usuarios en función de su contexto y preferencias, e incluso brinda de forma automática servicios adicionales a los utilizados habitualmente por el usuario.

Dicho proceso de recomendación se compone de una serie de etapas que parten de la percepción del contexto o entorno del usuario, y a partir de éste, construir las acciones que satisfagan los objetivos del usuario, realizando una búsqueda de servicios idóneos en la infraestructura de servicios.

1.1.2. Agente inteligente asistente mIO!

En esta sección se explica cuál es, a grandes rasgos, la funcionalidad esperada por el agente inteligente, dando una visión global de cuál es el proceso que éste lleva a cabo para poder desempeñar su función.

Para ello, es necesario comenzar por explicar qué es un agente inteligente. Definimos agente inteligente como aquella entidad software que, basándose en su propio conocimiento, realiza un conjunto de operaciones para satisfacer las necesidades u objetivos de un usuario u otro programa, bien por iniciativa propia o porque alguno de éstos se los requiere (Wooldridge, 1995).

Además, en el caso del agente inteligente mIO! se espera que dicha actuación tenga cierto componente proactivo, esto es, dicho agente sea capaz de mostrar un comportamiento orientado a sus propios objetivos tomando la iniciativa para satisfacer sus fines.

Más concretamente, el comportamiento proactivo será mostrado por el agente asistente inteligente cuando proponga, de forma autónoma, acciones a realizar al usuario recomendado, partiendo de un proceso de detección de los objetivos o necesidades de éste.

El planteamiento de la detección de los objetivos que el usuario podría querer satisfacer está totalmente ligado al concepto de agente intencional, donde se plantean agentes en los que la problemática es precisamente la identificación y formación de objetivos. Aunque existen otras arquitecturas clásicas para la implementación de este tipo de agentes (por ejemplo, la arquitectura BDI – "*belief-desire-intention*"), los agentes intencionales están especialmente indicados para la adaptación de una serie de objetivos previamente detectados al nuevo conocimiento (o información contextual en mIO!) que el agente percibe (Dignum, 1998).

Para poder llevar a cabo la recomendación de acciones al usuario el agente deberá, en primer lugar, percibir la información del contexto o entorno relevante del usuario para poder, en función de esta información, generar objetivos acordes a la coyuntura o situación actual del usuario. Dichos objetivos serán por tanto generados, buscando satisfacer las necesidades detectadas del usuario.

A partir de estos objetivos generados, se desarrollarán una serie de acciones que serán mostradas al usuario en lenguaje natural. Cada una de estas mencionadas acciones tendrán asociadas un servicio, cuya ejecución satisfaría los objetivos que motivaron la creación de dicha acción.

Este proyecto tiene como objetivo hacer realidad las tecnologías que permiten prestar servicios ubicuos en un entorno inteligente y adaptado a cada individuo y a su contexto, usando el terminal móvil como base de interacción para la prestación de servicios.

Por tanto, el agente asistente inteligente centrará su objetivo en la recomendación de servicios que creen valor para el usuario, adaptándose a las circunstancias concretas de éste.

1.1.3. Gestión de contexto

En el ámbito de este proyecto, se considera contexto a la información o conocimiento implícito que rodea a la persona física que será recomendada. Por tanto, a la hora de referirse a contexto, será en referencia al entorno relevante del usuario.

Existe mucha problemática en torno a lo que delimitación de contexto se refiere. Puesto que el contexto de una persona física es algo infinito, es preciso delimitarlo, en caso contrario sería inmanejable computacionalmente. Es por esto que la definición de contexto de una entidad se limita a aquel subconjunto de características o propiedades relevantes del entorno (Penela, 2010).

El objetivo de la percepción del contexto y la gestión relativa a éste, en el ámbito de este proyecto, es el poder convertir (a partir de información de contexto) los objetivos en acciones que impliquen la satisfacción de dichos objetivos, a través de la ejecución de servicios.

Para poder llevar a cabo la percepción de dicho contexto, son necesarios una serie de mecanismos y componentes que se pasan a describir brevemente a continuación. En la Figura 1 se puede observar la arquitectura de un gestor de contexto en términos de sus componentes y conectores (iSoco, Telefonica I+D, 2009).

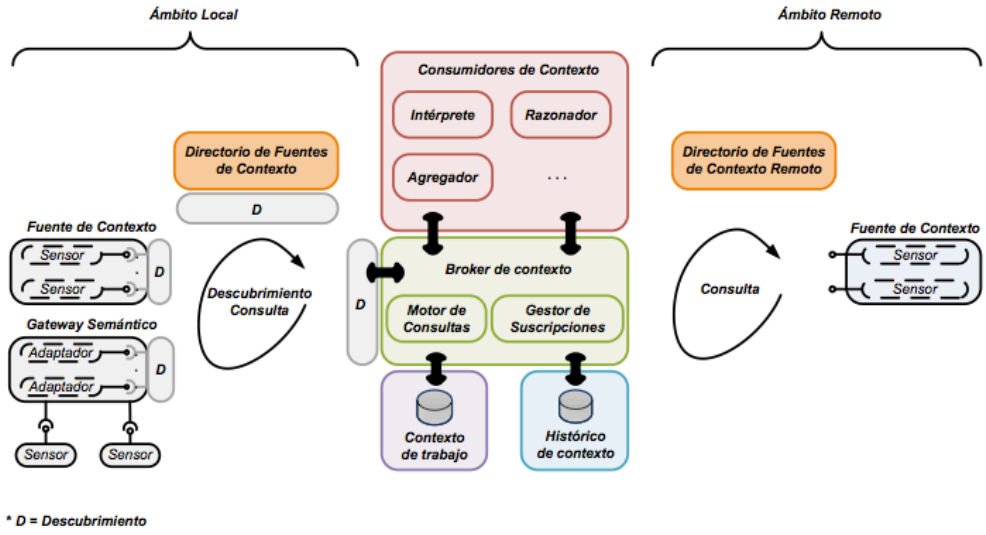

Figura 1 - Arquitectura de gestor de contexto

Puesto que se trata de poder percibir y manejar la información del contexto de una entidad, es necesario conocer cuál o cuáles van a ser las *fuentes de contexto*, entendiéndose como tal a toda entidad computacional que es capaz de proporcionar cierta información siguiendo una convención de formato utilizando un modelo ontológico.

Por otra parte un *Gateway semántico* es el componente que permite la adaptación y comunicación con fuentes de contexto que no han sido diseñadas para proveer su información de manera semantizada y adaptada al entorno mIO!. Es decir, actúa como *bridge,* alineando la percepción del contexto por parte de componentes externos a mIO! y la información de contexto que es asimilada por éste.

Un *consumidor de contexto* es una entidad computacional que necesita de cierta información sobre el estado contextual de otra entidad para su correcto funcionamiento u operación. En el ámbito del proyecto, el agente asistente inteligente mIO! es un consumidor de contexto, en tanto en cuanto precisa de su consulta, para poder formular los objetivos específicos para el usuario físico. Para ello, el módulo encargado de percibir el contexto y consumirlo, debe poder razonar semánticamente, interpretarlo y por último agregarlo, en pos de la generación correcta de objetivos acordes a la coyuntura actual del usuario.

Este consumidor de contexto, interactuará con el *bróker de contexto*, el coordinador entre las diferentes entidades, llevando un control del flujo de la información de contexto dentro de la arquitectura. Dicho bróker además, obtendrá información del *histórico de contexto*.

Pese a que en el proyecto mIO! se lleve a cabo la percepción de la información del entorno del usuario, para poder generar así objetivos acordes a la situación del usuario (su contexto), no es objetivo de este proyecto la explicación de cómo se ha diseñado e implementado este módulo. Sin embargo, ha sido precisa su utilización para poder obtener los *inputs* necesarios para el proceso de recomendación.

1.1.4. Servicios mIO!

En apartados anteriores, se ha mencionado que el objetivo del agente asistente inteligente mIO! es la recomendación de servicios que creen valor para los usuarios y que dichos servicios serán recomendados a partir de información de contexto, traducida en objetivos para el usuario.

Estos servicios que proporcionan al usuario el medio para poder obtener sus fines u objetivos, se definen en el entorno del proyecto mIO! como una entidad software sensible al contexto, con interfaz de usuario y que se ejecuta en un servidor residente: bien en el propio dispositivo móvil o bien en la red.

Se podría, por tanto, concebir como una aplicación que ofrece una funcionalidad útil para una persona y que además dispone de una interfaz de interacción con humanos.

Además, es importante notar el hecho de que los servicios en mIO! son materializados bajo dos concepciones o formas:

- Cuando no es más que una descripción de servicio publicada sin un proveedor concreto asociado, se denomina *plantilla de servicio*. Esta plantilla de servicio ofrece la descripción de su funcionalidad y algunos parámetros necesarios para la ejecución, como pueden ser las entradas y las salidas del servicio, las post-condiciones y las precondiciones.

- Cuando un proveedor ofrece un servicio concreto entonces hablamos de una *instancia de servicio*. Es decir, la diferencia principal entre una plantilla de servicio y una instancia de servicio, es la existencia de un proveedor que proporciona una implementación concreta ejecutable en una plataforma de servicios.

1.1.4.1. Lenguajes de descripción de servicios

A través del dispositivo móvil se ofrecerán servicios de valor al usuario. La forma en que el dispositivo móvil accede a toda la información de su entorno (información de contexto, servicios, etc.), es a través del uso de tecnologías semánticas, que permiten procesar esta información de forma automática por agentes software.

Existen diversos lenguajes de descripción semántica, como es el caso de *OWL-S* ó *SA-WSDL*:

- *OWL-S* o *Web Ontology Language for Services* es un lenguaje para la descripción de servicios web propuesto por la W3C en el año 2004. Es por tanto una ontología formada por los siguientes 3 módulos:

    - *Service Profile*, describe qué hace el servicio, qué le ofrece al usuario. Toma una visión global del servicio independientemente de cómo se lleve a cabo la ejecución de éste.
    - *Process model*, describe cómo el servicio es usado.

- *Service Grounding*, usado para describir cómo interactuar con el servicio. Proporciona la información necesaria dada por el proveedor para acceder al servicio.

Es el *service profile* a partir del cual se realiza el proceso de descubrimiento del servicio, puesto que es el que describe las características del servicio.

Este lenguaje descrito anteriormente es utilizado para la descripción semántica de servicios y debe ofrecernos la siguiente funcionalidad:

- **Descubrimiento de servicios**. La descripción de servicios con tecnología semántica, permite realizar una búsqueda de servicios basada en la comparación de los objetivos del usuario, con la descripción de servicio (proceso denominado *matchmaking*). De forma, que si los objetivos del usuario son coherentes con la descripción del servicio, entonces éste será idóneo para su ejecución.

- **Invocación automática**. El dispositivo computacional debe ser capaz de invocar el servicio remoto.

- **Composición de servicios**. En ocasiones para poder satisfacer las necesidades de los usuarios, es necesario llevar a cabo un proceso de composición de servicios, a través del cual y gracias a la combinación de determinados servicios, se satisfagan los objetivos de éstos.

1.1.4.2. Plantilla de Servicio

Se trata de la descripción estática de un servicio, almacenada en un repositorio de plantillas. Incluye toda la información relativa a un servicio determinado, organizada en bloques: bloque maestro (información de identificación del servicio), bloque de lógica (orquestación que debe ejecutarse), bloque de semántica (descripción semántica del servicio para su descubrimiento), bloque de seguridad (listas de control de acceso), bloque de negocio (información de tarificación).

Por tanto, la plantilla de servicio permite la descripción de un servicio para que éste se instancie, materializándose con el contexto asociado al usuario que lo instancia.

Ejemplo*: Existe una plantilla de servicio llamada "mi localización", se trata de una abstracción de servicio que permite conocer en cualquier momento la localización física (identificada por longitud y latitud). En caso de que alguien quisiese ofrecer dicho servicio, debería configurar la plantilla de servicio (con la identificación de usuario entre otros parámetros) y proveer así el servicio de localización.*

En caso de que otra persona decidiese conocer su localización en determinado momento, buscará instancias de servicios que le brinden la localización. Tras realizar la búsqueda en el directorio de servicios, recibirá N instancias de servicio ofrecidas por N proveedores diferentes (entre ellas la instancia de "mi localización"). En caso de que dicho usuario escogiese una de estas instancias, entonces se convertiría en consumidor de la misma.

1.1.4.3. Instancia de Servicio

Una instancia de servicio es ofrecida por un proveedor concreto, que instancia una plantilla de servicio mIO! El proveedor de servicios que quiera ofrecer un servicio concreto necesitará compartir la plantilla de servicios e instanciar la información sobre la ejecución.

En términos de semántica de descripción de servicios, un proveedor concreto de un servicio tendrá que completar la información relativa al *grounding* y el *process model* de un servicio *OWL-S* compartiendo el mismo *profile.*

1.1.4.4. Directorio de servicios

Un Directorio de Servicios o simplemente un directorio, puede ser interpretado de diferentes formas. En el ámbito de este proyecto el Directorio se concibe como una infraestructura de servicios, entendida como una estructura compleja de componentes que trabajan cooperativamente para proveer un servicio (hardware que lo contiene, los clientes que lo utilizan, los servicios que se ofertan…) (Fernandez A., 2010).

Los directorios de servicios ofrecen una estructura compartida de información para localizar, operar, manejar y organizar los componentes comunes de ésta (típicamente servicios).

En el ámbito de este proyecto, el Directorio provee una interfaz de servicio web a través de la cual se puede acceder a las entradas de servicios que son provistas en dicho directorio. El servicio, es por tanto, la unidad atómica del Directorio, permitiendo así a los clientes registrar, eliminar, modificar o buscar servicios en éste.

1.2. Arquitectura de servicios REST

Rest (*Representational State Transfer*) es un estilo arquitectónico para sistemas hipermedia distribuidos, definido por *Roy Fielding* en el año 2000.

Se concibió como un estilo arquitectónico que tratase de dar respuesta a cómo debería funcionar la Web, y se basa en los siguientes principios (Fielding R. , 2000) (Saugar, 2010):

- Un recurso es cualquier cosa que tenga identidad y debe estar identificado mediante algún mecanismo que no muestre detalles de su implementación. Para la identificación se utilizan URIs (*Universal Resource Identifier*), que son identificadores unívocos dentro de la web.

- Todos los recursos deben compartir la misma interfaz uniforme. Los métodos definidos en esta interfaz deben ser utilizados acorde a la semántica que se les impuso al crearlos.

- El formato de las representaciones debe estar documentado y ser estándar.

- La comunicación debe ser autocontenida. Hace referencia a que el servidor no deberá guardar estado (*stateless*), sino que será la propia petición la que dé la información completa sobre la operación a realizar.

Estos principios permiten por tanto crear un sistema por capas con características deseables como: sencillez, escalabilidad, rendimiento, evolucionabilidad, bajo acoplamiento.

Rest es además una filosofía para la implementación de servicios web basados en *HTTP*, ofrece por tanto una semántica específica a cada uno de los cuatro métodos básicos de *HTTP* 1.0:

- *PUT*: actualiza la información de un recurso alojado en determinada URI.
- *DELETE*: borra el recurso alojado en la URI especificada.
- *POST*: añade un recurso
- *GET*: obtiene una representación de un recurso (en un formato compatible).

En el ámbito de este proyecto se ha utilizado el estilo arquitectónico *Rest* para la definición tanto del servicio de directorio de servicios, como para la interfaz de acceso al gestor de contexto. En el caso del servicio de directorio, éste se encuentra definido acorde a una estructura de recursos para el acceso a los distintos servicios web, ofrecidos por la infraestructura. En futuros apartados, se explicará en detalle cómo se ha diseñado dicho directorio de servicios, ofreciendo distintas interfaces según el tipo de servicio al que se accede, así como en función de la tecnología utilizada para la descripción de los servicios.

El gestor de contexto por su parte, contiene una serie de recursos que permiten el consumo de información contextual.

1.3. Modelos de confianza

En el ámbito de entornos ubicuos y abiertos, se ha hecho necesaria la presencia de mecanismos de confianza y reputación como herramientas de control para el problema de la selección de proveedores con los que interactuar. La existencia de mecanismos de confianza, viene por tanto motivada porque los servicios pueden ser provistos por multitud de proveedores, que ofrecen dichos servicios de formas muy diferentes.

Para llevar a cabo la recomendación de servicios adaptados al usuario y a su contexto, se hace indispensable la presencia de un mecanismo de confianza, que permita realizar una eficiente selección de proveedores de servicio para el usuario, usando por una parte información almacenada de experiencias pasadas, así como conocimiento inferido por el propio agente sobre el potencial comportamiento del proveedor de servicios.

Este proyecto está enfocado a la creación de un agente recomendador de servicios que hará uso del módulo de confianza para la elección de proveedores de servicios. A través de las distintas valoraciones de los usuarios recomendados que hayan tenido interacciones previas con determinado proveedor, harán que el mecanismo de confianza tenga

retroalimentación y pueda de esta manera utilizar estas experiencias pasadas para generar expectativas sobre la ejecución de servicios futuros, proveyendo así información para recomendar el mejor de los proveedores posible.

El módulo de confianza recibe como entrada una lista de proveedores, sobre los cuales tendrá o no información pasada. En función, tanto de la posible información pasada, así como de mecanismos de inferencia, tratará de abordar el problema de la selección de proveedor. La existencia de dicho mecanismo de confianza genera valor añadido para el usuario, puesto que por una parte, soluciona el problema de la existencia de proveedores deshonestos y, por otra, gracias a la introducción de este mecanismo de confianza en el sistema, el asistente puede discernir qué proveedor se espera sea el mejor ofreciendo el servicio (Billhardt, Hermoso, & Centeno, 2007).

1.4. Dispositivos móviles

La evolución de las tecnologías en los últimos tiempos ha hecho posible la existencia de dispositivos cada vez más potentes. Dicha evolución de las *TIC* (Tecnologías de la información y la comunicación) ha provocado un vertiginoso aumento del potencial de los dispositivos móviles, capaces de ofrecer servicios de valor para el usuario.

La expansión de la movilidad ha hecho posible plantearse nuevas formas de interacción entre la persona y el terminal. Este proyecto, pretende hacer frente a esta realidad, dando respuesta a las cambiantes necesidades detectadas en la sociedad.

Como ya se ha hecho referencia previamente, el agente asistente inteligente mIO! se encuentra embebido en un dispositivo móvil, a través del cual se permiten satisfacer las necesidades detectadas para el usuario.

El dispositivo móvil será por tanto la herramienta que hará de interfaz entre el usuario y el entorno. Permitirá por una parte, percibir el contexto a través del cual se generarán los objetivos acordes a las preferencias del usuario y por otra parte, será el que haga de interfaz en la comunicación con el directorio de servicios.

Por tanto, el terminal móvil se convierte en un facilitador que permite poner en relación servicios con el entorno relevante, el contexto, gracias a la información que proporcionará el propio terminal móvil o la interacción con otros dispositivos y lugares. Todo ello, basado en las capacidades de los teléfonos móviles, que están evolucionando a potentes ordenadores de bolsillo con las máximas capacidades de comunicación.

1.4.1. Android

Para el desarrollo de la aplicación en dispositivo móvil, se ha optado por utilizar *Android* como plataforma para dispositivos móviles.

Android es una plataforma de código abierto para dispositivos móviles que está basada en Linux, fue desarrollado por la *Open Handset Alliance* (2007) un conglomerado de fabricantes y desarrolladores de hardware, software y operadores de servicio.

En la elección de esta plataforma se han valorado muy positivamente los siguientes factores:

- El lenguaje de programación orientado a objetos más ampliamente utilizado en este sistema operativo es Java. El uso de este lenguaje permite desarrollar software fuera del dispositivo, probarlo en plataformas de desarrollo clásicas y después reutilizarlo en la plataforma móvil, sin apenas modificaciones.

- Para el desarrollo sobre *Android*, se hace uso de plataformas y entornos de desarrollo clásicos. Además, desde la empresa desarrolladora se intenta fomentar el incremento de la comunidad de desarrolladores con gran cantidad de documentación y herramientas.

- La empresa Google Inc. respalda el proyecto, con una gran proyección de futuro.

Además, *Android* en los últimos años se ha consolidado como uno de los grandes sistemas operativos para dispositivos móviles, tras desbancar a otros sistemas operativos móviles como *Symbian* de Nokia.

En el siguiente gráfico se puede observar cómo en tan sólo dos años, *Android* (en color verde) ha pasado de tener una escasa cuota de mercado del 1% al 33%.

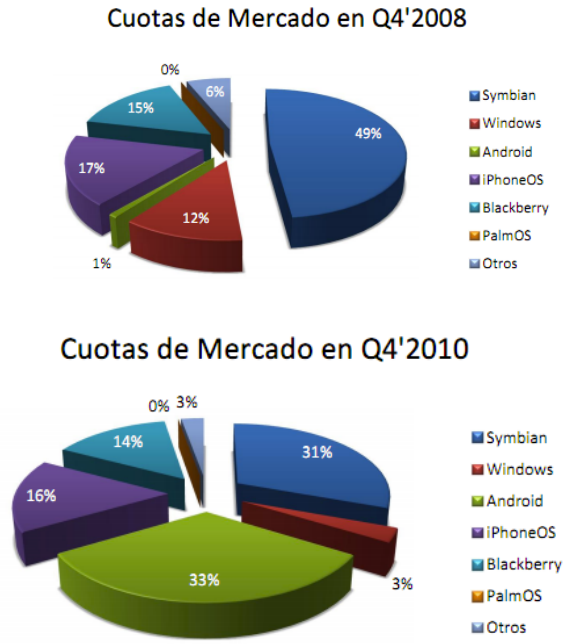

Figura 2 - Comparación de cuotas de mercado de sistemas operativos móviles período 2008 - 2010

Por lo que, esto confirma en cierta manera que en los últimos años, se está apostando por este sistema operativo.

2. Objetivos

En este apartado se exponen cuáles son los principales objetivos que han motivado la creación de este proyecto. Para ello, en primer lugar se expone un caso de estudio, a través del cual se pretende poner en situación de qué es lo que el agente recomendador presentado en este proyecto ofrece al usuario. Tras dicho caso de estudio, se expondrán cuáles son los objetivos a alcanzar en este Proyecto Fin de Carrera.

2.1. Caso de estudio

Para facilitar el entendimiento de la funcionalidad que ofrece el asistente recomendador que se presenta en este proyecto, se hará uso del escenario o caso de estudio que se expone en este apartado. Además, el caso de estudio que se presenta en este apartado ha sido utilizado como referencia dentro del proyecto mIO! para la validación de resultados.

Dicho escenario trata de poner de manifiesto en qué ámbito el asistente recomendador lleva a cabo el proceso de recomendación, así como la funcionalidad que éste brinda al usuario.

Este escenario se encuentra ambientado en la ciudad de Córdoba. Paula, una joven de Madrid acaba de llegar a Córdoba junto con su amigo Andrés, para realizar turismo durante un fin de semana. Una vez han llegado al hotel, y tras realizar un pequeño descanso, se proponen comenzar a visitar la ciudad de Córdoba antes de ir a comer.

Para ello, Paula hace uso de su dispositivo móvil que actúa como agente recomendador, el cual le aconseja ciertas acciones que puede llevar a cabo en éste contexto, entre ellas:

- Recibir información turística de “Tour de arte y cultura en Córdoba”
- Reservar partida de Golf en Córdoba
- Recibir información turística de “Visita a patios Cordobeses”
- Reservar mesa en un restaurante de comida típica de Córdoba
- Ir a casa de su amiga Andrea que vive en Córdoba

Puesto que es la primera vez que Paula visita Córdoba, le gustaría comenzar por una visita por los monumentos culturales más representativos de la ciudad. De las acciones ofrecidas por el asistente, la que mejor le conviene para satisfacer sus objetivos es la de “Recibir información turística de Tour de arte y cultura en Córdoba”. Paula decide que quiere realizar un tour, pero le pide a su asistente personal acciones más concretas, ya que la que le ofrece es demasiado genérica.

Tras esta petición, el asistente actualiza su lista de recomendaciones ofreciendo a Paula las siguientes:

- Recibir información turística de “Vista a Mezquita Catedral”
- Recibir información turística de “Visita a Templo Romano”
- Recibir información turística de “Visita a Medina Azahara”
- Reservar partida de Golf en Córdoba
- Recibir información turística de “Visita a patios cordobeses”
- Reservar mesa en restaurante de comida típica de Córdoba
- Ir a casa de su amiga Andrea

Deciden comenzar por visitar la Mezquita antes de la comida. En la pantalla del dispositivo móvil, conforme van realizando la visita, va apareciendo información turística relacionada con cada uno de los monumentos dentro de la Mezquita.

Para poder proveer de información a Paula, el dispositivo móvil ha precisado la comunicación con una plataforma o infraestructura en la que se encuentren todos los servicios disponibles. Para ello, se ha llevado a cabo una búsqueda de proveedores de servicios que provean información turística en tiempo real de la Mezquita Catedral de Córdoba.

Tras la ejecución del servicio, y por tanto de la recepción de información sobre la Mezquita Catedral, Paula deberá valorar al proveedor que le ha prestado dicho servicio. Esta valoración servirá para retroalimentar al mecanismo de confianza.

A partir de la información contenida en este escenario, se explican todos los elementos y funcionalidades que debe prestar el agente recomendador alojado en el dispositivo móvil de Paula. De hecho, así se verá reflejado en secciones posteriores cuando se haga referencia a la funcionalidad esperada por el sistema.

2.2. Objetivos del proyecto

Una vez descrita la idea central del proyecto y la motivación que ha llevado a construirlo, en este apartado se concretan los objetivos que a lo largo del mismo se desean alcanzar.

Objetivo 1. Realizar el diseño e implementación de un agente asistente inteligente.

Creación de un agente asistente inteligente para la recomendación proactiva de servicios en el contexto del proyecto mIO! a partir de una propuesta de arquitectura dada, permitiendo así prestar servicios ubicuos en un entorno inteligente y adaptado a cada individuo.

Para ello, es necesario llevar a cabo un proceso de creación de planes a partir de los objetivos del usuario, localizando además a los proveedores que ofrezcan servicios que satisfagan dichos objetivos, utilizando información de experiencias pasadas.

Objetivo 2. Analizar las distintas fases del proceso de recomendación.

El proceso necesario para llevar a cabo la recomendación de servicios acordes a las necesidades del usuario, será analizando tratando de poner en relevancia las fases que lo componen, así como los objetivos de cada una de estas.

Para ello, habrá que tener en cuenta las necesidades cambiantes del usuario, por lo que se atenderá no sólo a la definición de acciones a realizar, sino también a la posibilidad de refinamiento de las acciones, permitiendo la generalización o particularización de éstas, e incluso la eliminación de éstas.

Objetivo 3. Analizar la interacción e interfaz del agente recomendador con el usuario recomendado.

El agente asistente inteligente deberá poder interaccionar con el usuario con el fin de ofrecerle servicios de valor que satisfagan sus objetivos. Para ello, será necesario contar con una interfaz entre el agente y el usuario recomendado, para poder llevar a cabo la interacción.

Objetivo 4. Realizar el diseño de la interfaz de usuario del agente recomendador centrado en plataformas móviles.

Puesto que el proyecto se basa en la idea de movilidad, se pretende diseñar un prototipo adaptado para dispositivos móviles, donde resida el agente asistente inteligente, así como una interfaz de usuario para la interacción con éste.

Objetivo 5. Generar una implementación de referencia.

Con todo ello, crear una implementación de referencia del diseño del agente recomendador y un prototipo del mismo, junto con su interfaz, para una plataforma de dispositivos móviles.

3. Descripción Informática

En este apartado el objetivo es dar una descripción explícita del software desarrollado en este proyecto. Siguiendo la metodología empleada, se expondrán las diferentes vistas: análisis, arquitectura, diseño y la implementación de la aplicación.

3.1. Metodología empleada

La metodología seguida para desarrollar el software de este proyecto es similar a la propuesta por el Proceso Unificado para cada iteración de un modelo Iterativo, es decir, especificación de requisitos, análisis, diseño, implementación y pruebas.

Para dotar de una mayor flexibilidad a la construcción del sistema, y obtener retroalimentación al finalizar cada una de las iteraciones, se ha combinado la utilización del Proceso Unificado de Desarrollo con un desarrollo guiado por prototipos.

De forma genérica, partiendo de la creación de un prototipo inicial que contiene todos los requisitos básicos, se ha ido revisando y desarrollando un plan para el siguiente incremento, decidiendo así qué funcionalidades añadir. Dicho plan afronta la modificación del producto central a fin de cumplir mejor las necesidades del cliente y la entrega de funciones y características adicionales. Este proceso se repite siguiendo la entrega de cada incremento, hasta que se elabore el producto completo.

Esta combinación de ambas metodologías, permite obtener una visión inicial funcional para poder evaluarla y añadir mejoras, adaptadas a los requisitos del cliente.

Durante el desarrollo del producto, se han seguido las fases típicas de la metodología propuesta por el Proceso Unificado: requisitos, análisis, diseño, implementación y pruebas.

En la fase de requisitos se recogieron una serie de funcionalidades que se esperaba cumpliese el sistema final y que han sido descritas anteriormente en el apartado 2.2. En la siguiente fase de análisis, se llevó a cabo un esbozo inicial que guiaría posteriormente el diseño de la aplicación.

En la fase de diseño, tras un riguroso estudio de las tecnologías candidatas a ser utilizadas, se procedió al diseño del esbozo que se había realizado previamente en análisis. Este diseño es por tanto, una ampliación del análisis, pero ofreciendo un diseño dependiente de la plataforma y del lenguaje que sería utilizado posteriormente en la implementación del sistema. Dentro del diseño, hay que tener en cuenta que por una parte, se ha realizado el diseño concreto de la interfaz de usuario (tanto en dispositivo móvil, como una interfaz adaptable a cualquier tipo de dispositivo) y por otra parte, el diseño de la funcionalidad del agente asistente.

En la fase de implementación y pruebas, además de la codificación en sí del sistema, se han llevado a cabo las pruebas necesarias para comprobar el correcto funcionamiento del

sistema y corroborar así que presenta la funcionalidad esperada por los usuarios. De forma, que la fase de pruebas se ha orientado hacia la validación del sistema, a través de entrevistas con los clientes del sistema.

A partir de estas pruebas realizadas, se planifica la siguiente iteración y por tanto, poder incrementar la funcionalidad del prototipo a través de la retroalimentación generada por los clientes del sistema.

Como parte de la metodología de desarrollo, todo el análisis, diseño e implementación de la aplicación se ha realizado utilizando el Lenguaje Unificado de Modelado (*Unified Modelling Language, UML*) (Booch, Rumbaugh, & Jacobson, 1999). UML es un lenguaje de modelado que proporciona un vocabulario y unas reglas para la representación conceptual y física de un sistema, lo que nos va a permitir tener varias vistas o modelos que nos ayudarán a comprender las distintas funcionalidades del sistema.

En definitiva, UML cubre la especificación de todas las decisiones de análisis, diseño e implementación que deben realizarse al desarrollar y desplegar el sistema software.

3.2. Arquitectura de agentes inteligentes de recomendación

En este apartado de Arquitectura, se trata de dar una visión de alto nivel en la que se pueda determinar de manera precisa la estructura de ésta y permitir por tanto, determinar los subsistemas, componentes y relaciones que hay entre ellos.

Dicha arquitectura, en el ámbito del presente proyecto, viene dada, por lo que este apartado se limita a explicar la estructura de ésta, tal y como se verá en la Figura 5.

3.2.1. Agenda

En la Figura 3, se expone de forma general la funcionalidad que va a desempeñar el agente inteligente dentro del proyecto. El agente inteligente, alojado en el dispositivo móvil, actuará como un asistente personal del usuario, facilitando el acceso a todo el universo de servicios disponibles, así como permitiendo el acceso a la información contextual disponible en cada momento. Para ello, el asistente captará la información de contexto y a partir de ella, creará planes de acción (en adelante, acciones) que satisfagan los objetivos del usuario.

Por tanto, en el marco de este proyecto, se define *acción* como la instrumentación de los objetivos que el usuario pretende alcanzar a través de la recomendación del agente. Estas acciones incluirán por una parte, los objetivos que satisfacen, así como las plantillas de servicio que permiten llevar a cabo dichos objetivos.

Para llevar a cabo este fin, el asistente personal se comunicará con el usuario utilizando una agenda de acciones. Esta agenda incluye una serie de acciones que el usuario desea o tal vez desearía realizar.

Estas acciones, tal y como se puede observar en la Figura 3, pueden ser añadidas bien a través de propuestas proactivas del agente o bien bajo petición del propio usuario. En este Proyecto Fin de Carrera se ha decidido tan sólo tener en cuenta las propuestas por el asistente personal, en base a las preferencias del usuario y a las actividades anteriormente aprendidas, así como con la información contextual, dejando por tanto fuera del proyecto las posibles peticiones del usuario.

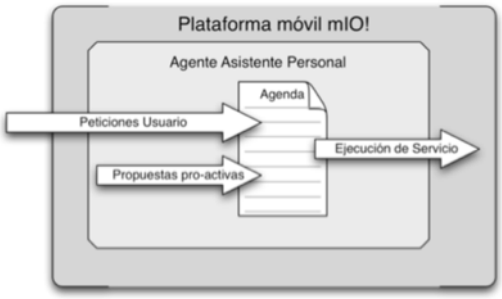

Figura 3 - Funcionalidad general del agente inteligente mIO!

Por tanto, actualmente la forma en la que se rellenan las acciones en la agenda, es a partir de las propuestas proactivas realizadas por el agente recomendador. De forma general, el agente recomendador es concebido como un agente capaz de actuar de manera autónoma, proactiva y flexible con el fin de conseguir sus objetivos (en este caso, satisfacer los del usuario). Además, dicho agente inteligente muestra un comportamiento proactivo orientado a sus propios objetivos, tomando la iniciativa para satisfacer sus fines.

Una vez generadas de forma proactiva las acciones disponibles en la agenda del asistente personal, el usuario podrá seleccionar una en concreto con la intención de querer llevarla a cabo o realizar otras gestiones sobre dicha acción (refinarla). La ejecución de la acción conllevará la ejecución de los servicios asociados a la acción, con el proveedor de servicios que el asistente estime oportuno (siempre en base a las preferencias del usuario y a sus experiencias pasadas).

Para llevar a cabo la funcionalidad básica anteriormente detallada, se va a proceder a dividir el proceso de recomendación en un total de cuatro fases: búsqueda de objetivos, planificación de acciones, búsqueda de servicios y ejecución de servicios.

Aunque el objetivo final es la creación de un asistente personal para recomendar al usuario, la división del problema en distintas fases permite la reutilización de cada uno de estos elementos de la arquitectura de forma independiente y su posible aplicación para la resolución de otros problemas similares.

Las fases en las que se divide el proceso de recomendación, se traducen en módulos que forman parte de la arquitectura y se detallan a continuación:

- ***Búsqueda de Objetivos***: componente encargado de detectar las necesidades del usuario y expresarla en forma de objetivos. Estos objetivos no serán más que ciertas características del contexto que se desea tomen un cierto valor deseado.

- ***Planificación de Acciones***: componente que se encarga de transformar las necesidades del usuario (objetivos) en planes factibles (acciones) compuestos por plantillas de servicios (o descripciones de servicios) u otros planes disponibles en el universo de servicios.

- ***Búsqueda de Servicios***: componente encargado de buscar instancias de servicios (con proveedor asociado) a ejecutar para poder transformar una acción en ejecutable. Parte de los criterios de selección de dichos servicios están basados en experiencias pasadas.

- ***Ejecución de Servicios***: componente encargado de invocar la ejecución de una instancia de servicio y además de recoger toda la información para alimentar los sistemas basados en experiencias pasadas.

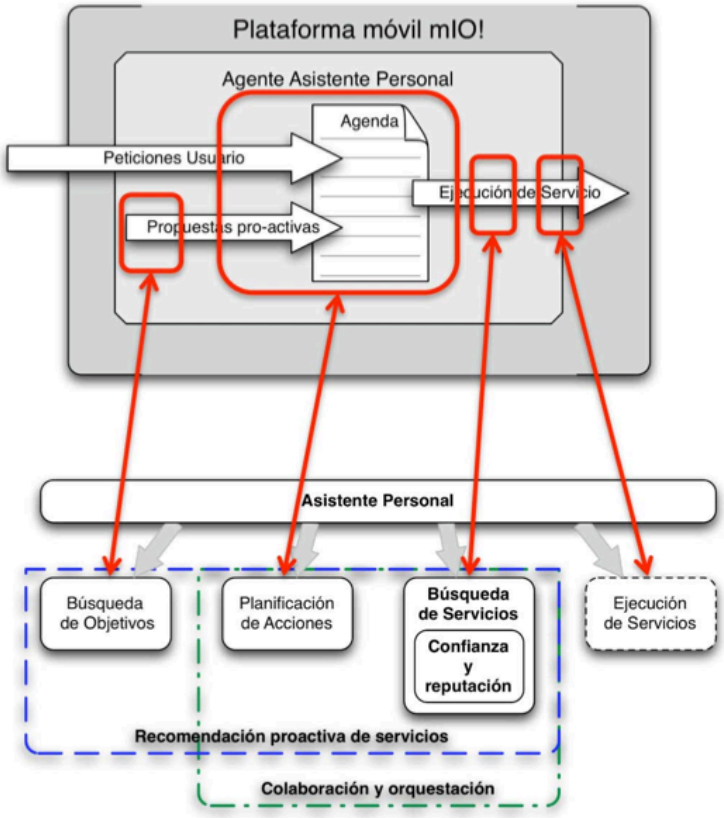

Figura 4 - Relación del asistente inteligente con la arquitectura de agentes inteligentes

En la Figura 4 se muestra cómo se relaciona cada uno de los componentes propuestos en la arquitectura inicial con las funcionalidades de los asistentes personales descritas anteriormente. Puede apreciarse cómo parte de los elementos de la arquitectura sólo entran en juego cuando se requiere una recomendación proactiva de acciones (no así cuando son especificadas por el usuario). De igual manera se puede observar cómo se le permite al usuario seleccionar una acción, pero la selección del proveedor concreto para dicha acción se lleva a cabo de forma automática por parte del asistente personal.

3.2.2. Arquitectura propuesta

A continuación, se va a realizar una descripción detallada de la arquitectura propuesta para la instrumentación de los agentes inteligentes, así como profundizar en las distintas funcionalidades de cada uno de sus componentes.

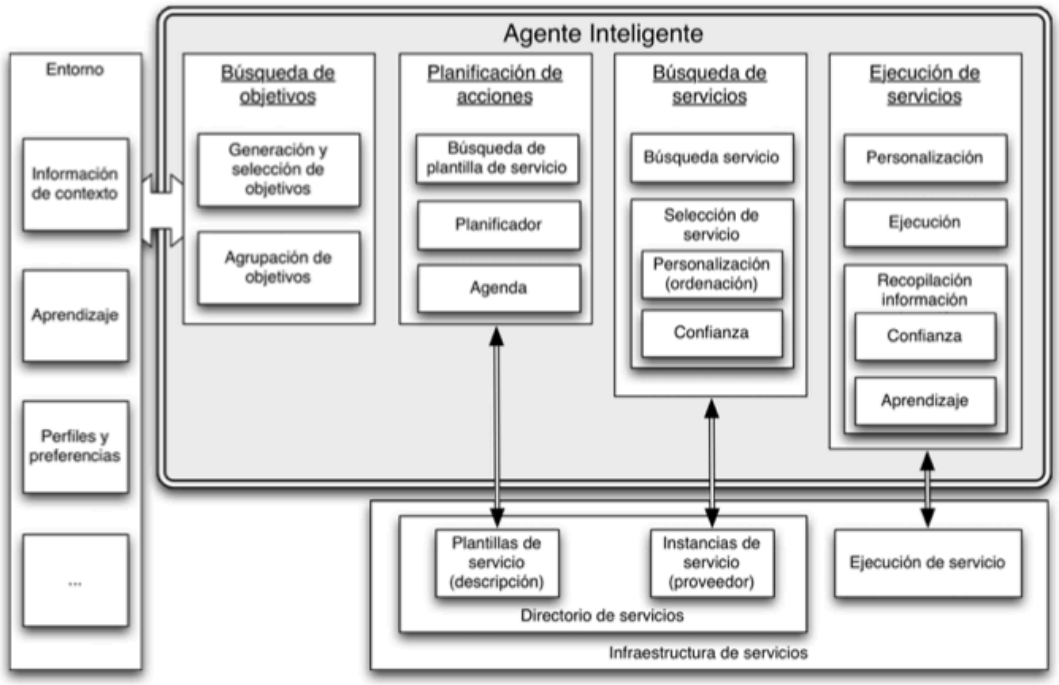

Figura 5 - Detalle de arquitectura de agentes inteligentes

Tal y como se ha puntualizado anteriormente, el agente inteligente está rodeado de lo que llamamos entorno. Por entorno de usuario entendemos toda la información que puede ser relevante en un momento dado para el usuario. Esta información incluye todo lo relacionado con la información contextual, aprendizaje de secuencias de acciones pasadas, información referente a las preferencias y perfiles, así como cualquier otra información disponible que pueda ser de utilidad para identificar las necesidades del usuario.

Por otra parte, otro de los elementos externos al agente es la infraestructura de servicios, que pone a disposición de los agentes inteligentes la búsqueda de servicios (tanto de plantillas como de instancias). Además, dicha infraestructura también proporciona la plataforma para la ejecución de servicios. El agente inteligente utilizará toda la información del servicio de directorios para componer planes de acciones así como para seleccionar proveedores concretos para la ejecución de las acciones seleccionadas por el usuario.

Dentro de la arquitectura del agente propiamente dicho, existen cuatro grandes bloques cuya funcionalidad se describe a continuación en detalle.

3.2.2.1. Búsqueda de objetivos

El módulo de búsqueda de objetivos es el encargado de recopilar toda la información contextual, de preferencias de usuario, secuencias aprendidas y toda aquella información relevante, para la identificación de objetivos que el usuario asesorado podría querer realizar.

De entre toda la información recopilada, se seleccionará la más relevante según el perfil del usuario y el rol que esté desempeñando en un momento dado, detectando las posibles necesidades. Éstas, se expresarán como descripciones de características que representen estados del mundo que desean alcanzarse. De forma teórica en el marco del proyecto mIO! se plantea la necesidad de tener en cuenta el rol concreto que toma el usuario en cada momento, pero es un aspecto que no se ha tenido en cuenta en el desarrollo del presente Proyecto Fin de Carrera.

Además del proceso de generación de objetivos, éstos también se agrupan según similitudes en su definición. De esta manera, en pasos posteriores se puede conseguir que un plan de acción sea capaz de conseguir alcanzar múltiples objetivos relacionados.

3.2.2.2. Planificación de acciones

La planificación de acciones recibe como entrada una serie de descripciones de estados del mundo que se desean alcanzar (objetivos). A partir de estas definiciones, se trata de crear acciones factibles para que puedan ser ejecutadas.

Para la creación de dichas acciones, primeramente se busca qué plantillas de servicio pueden emplearse para alcanzar algunos de los objetivos propuestos. Para ello deberá hacerse uso de la infraestructura de servicios, la que mediante un proceso de alineamiento semántico, determine qué plantillas de servicio satisfarían las necesidades del usuario expresadas en forma de objetivos.

Con las plantillas de servicios obtenidas y con los objetivos detectados en la fase de búsqueda de objetivos, se procede a realizar la planificación de acciones que serán mostradas al usuario a través de la agenda.

3.2.2.3. Búsqueda de servicios

La búsqueda de servicios se desencadena cuando el usuario del asistente personal selecciona una de las acciones propuestas. A partir de este momento se procede a seleccionar un proveedor específico para la ejecución de la acción. Para ello, se utiliza la infraestructura de servicios para la búsqueda de instancias de servicios.

Con los posibles proveedores del servicio obtenidos de la plataforma, se realiza una selección, teniendo en cuenta las preferencias y la experiencia pasada del usuario. Las experiencias pasadas del usuario consisten en la agrupación de valoraciones por parte del usuario de servicios anteriormente consumidos.

A partir de estas valoraciones, organizadas por tipos de servicios y proveedores, se puede inferir cuál podría ser el resultado de consumir determinado servicio, pudiendo seleccionar así el servicio que más podría satisfacer al usuario.

3.2.2.4. Ejecución de servicios

Los agentes inteligentes no son los encargados directos de la ejecución de los servicios, ya que de ello se encarga la infraestructura de servicios.

En este punto, tras la ejecución del servicio, el agente inteligente recopila la información necesaria para el aprendizaje y pide al usuario que evalúe el servicio consumido, para mejorar la información respecto al nivel de confianza y satisfacción del usuario con dicho servicio.

3.3. Análisis

El objetivo de la fase de análisis es la realización de una especificación más precisa de los requisitos, a través de un diagrama de casos de uso del sistema.

Por otra parte, se construye el esbozo general de lo que será el sistema, facilitando así la comprensión, preparación, modificación y mantenimiento de los requisitos, creando una primera aproximación del modelo de diseño.

3.3.1. Casos de uso generales del sistema

Un caso de uso es una descripción de un conjunto de secuencias de acciones, incluyendo variantes, que ejecuta un sistema para producir un resultados de valor para un actor (Booch, Rumbaugh, & Jacobson, 1999).

Los diagramas de casos de uso se emplean para modelar la vista de casos de uso de un sistema. La mayoría de veces, esto implica modelar el contexto del sistema, subsistema o clase, o el modelado de los requisitos de comportamiento de estos elementos.

Estos diagramas tienen gran importancia a la hora de visualizar, especificar y documentar el comportamiento de un elemento. Facilita que los sistemas, subsistemas y clases sean abordables y comprensibles, al presentar un vista externa de cómo pueden utilizarse estos elementos en un contexto dado.

A continuación, se puede observar el diagrama de casos de uso que aglutina la funcionalidad esperada por la aplicación. Tras el diagrama se especifican brevemente cada uno de estos casos de uso y posteriormente, se analiza más en detalle las interacciones que se producen en cada uno de ellos.

Como se puede observar en el diagrama de la Figura 6, existen dos agrupaciones de casos de uso: por una parte los referidos a la funcionalidad del agente recomendador y por otra parte, las aglutinadas bajo el paquete *agenda* que hacen referencia a otras interacciones que se pueden realizar en la agenda de acciones, es decir, toda la funcionalidad que se puede realizar desde la interfaz del dispositivo móvil.

Existen dos posibles actores en el sistema, por una parte, el *usuario recomendado* que es el que representa al usuario del sistema que en este caso concreto corresponde al humano. Este será el actor principal del sistema, puesto que el proceso de recomendación girará en torno a él.

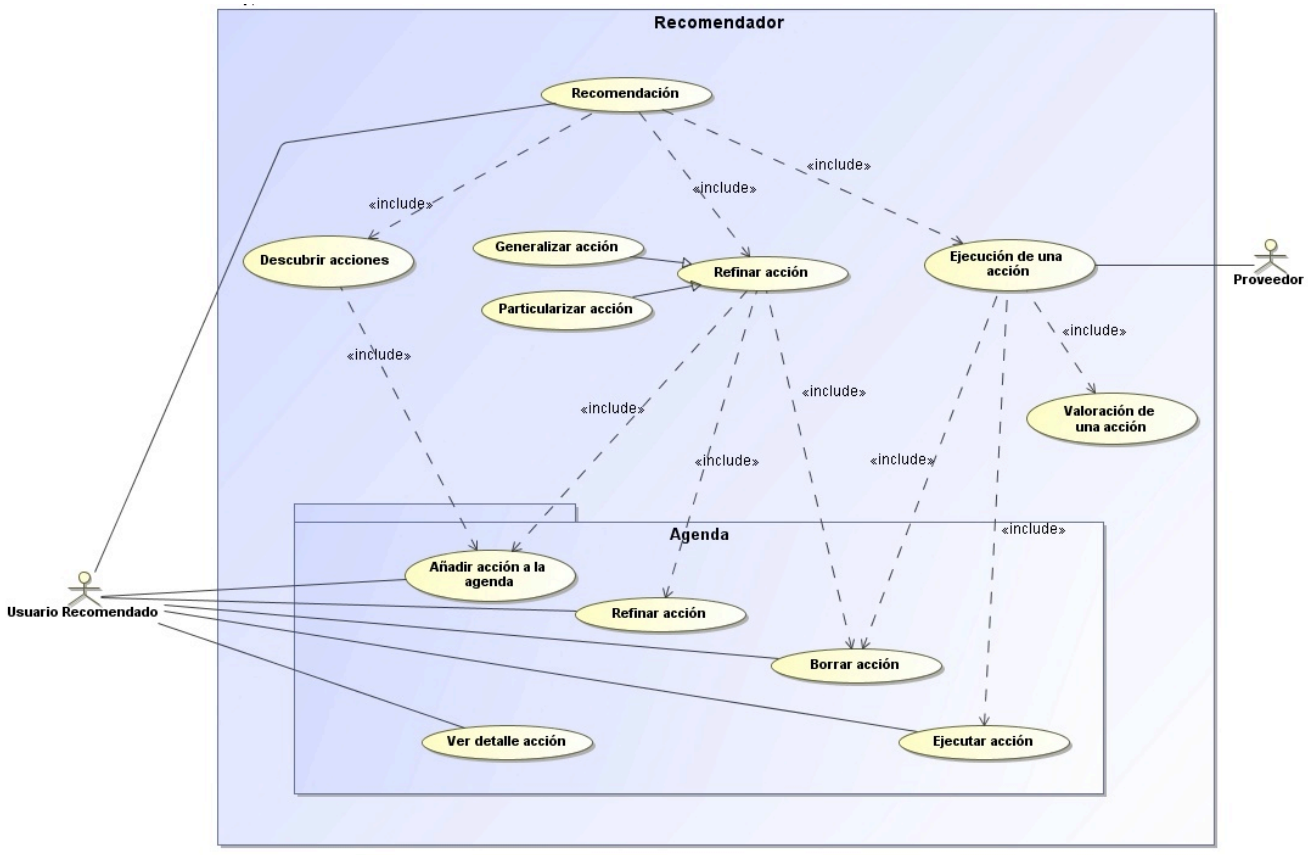

Figura 6 - Diagrama de casos de uso

Por otra parte, otro actor participante en el sistema es el *proveedor de servicios*, que forma parte del sistema en tanto en cuanto, el servicio es ejecutado por éste de forma externa al agente recomendador. Este actor participará por tanto, en los casos de uso relacionados con la ejecución de las acciones, así como en la publicación de instancias de servicio en la infraestructura.

A continuación, se detallan cada uno de éstos casos de uso, indicando qué actor o actores participan en él, así como una breve descripción de la funcionalidad que ofrece dicho caso de uso.

3.3.1.1. Recomendación

Actor

El usuario recomendado.

Descripción

Este caso de uso es el que aglutina todo el proceso de recomendación. Hace referencia al flujo compuesto por los casos de uso “descubrir acciones” y “ejecución de una acción”, así como la posibilidad de refinamiento de una de las acciones.

3.3.1.2. Descubrir acciones

Actor

El usuario recomendado

Descripción

El caso de uso comienza cuando el usuario ejecuta la aplicación y tras percibir el contexto actual, se crean una serie de objetivos (expresados en términos de cómo desea el usuario sea el estado del mundo tras la ejecución de un servicio) que serán instrumentados en acciones, que satisfagan las plantillas de servicio encontradas que cumplen dichos objetivos.

Una vez creadas estas acciones que cumplen los objetivos deseados, se añadirán a la agenda para ser mostrada al usuario.

3.3.1.3. Refinar acción

Actor

El usuario recomendado

Descripción

Por cada una de las acciones que se muestran en la agenda, es posible llevar a cabo un proceso de refinamiento de la acción. Entendiendo por refinar, la particularización o generalización de dicha acción.

Es decir, si la acción mostrada inicialmente en la agenda, no es exactamente la que el usuario quiere para satisfacer sus objetivos, es posible agrupar una serie de acciones, o bien disgregar una acción en varias que satisfagan los mismos objetivos. De forma, que si el usuario escoge la opción de refinar acción, comenzará un proceso de refinamiento, tras el cual, el estado de las acciones de la agenda cambie, mostrando al usuario las nuevas acciones.

3.3.1.4. Ejecución de una acción

Actor

En este caso de uso participan dos actores:

- el usuario recomendado que solicita la ejecución de una acción
- el proveedor de servicios, encargado de la ejecución del servicio y de la publicación de las plantillas de servicio.

Descripción

El usuario selecciona la opción de ejecutar una acción. La ejecución se realiza fuera del dispositivo móvil, más concretamente en la infraestructura de servicios, por lo que

internamente este caso de uso tan sólo implica que el dispositivo ordena la ejecución del servicio y a continuación, se lleva a cabo la valoración del proveedor de servicio, dando la retroalimentación necesaria al módulo de confianza.

Este caso de uso incluye por tanto, el caso de uso de valoración de la acción.

3.3.1.5. Añadir acción a la agenda

Actor

El usuario recomendado

Descripción

Esta funcionalidad permite añadir una acción a la agenda o lista de acciones que se presentan al usuario en la interfaz del dispositivo.

3.3.1.6. Borrar acción de la agenda

Actor

El usuario recomendado

Descripción

El usuario a través de la interfaz, escoge la funcionalidad de eliminar una acción de la agenda. Además, este caso de uso implica que una vez borrada la acción de la agenda, se elimine la posibilidad de poder aparecer de nuevo en la agenda de acciones.

3.3.1.7. Ver detalle de la acción

Actor

El usuario recomendado

Descripción

A petición del usuario, a través de la interfaz del dispositivo móvil, se pueden consultar los detalles de una acción.

3.3.2. Diagramas de estado para cada caso de uso

Un diagrama de estados identifica cada una de las rutas o caminos que puede tomar un flujo de información, poniendo énfasis en la distribución temporal de dicho flujo.

En esta sección se muestran los diagramas de estado correspondientes a cada uno de los casos de uso descritos anteriormente.

3.3.2.1. Caso de uso “Recomendación”

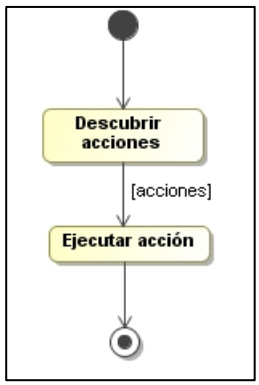

Figura 7 - Diagrama de estados de "recomendación"

El diagrama de la figura anterior hace referencia al proceso genérico de recomendación llevado a cabo por el agente recomendador.

Tal y como se puede observar, está compuesto de dos casos de uso, por una parte el descubrimiento de las acciones a partir de la información contextual y del entorno, y por otra parte, la ejecución de la acción que haya resultado seleccionada por el usuario recomendador.

Los citados casos de uso, serán explicados con más detalle en secciones posteriores.

3.3.2.2. Caso de uso “Descubrir acciones”

La Figura 8 se corresponde con el diagrama de estados del caso de uso *Descubrir acciones en la agenda*. El flujo se inicia cuando se entra en el estado *Generación de objetivos* que es el encargado de generar los objetivos que se pretenden alcanzar a partir del contexto (*Percibir contexto*) actual.

Una vez generados los objetivos que se pretenden satisfacer, se pasa al estado correspondiente a la generación de un plan concreto que cumpla dichos objetivos (*Crear plan*). Para poder crear estos planes (que se traducirán en acciones) se debe conocer cuáles son las plantillas de servicio (*descripción de servicio*) que satisfacen los objetivos. Esto se hará a través de una consulta al módulo externo llamado directorio de servicios.

Tras obtener las plantillas de servicio, se componen las acciones que se añadirán posteriormente en la agenda de acciones. No es objetivo de este proyecto, la explicación detallada del proceso de planificación de acciones, por lo que no se profundiza en este aspecto.

El estado “*Añadir acciones*” hace referencia al caso de uso de la agenda que permite añadir éstas a la lista de acciones de la agenda.

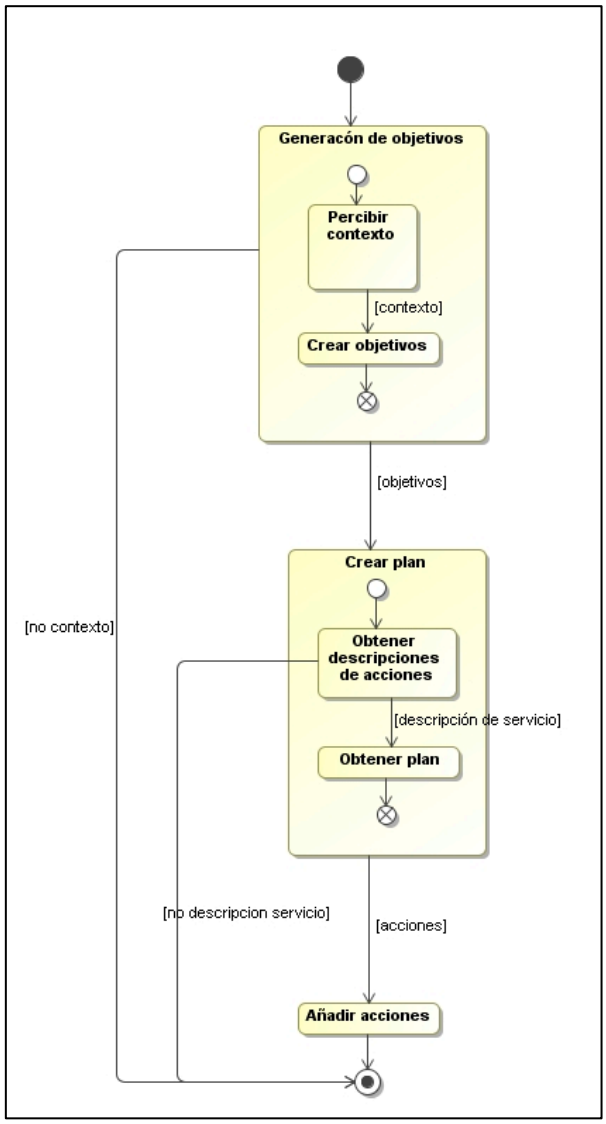

Figura 8 - Diagrama de estados de "Descubrir acciones"

3.3.2.3. Caso de uso “Ejecución de una acción”

El diagrama de estados anterior se corresponde con el caso de uso de *Ejecución de una acción* e implícitamente, la valoración del proveedor escogido en la ejecución de la acción.

El flujo comienza en el estado correspondiente a la elección en la agenda de la acción que se desea ejecutar. Tras dicha elección, se transita a un estado encargado de la búsqueda de instancias de servicio (*Obtener instancias de servicio*) que satisfagan los objetivos del usuario asociados a la acción seleccionada. La obtención de los proveedores de servicio, implica una consulta al directorio externo de servicios, tras la cual se obtendrá una lista de potenciales proveedores del servicio seleccionado, que deberán ser ordenados en términos de confianza del usuario (*Ordenar proveedores*).

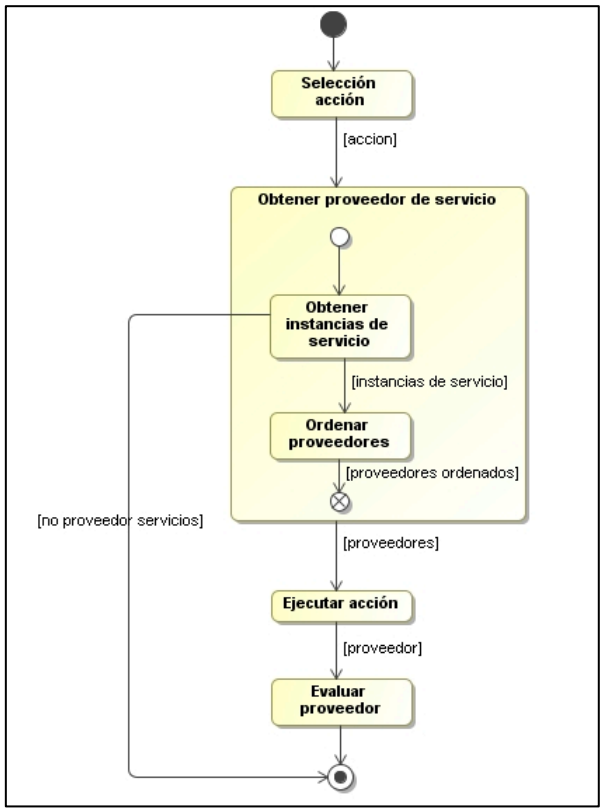

Figura 9 - Diagrama de estados de "Ejecución de una acción"

Una vez transitado al estado de ejecución del servicio seleccionado con el proveedor escogido previamente (hace referencia al caso de uso de la agenda *Ejecutar acción*) es obligatoria la valoración por parte del usuario del proveedor con el cual ha tenido la interacción. Dicha obligatoriedad viene derivada de la necesidad del modelo de confianza (el encargado de ordenar en términos de confianza los proveedores de servicio) de obtener una debida retroalimentación tras la ordenación de los proveedores, incrementando el volumen de información con respecto a experiencias pasadas del usuario.

3.3.2.4. Caso de uso "Refinar acción"

Este diagrama de estados se corresponde con el caso de uso de *Refinar acción*. Como ya se ha comentado anteriormente en este documento, existen diversas interacciones a realizar con una acción, siendo una de ellas la de refinar acción.

Refinar acción, implica que se realice bien una particularización (*zoom in*) de una acción seleccionada, o bien una generalización (*zoom out*). Una vez determinado el tipo de interacción a realizar, así como la acción concreta sobre la que se desea realizar, se procede de igual manera que en el caso de uso de *añadir acciones a la agenda*, es decir, se pasa a un estado en el que se obtiene a partir de los objetivos del usuario, un plan de acción que satisfaga dichos objetivos.

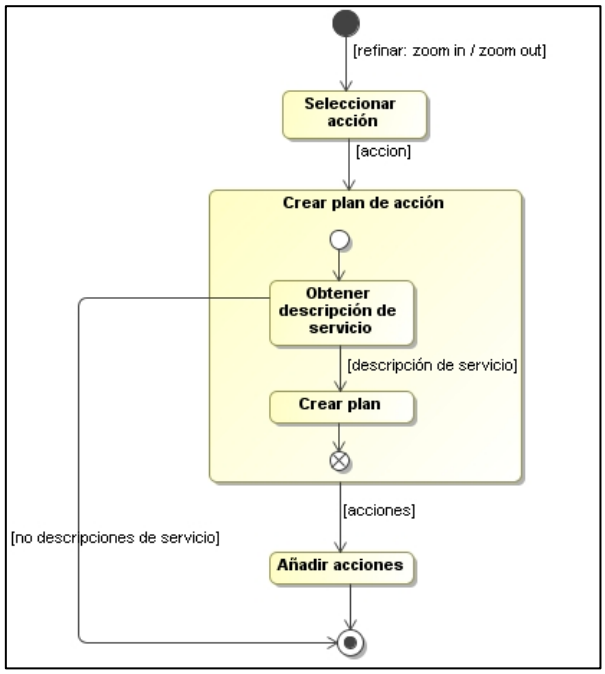

Figura 10 - Diagrama de estados de "Refinar acción"

Para ello, en primer lugar se transita al estado *Crear plan*, en el que se obtendrá a partir de las plantillas de servicio que satisfagan los objetivos, un conjunto de acciones que serán añadidas a la agenda (*Añadir acciones*). Al igual que en casos anteriores, el añadir la acción a la agenda hace referencia al caso de uso que permite la adición de nuevas acciones a la lista.

3.3.2.5. Casos de uso de la agenda

En los casos de uso anteriores, se ha visto a través del diagrama de estados, el comportamiento del sistema cuando se realiza alguna de las acciones genéricas (añadir acciones, refinar propuesta…).

A continuación, se verá cuál es el impacto en la agenda de acciones, de cada uno de los casos de uso que han sido agrupados en el diagrama de casos de uso bajo el paquete *Agenda*.

De forma, que cada uno de los diagramas que a continuación se muestran, reflejará qué acciones se realizan en la agenda por cada uno de los casos de uso explicados anteriormente.

Caso de uso de la agenda "Añadir acción a la agenda"

El caso de uso de *añadir acciones a la agenda*, comienza una vez que se ha determinado la nueva acción a añadir.

El siguiente estado al que transita hace referencia al caso de uso de "*Añadir acción*" de la agenda, que implica que la lista de acciones que contiene la agenda se verá incrementada con la nueva acción que se ha añadido.

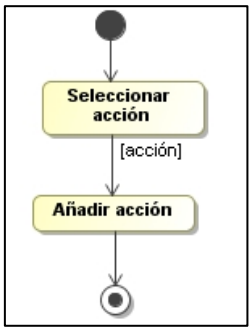

Figura 11 - Diagrama de estados de Agenda "Añadir acción a la agenda"

Caso de uso de la agenda "Borrar acción de la agenda"

El diagrama de estados de la Figura 12 indica el comportamiento de la agenda cuando se da el caso de uso de *Borrar acción de la agenda*. Como es de esperar, una vez seleccionada la acción a borrar (*Seleccionar acción*) se procede a la eliminación de la acción de la lista de acciones contenidas en la agenda, minorando así el número de acciones contenidas en ésta.

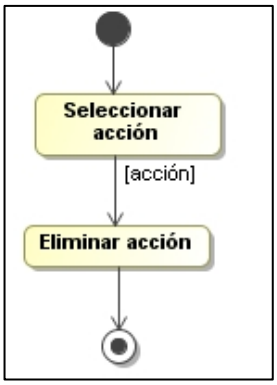

Figura 12 - Diagrama de estados de Agenda de "Borrar acción de la agenda"

Caso de uso de la agenda "Refinar acción de la agenda"

En la Figura 13 se puede ver el diagrama de estados del caso de uso de *refinar propuesta*, desde el punto de vista de la agenda.

El flujo comienza con la elección del tipo de interacción que se desea tener con la acción seleccionada: ya sea particularización o generalización.

Una vez seleccionada la acción (*Seleccionar acción*) es borrada de la agenda para no volver a ser ofrecida al usuario, y a continuación se realiza el proceso de refinamiento de la acción (tanto si es particularización como si se trata de generalización de la acción) y se generan las nuevas acciones que sustituyen a la anterior (*Obtener nuevas acciones*) y por último se añaden las acciones a la agenda para poder ser mostradas al usuario.

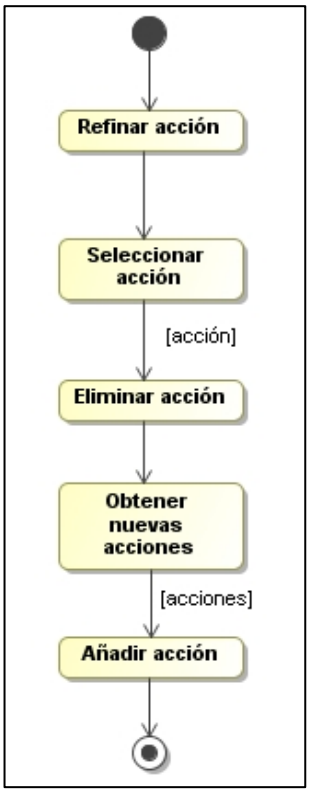

Figura 13 - Diagrama de estados de "Refinar propuesta"

Caso de uso de la agenda "Ver detalle de una acción"

El caso de uso de *Ver detalle de una acción* comienza con la selección de la acción en la agenda por parte del usuario y a continuación, se transita al estado correspondiente a la obtención de información adicional sobre la acción que se ha seleccionado previamente.

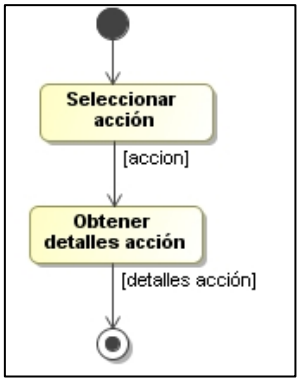

Figura 14 - Diagrama de estados de Agenda de "Ver detalle de una acción"

Caso de uso de la agenda "Ejecutar acción"

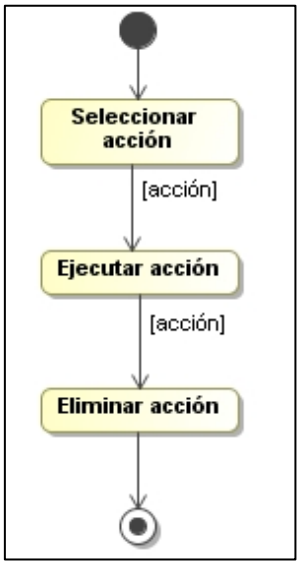

Figura 15 - Diagrama de estados de Agenda de "Ejecutar acción"

El caso de uso de *ejecución de una acción*, desde el punto de vista de la agenda de acciones, comienza cuando es seleccionada la acción a ejecutar y tras ser ejecutada en el módulo externo dedicado a la ejecución de los servicios, entonces es eliminada de la lista de acciones de la agenda.

3.4. Diseño

En esta sección, se muestra el modelo de diseño del sistema. Este modelo está centrado en la secuencia de acciones entre los distintos subsistemas de los que consta la aplicación, así como la interacción necesaria entre los distintos módulos para brindar al sistema la funcionalidad esperada.

El objetivo de esta sección es, observar la transformación que sufre toda la interacción descrita, en cuanto a lo que análisis y arquitectura se refiere, hacia un diseño concreto.

Este proyecto consta de dos partes bien diferenciadas, por lo que, el diseño de éstas se expondrá de forma separada: por una parte, lo referente tanto a la arquitectura como a la agenda de acciones y por otra, la parte relacionada con la implantación en plataformas móviles.

Antes de exponer el diseño de cada una de las partes, se analizará en detalle el diseño de la arquitectura general del sistema, a partir de la cual se ha basado toda la aplicación.

3.4.1. Diseño de la arquitectura general

Para el diseño e implementación del asistente inteligente y dado que interactúa con multitud de elementos externos (gestor contexto, infraestructura de servicios…) se hace necesaria la definición de una arquitectura software. Además, este concepto hará posible la reutilización de estructuras asociadas a agentes similares.

En este apartado se mostrará la arquitectura software del agente asistente, a través de la explicación de los componentes u elementos software, las propiedades visibles de los elementos y las relaciones entre ellos.

3.4.1.1. Diagrama de clases de la arquitectura del sistema

A continuación, en la Figura 16 se muestra un diagrama de clases general para la instrumentación del presente sistema, utilizando como base, la arquitectura propuesta en secciones anteriores. De forma general, se pueden observar cuatro grandes paquetes: infraestructura de servicios, infraestructura de contexto, codificación de datos que se intercambian entre ellos y el asistente personal (agente inteligente). De entre estos módulos definidos anteriormente, no es objetivo de este proyecto la definición de la infraestructura de servicios, ni del gestor de contexto, pero sin embargo, sí se ha llevado a cabo el diseño del asistente personal, así como la interfaz de comunicación con los componentes externos.

Dentro del asistente personal, se pueden distinguir claramente los componentes de la arquitectura propuesta en el apartado 3.2 del presente documento, así como cada una de sus funcionalidades.

En el resto de esta sección se procede a hacer una descripción detallada y pormenorizada de cada una de las clases que componen el diagrama de clases de la aplicación.

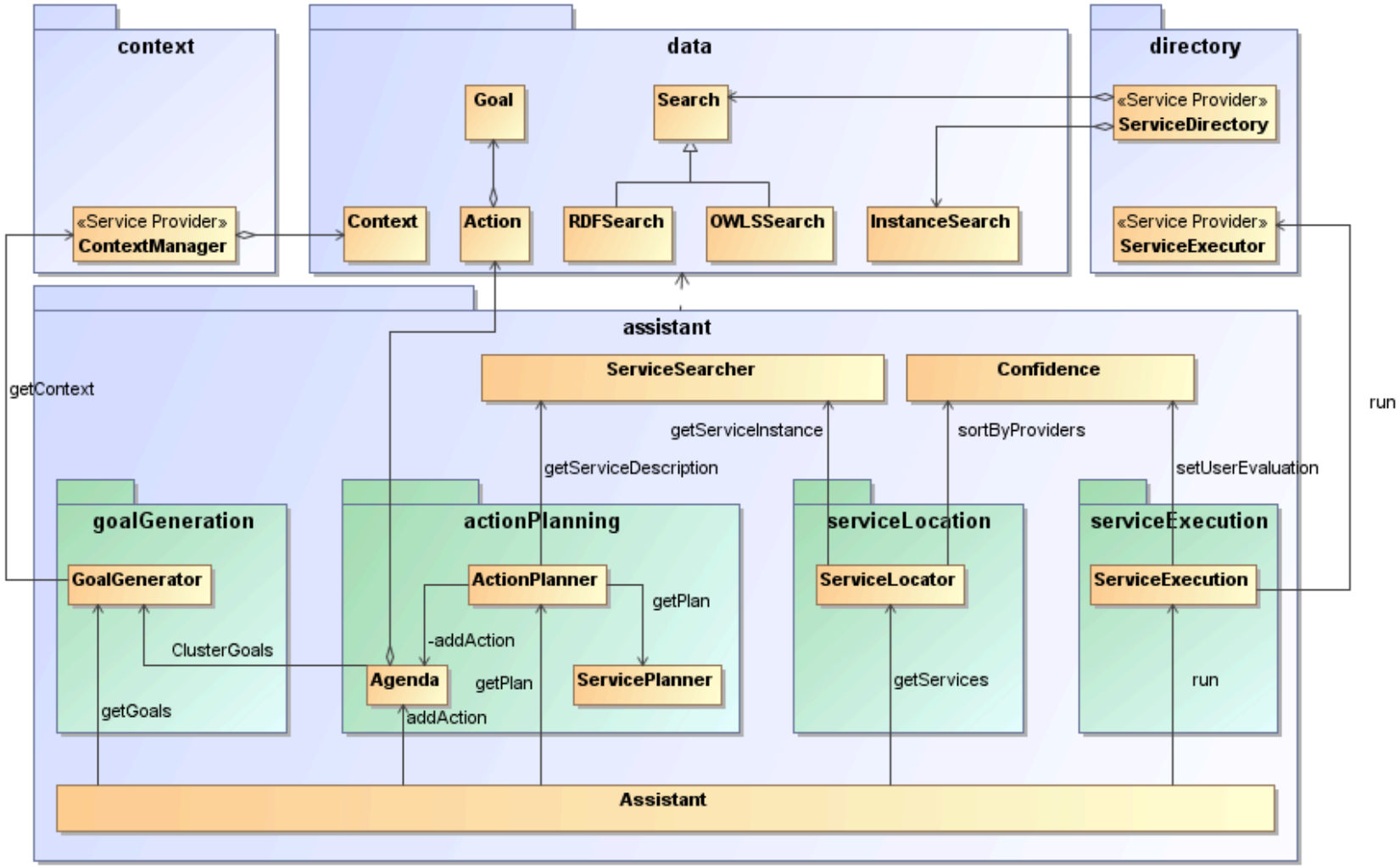

Figura 16 - Diagrama de clases general

Paquete data

Nombre de paquete: **data**

- Situación: Se trata de uno de los paquetes principales del diagrama de clases.
- Funcionalidad: Este paquete contiene todas las clases calificadas como "datos" y que se explicarán a continuación. Es la información base que se utilizará para el intercambio de información entre los distintos componentes.
- Relaciones: Al agrupar los tipos de datos básicos, tiene relación con todos los demás paquetes que forman parte del diagrama.

Clases

Nombre de clase**: Context.**

- Situación: Clase contenida en el paquete *data*.
- Funcionalidad: Una instancia de la clase *Context* representa un estado concreto del mundo.
- Relaciones: Se relaciona con el gestor de contexto (*ContextManager*), puesto que éste es el encargado de recabar dicha información del entorno.

  La información de contexto, es usada por el módulo generador de objetivos (*GoalGenerator*), para la búsqueda de objetivos.

Nombre: **Goal**

- Situación: Clase contenida en el paquete *data*.
- Funcionalidad: Un objetivo define el estado del mundo que se desea alcanzar en base a descripciones contextuales, o dicho de otra forma, indica qué valores de información de contexto se desean alcanzar en un futuro.
- Relaciones: Tiene relación directa con el generador de objetivos (*GoalGenerator*), que será la clase que cree instancias de objetivo (*Goal*).

  Además, tiene relación con la clase *Action*, puesto que ésta contiene una lista de objetivos (*Goal)* a satisfacer.

Nombre de clase: **Action**

- Situación: Clase contenida en el paquete *data*.
- Funcionalidad: Una acción está formada por una lista de objetivos que satisface y un conjunto de plantillas de servicios (*ServiceDescription*). Estas plantillas de

servicio, son las que mediante servicios pueden hacer alcanzar una situación contextual deseada.

- Relaciones: Una de las relaciones principales de la clase *Action*, es su relación con la clase *Agenda*, puesto que la *Agenda* es una lista ordenada de acciones posibles a realizar. Tal y como se ha comentado en la funcionalidad, la clase *Action* tiene relación con la clase *Goal* y con *ServiceDescription*.

Nombre de clase: **Search**

- Situación: Clase contenida en el paquete *data*

- Funcionalidad: Encargado de almacenar las búsquedas de descripciones de servicios, pudiendo ser especializada dependiendo de los distintos tipos de lenguajes de descripción de servicios utilizado. Esta clase se encarga de gestionar todos los datos relacionados con la búsqueda: tanto la consulta en sí, como sus resultados.

- Relaciones: El servicio de directorio (*ServiceDirectory*) es el encargado de gestionar las búsquedas realizadas en el directorio.

Nombre de la clase: **InstanceSearch**

- Situación: Clase contenida en el paquete *data*

- Funcionalidad: Clase encargada de almacenar toda la información relacionada con la búsqueda de instancias de servicios (descripción de servicio asociada a un proveedor concreto). La búsqueda almacena la descripción del servicio y como resultados, todos los proveedores que ofertan una instancia de dicha descripción de servicio.

- Relaciones: Las búsquedas de instancias de servicios son gestionadas por el directorio de servicios (*ServiceDirectory*).

Paquete Context

Nombre de paquete: **context**

- Situación: Al igual que *data*, es un paquete principal del diagrama.

- Funcionalidad: El paquete *context* se limita a un gestor de contexto (*ContextManager*) sin especificar cuál o cuáles son las fuentes de información y quién ha generado las mismas, aunque sí se debe especificar el tipo de interacción para obtener la información contextual.

- Relación: Este paquete tiene relación con los paquetes: *data*, puesto que se relaciona específicamente con la clase *Context*, y por otra parte, con el paquete *assistant*, para la generación de objetivos.

Clases

Nombre de clase: **ContextManager**

- Situación: La única clase perteneciente al paquete *context*.

- Funcionalidad: La clase *ContextManager* es un gestor de contexto y, por tanto, la clase encargada de manejar información de contexto (*Context*). Proporciona información contextual cuando le sea necesaria al generador de objetivos (*GoalGenerator*). Este gestor de contexto podrá ser modelado como un servicio.

- Relación: Las relaciones que tiene esta clase son con la clase *Context*, puesto que es la encargada de recuperarla del entorno, y por otra parte, con el *GoalGenerator* pasándole dicha información.

Paquete Directory

Nombre de paquete: **directory**

- Situación: Uno de los paquetes principales del diagrama de clases.

- Funcionalidad: El paquete *directory* engloba el directorio de información externa donde está almacenado tanto la lista de servicios, como los procedimientos para su ejecución.

- Relación: El paquete *directory* tiene relación tanto con el paquete de *data* (fundamentalmente con *ServiceInstance*), como con *assistant*.

Paquete Assistant

Nombre de paquete: **assistant**

- Situación: Es el paquete principal que contiene las clases con la funcionalidad básica para el asesoramiento.

- Funcionalidad: Es por tanto el encargado de realizar la coordinación de todos los paquetes que están contenidos en éste.

- Relación: Al ser el paquete principal del diagrama, tiene relación con todos los paquetes restantes.

Subpaquete goalGenerator

Nombre de paquete: **goalGenerator**

- Situación: Subpaquete contenido en el paquete *assistant*.
- Funcionalidad: Este paquete es el encargado de, en función de la información de contexto, generar y agrupar los objetivos que se cumplirán a través de las acciones.
- Relación: Por ello, tiene relación tanto con el paquete *context*, como con *data*.

Clases
Nombre de clase: **GoalGenerator**

- Situación: Clase contenida en el subpaquete *goalGenerator*.
- Funcionalidad: Este módulo es el encargado de la generación y agrupación de objetivos determinados a través de la información de contexto recuperada del entorno.
- Relación: El modulo generador de objetivos tiene por ello, relación con la clase *ContextManager* contenida en el paquete *context* y con la información de contexto (*Context*) propiamente dicha, contenida en el paquete *data*.

Subpaquete actionPlanning

Nombre de paquete: **actionPlanning**

- Situación: Subpaquete contenido en el paquete *assistant*.
- Funcionalidad: Este subpaquete es el encargado de la planificación de acciones de la *Agenda*, a partir de los objetivos generados anteriormente.
- Relación: Sólo tiene relación con el paquete de *data*.

Clases
Nombre de clase: **ActionPlanner**

- Situación: Clase contenida en el subpaquete *actionPlanning*.
- Funcionalidad: Es la clase encargada de la coordinación de las otras dos clases contenidas en ésta.
- Relación: Se relaciona con las clases del subpaquete: *Agenda* y *ServicePlanner*, y también con una clase externa, la clase *ServiceSearcher*.

Nombre de clase: **ServicePlanner**

- Situación: Clase contenida en el paquete *actionPlanning*.

- Funcionalidad: Esta clase es la encargada de, en función de los objetivos generados en la etapa anterior y las plantillas de servicio acordes a dichos objetivos, devolver una acción a añadir a la *Agenda*.

- Relación: Puesto que el paquete cuenta con la clase que se encarga de la coordinación, ésta se relaciona sólo con dicha clase.

Nombre de clase: **Agenda**

- Situación: Clase contenida en el paquete *actionPlanning*.

- Funcionalidad: la clase *Agenda* es aquella que está formada por una lista de acciones a realizar y lleva aparejado, un conjunto de objetivos a satisfacer (de forma implícita).

- Relación: Se relaciona con *Action,* puesto que está formada por una lista de acciones y con la clase coordinadora del paquete, *ActionPlanner*.

Subpaquete serviceExecution

Nombre de paquete: **serviceExecution**

- Situación: Subpaquete contenido en el paquete assistant.

- Funcionalidad: Este paquete será el encargado de ordenar la ejecución del servicio previamente seleccionado.

- Relación: Este paquete se relaciona con la clase que contiene la información del modelo de confianza (*Confidence*) y con el paquete *directory,* que se encargará de la ordenación de la ejecución del servicio.

Clases
Nombre de clase: ServiceExecution

- Situación: Clase contenida en el paquete *serviceExecution*.

- Funcionalidad: Esta clase será la encargada de ordenar la ejecución, una vez terminada la ejecución de un servicio deberá realizar la retroalimentación del modulo de confianza, es decir, tras recibir información del usuario sobre el servicio prestado, actualizará la información de confianza disponible.

- Relación: Principalmente se relaciona con el modulo *Confidence* para la retroalimentación comentada y con el *ServiceExecutor* contenido en el paquete *directory*.

Clases

Nombre de clase: **ServiceSearcher**

- Situación: Clase contenida en el paquete *assistant* y que no está contenida en ninguno de los subpaquetes descritos anteriormente.

- Funcionalidad: La función principal de esta clase *ServiceSearcher* es la búsqueda tanto de instancias como de plantillas de servicio en el directorio de servicios (*ServiceDirectory*).

- Relación: Puesto que su funcionalidad es la búsqueda, se relaciona con *ServiceDirectory*, donde realizará dicha búsqueda, y además se relaciona con el paquete *actionPlanning*, que es el encargado de realizar la llamada a *ServiceSearcher*.

Nombre de clase: **Confidence**

- Situación: Clase contenida en el paquete *assistant*, y que no está contenida en ninguno de los subpaquetes descritos anteriormente.

- Funcionalidad: El módulo de confianza realiza una doble funcionalidad: por una parte, presta información mediante un mecanismo de confianza que ayuda a la selección de un proveedor concreto para un servicio, por otra parte, se encarga de recopilar la información de satisfacción del usuario ante el servicio ejecutado y acumularla junto con su experiencia pasada.

- Relación: El módulo de confianza es utilizado por otras dos clases:

    - *ServiceLocator* para seleccionar el mejor proveedor de un servicio utilizando el mecanismo de confianza.

    - *ServiceExecution*, para recopilar las valoraciones subjetivas del usuario y acumularlas como experiencia local pasada.

3.4.1.2. Diagrama de secuencia de la arquitectura general

Los diagramas de secuencia UML son usados para modelar la interacción entre objetos en un sistema. Estos diagramas ponen énfasis en la distribución temporal de estas interacciones.

En este apartado se muestran los diagramas de secuencia para cada uno de los casos de uso explicados anteriormente.

Descubrir acciones

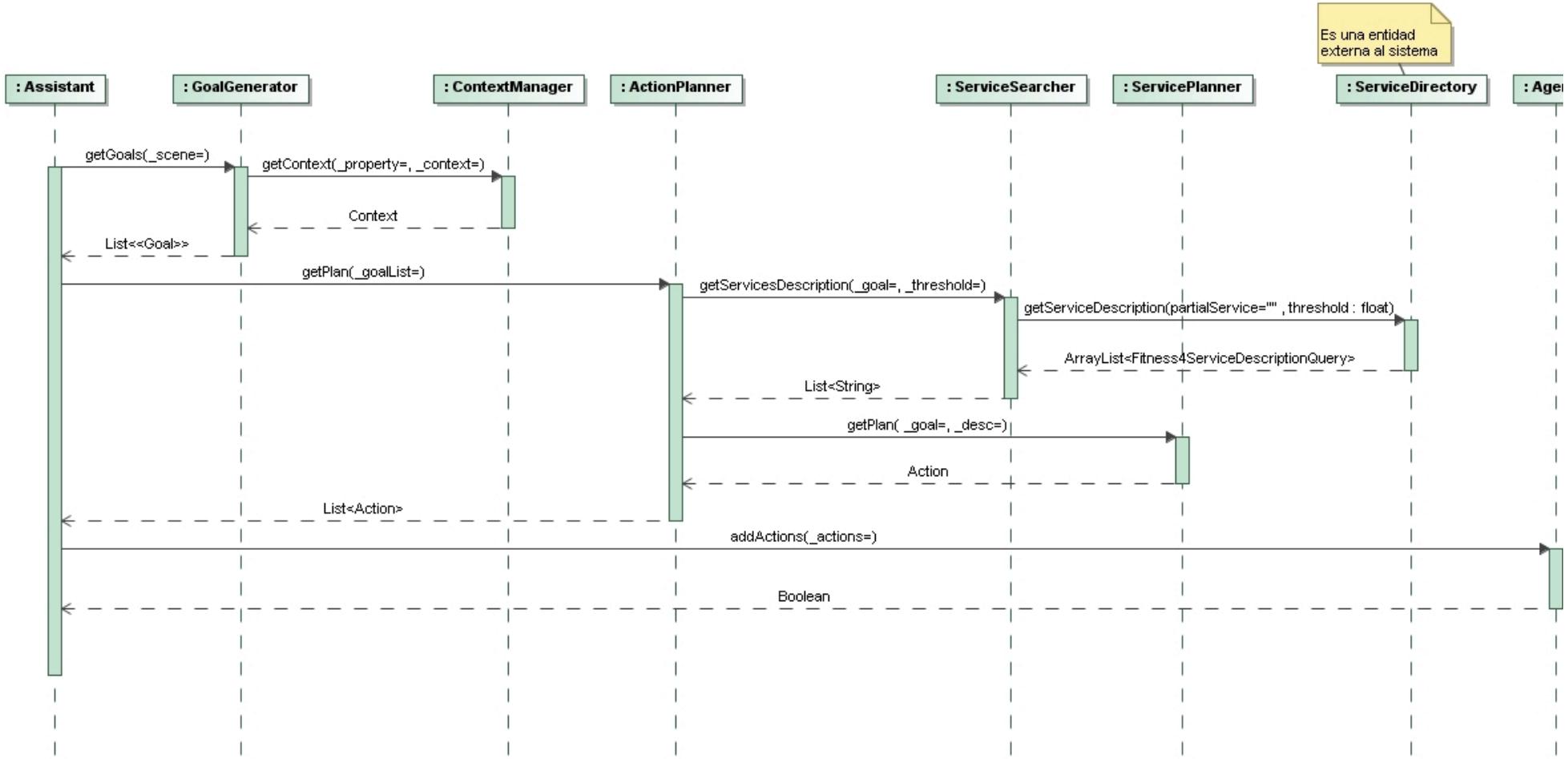

Figura 17 - Diagrama de secuencia de "Descubrir acciones"

En la figura Figura 17 se muestra el proceso necesario para añadir acciones a la Agenda, siendo una instrumentación a bajo nivel del caso de uso "*Descubrir acciones*".

Para ello, es necesario en primer lugar, generar los objetivos a través de la información de contexto recuperada del entorno; con dichos objetivos se generarán las acciones que satisfagan las plantillas de servicio encontradas que cumplen los objetivos.

A modo de simplificación, se considerará que el componente generador de objetivos *GoalGenerator* realizará una petición de información contextual al gestor de contextos (*ContextManager*) y a partir de dicha información se generará la lista de objetivos.

A través de la llamada a *getGoals*, se obtienen los objetivos generados a partir de la información de contexto.

La planificación de las acciones (un conjunto de objetivos y las descripciones de servicio que satisfacen estos objetivos) se lleva a cabo consultando al servicio de directorios por todas los servicios que podrían cumplir ciertos objetivos (o por aquellos servicios que son capaces de alcanzar un estado del mundo deseado y definido por los objetivos).

Por tanto, la idoneidad del plan estará en gran parte determinada por: la generación de los objetivos; la equiparación de dichos objetivos con el servicio de directorios. Una vez asociados los objetivos con los servicios obtenidos, se forma la acción, uniendo ambos componentes.

Cabe destacar que tanto el tipo de servicio, como los lenguajes utilizados para definir los mismos, son transparentes al proceso de asesoramiento: únicamente es necesario un componente que alinee dichas descripciones de servicios, con unas descripciones semánticas de los objetivos a conseguir (alineadas con las descripciones contextuales). La utilización de servicios descritos semánticamente ayuda en gran medida a esta labor, puesto que simplifican el proceso de equiparación.

A continuación se pasa a detallar la secuencia necesaria para la correcta generación de acciones.

Las acciones generadas, serán añadidas a la Agenda, con el propósito de que el usuario sea capaz de escoger en cada momento, cuál desea desempeñar.

Cabecera del método	**getGoals(): List<Goal>**
Invocado por	Assistant
Destino	GoalGenerator
Parámetros	Void
Retorno	Devuelve la lista de objetivos generada

Descripción	Gestiona la búsqueda de objetivos haciendo uso de la información contextual. En el caso concreto, se presenta como un posible parámetro un escenario concreto para comprobar su funcionamiento con escenarios predefinidos.

Cabecera del método	**getContext(): Context**
Invocado por	GoalGenerator
Destino	ContextManager
Parámetros	void
Retorno	Devuelve la información de contexto solicitada
Descripción	Este método refleja el medio en que la generación de objetivos consume información contextual, de forma genérica.

Una vez generados los objetivos a cumplir, se generan las acciones que serán posteriormente añadidas a la Agenda.

Cabecera del método	**getPlan(g: List<Goal>): List<Action>**
Invocado por	Assistant
Destino	ActionPlanner
Parámetros	g: Lista de objetivos
Retorno	Las acciones que satisfacen los objetivos recibidos como parámetro.
Descripción	Este método es el encargado de transformar los objetivos de los agentes en servicios susceptibles de ser ejecutados (transformación de objetivos en acciones factibles).

Para cada uno de los objetivos que se pretenden satisfacer, se busca en el directorio de servicio, una lista de plantillas de servicio acordes a éste.

Cabecera del método	**getServiceDescription(g: Goal, threshold: float): List<String>**
Invocado por	ActionPlanner
Destino	ServiceSearcher
Parámetros	g: objetivo para el que buscar descripciones de servicio que puedan satisfacerlo. threshold: umbral que indica le grado de similitud entre un servicio y un objetivo, a partir del cual se considera que dicho servicio es capaz de cumplir un objetivo.
Retorno	La lista de plantillas de servicio que satisfagan ese objetivo en particular. Dichas plantillas se expresan utilizando un lenguaje de descripción de servicios.
Descripción	Este método se invocará tantas veces como objetivos haya. En cada una de estas llamadas, se devolverá una lista de plantillas de servicio (en caso de que existan).

A través de la llamada a este método, se busca en el directorio de servicio las plantillas de servicio acordes a la información de contexto.

Como ya se ha apuntado anteriormente, la infraestructura de servicio (directorio de servicios) es un componente externo al agente recomendador, que hace que el sistema sea distribuido. Todas las llamadas que se realizan al directorio no son más que invocaciones a servicios web remotos.

En este caso la llamada distribuida se realiza para la búsqueda de plantillas de servicio que satisfagan los objetivos generados para el usuario recomendado.

Cabecera del método	**getServiceDescription(partialService: String, threshold: float): List<Fitness4ServiceDescriptionQuery>)**
Invocado por	ServiceSearcher
Destino	ServiceDirectory
Parámetros	partialService: un String que representa una descripción de una consulta de servicio. threshold: el grado de encaje mínimo por el que se considera que un servicio cumple con la consulta dada.
Retorno	La lista de plantillas de servicio que satisfagan ese objetivo particular, ponderada con su grado de equiparación.
Descripción	Este método devolverá la plantilla/s de servicio que satisfagan el objetivo particular.

Una vez obtenida tanto la lista de plantillas de servicio, como los objetivos que se pretenden satisfacer a través de éstas, se decide cuál es la acción a añadir a la *Agenda* que cumpla con dichas restricciones.

Cabecera del método	**getPlan (f: List<Fitness4ServiceDescriptionQuery, g:List<Goal>): List<Action>**
Invocado por	ActionPlanner
Destino	ServicePlanner
Parámetros	Una lista de envoltorios de plantillas de servicio y un objetivo.
Retorno	Una lista de acciones posibles para satisfacer dichos objetivos.
Descripción	Planificación de acciones a realizar.

Una vez se obtienen las acciones, se deben añadir a la lista de acciones existentes en la Agenda.

Cabecera del método	**addAction(a: Action):List<Action>**
Invocado por	Assistant
Destino	Agenda
Parámetros	a: acción que hay que añadir al listado de la agenda
Retorno	La lista de acciones que componen la Agenda, una vez añadida la nueva acción.
Descripción	Añadirá en la agenda de manera ordenada la acción.

Seleccionar una acción y ejecutar

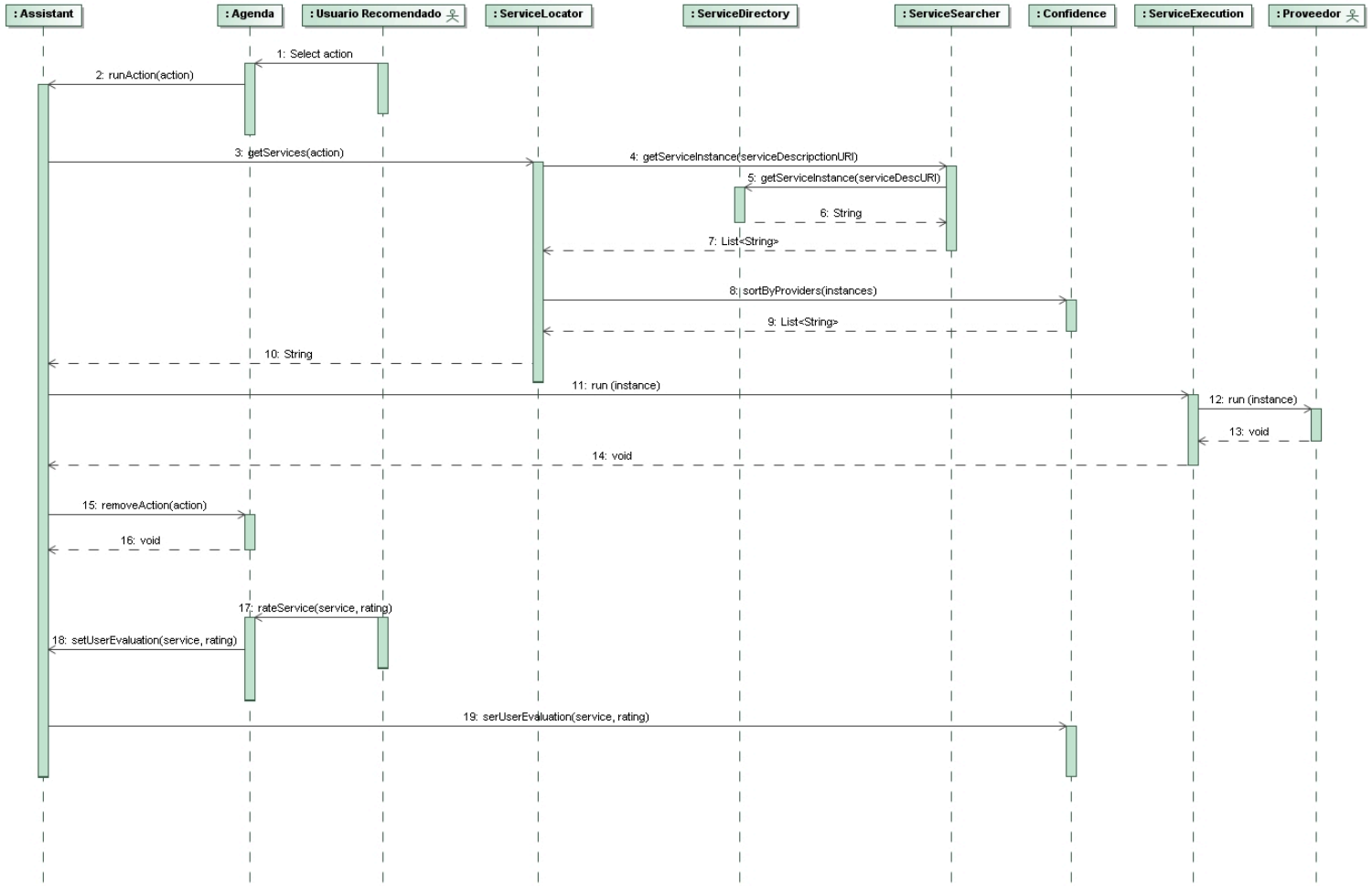

Figura 18 - Diagrama de secuencia de "Seleccionar acción y ejecutar"

Este diagrama de secuencia (Figura 18) se centra en la selección, por parte del usuario, de la acción a ejecutar, así como su ejecución y posterior valoración.

Para ello, en primer lugar es necesaria la interacción con el usuario, que determinará la acción a ejecutar. Una vez se haya seleccionado, la *Agenda* es la encargada de avisar al asistente, de qué acción ha sido seleccionada para su ejecución.

Cabecera del método	**runAction(a: Action):void**
Invocado por	Agenda
Destino	Assistant
Parámetros	a: acción seleccionada para su ejecución.
Retorno	Void
Descripción	El asistente será el encargado de la ejecución de la acción solicitada.

El asistente una vez que conoce la acción, es el encargado de llevar a cabo la coordinación necesaria para la búsqueda del proveedor de servicio adecuado (asociado a la plantilla de servicio que incluye la acción de forma implícita).

Cabecera del método	**getServices(a: Action):ServiceInstance**
Invocado por	Assistant
Destino	SeviceLocator
Parámetros	Este método recibe una acción (objetivo + plantilla de servicio) para poder ordenar la búsqueda de instancias a partir de la plantilla de servicio en el directorio.
Retorno	Una referencia a la instancia del servicio que será ejecutada.
Descripción	Ordena la realización de las búsquedas de servicios en el directorio.

El módulo *ServiceLocator* realiza la llamada para la búsqueda de la plantilla de servicio.

Cabecera del método	**getServiceInstance(d: ServiceDescription):List<ServiceInstance>**
Invocado por	ServiceLocator
Destino	ServiceSearcher
Parámetros	d: recibe una plantilla de servicio que buscará en el directorio.
Retorno	Una lista de instancias devueltas por el directorio.
Descripción	Ordena la búsqueda de plantillas de servicio en el directorio, devolviendo una lista de todas las instancias de servicio acordes a dicha plantilla.

A su vez, el módulo *ServiceSearcher* será el encargado de indicar al directorio de servicios que se desea realizar una búsqueda de instancias de servicio, que sean acordes a la plantilla de servicio recibida.

Al igual que en el caso de uso "Descubrir acciones", la llamada al servicio de directorio se realiza de forma remota, es decir, se trata de un elemento externo al agente recomendador.

En este caso se realiza una llamada a un servicio web provisto por el directorio de servicios, que permite la búsqueda de proveedores de servicio que satisfacen la plantilla de servicio pasada como parámetro.

Cabecera del método	**getServiceInstance(d:ServiceDescription):List<ServiceInstance>**
Invocado por	ServiceSearcher
Destino	ServiceDirectory
Parámetros	Este método recibe una plantilla de servicio que buscará en el directorio.
Retorno	Una lista de las instancias de la plantilla de servicio.
Descripción	Busca en el directorio la plantilla de servicio, devolviendo las instancias de servicio correspondientes.

Una vez se han obtenido todas las instancias de servicio posibles para proveer el servicio, debemos acudir al módulo de confianza, para saber qué proveedor escoger, basándonos en la información pasada acerca de dichos proveedores.

Cabecera del método	**sortByProviders(i:List<ServiceInstances>):List<ServiceInstances>**
Invocado por	ServiceLocator
Destino	Confidence
Parámetros	Este método recibe una lista de instancias de servicio que deberá ordenar en función de la información de confianza disponible.
Retorno	Una lista de instancias de servicio ordenadas.
Descripción	Ordena las instancias de servicio acorde a un modelo de confianza.

La instancia de servicio que haya resultado escogida, debe ser ejecutada, por lo que el asistente (encargado de la coordinación) ordenará la ejecución de dicho servicio.

Cabecera del método	**run(i: ServiceInstance, a: Action):void**
Invocado por	Assitant
Destino	ServiceExecution
Parámetros	Este método además de recibir la instancia de servicio a ejecutar, debe recibir la acción para poder eliminarla de la lista de acciones de la agenda (una vez haya sido ejecutada).
Retorno	Void
Descripción	Realiza la llamada para la ejecución del servicio en la plataforma de servicios y elimina la acción correspondiente de la agenda.

El módulo encargado de la ejecución del servicio propiamente dicho, es el *ServiceExecutor* (dentro de la infraestructura de servicios), un agente externo al sistema, con el que se interactúa.

Cabecera del método	**run(i: ServiceInstance):void**
Invocado por	ServiceExecution
Destino	Provider
Parámetros	Recibe la instancia de servicio a ejecutar
Retorno	Void
Descripción	El módulo ejecutor es en encargado de ejecutar el servicio ofrecido por el proveedor.

Una vez haya sido ejecutado el servicio, se debe eliminar de la *Agenda* la acción asociada a dicha instancia de servicio.

Cabecera del método	**removeAction(a: Action):void**
Invocado por	ServiceExecution
Destino	Agenda
Parámetros	Recibe la acción a borrar de la lista de la agenda
Retorno	Void
Descripción	Actualiza las acciones disponibles en la agenda, una vez que se hayan ejecutado, borrando la acción anteriormente ejecutada.

Con el objetivo de poder escoger el mejor proveedor en el futuro, el usuario debe valorar el servicio recibido. Es por ello que la instancia de servicio debe ser evaluada tras su ejecución, para poder retroalimentar el módulo de confianza para próximas ocasiones.

Cabecera del método	**rateService(i:ServiceInstance, float value):void**
Invocado por	Usuario recomendado
Destino	Agenda
Parámetros	La instancia de servicio recién ejecutada y la valoración del servicio
Retorno	Void
Descripción	El usuario recomendado a través de la interfaz de la Agenda, valora al proveedor del servicio.

Cabecera del método	**setUserEvaluation(i:ServiceInstance, float value):void**
Invocado por	Agenda
Destino	Assistant
Parámetros	La instancia de servicio recién ejecutada y la valoración del servicio
Retorno	Void
Descripción	La Agenda realiza una llamada al asistente para actualizar el valor del proveedor de servicio en el módulo de confianza.

Cabecera del método	**setUserEvaluation(i:ServiceInstance, float value):void**
Invocado por	Assistant
Destino	Confidence
Parámetros	La instancia de servicio recién ejecutada y la valoración del servicio

Retorno	Void
Descripción	El asistente realiza una llamada al módulo de confianza para poder actualizar la información del proveedor de servicio.

Refinar propuesta

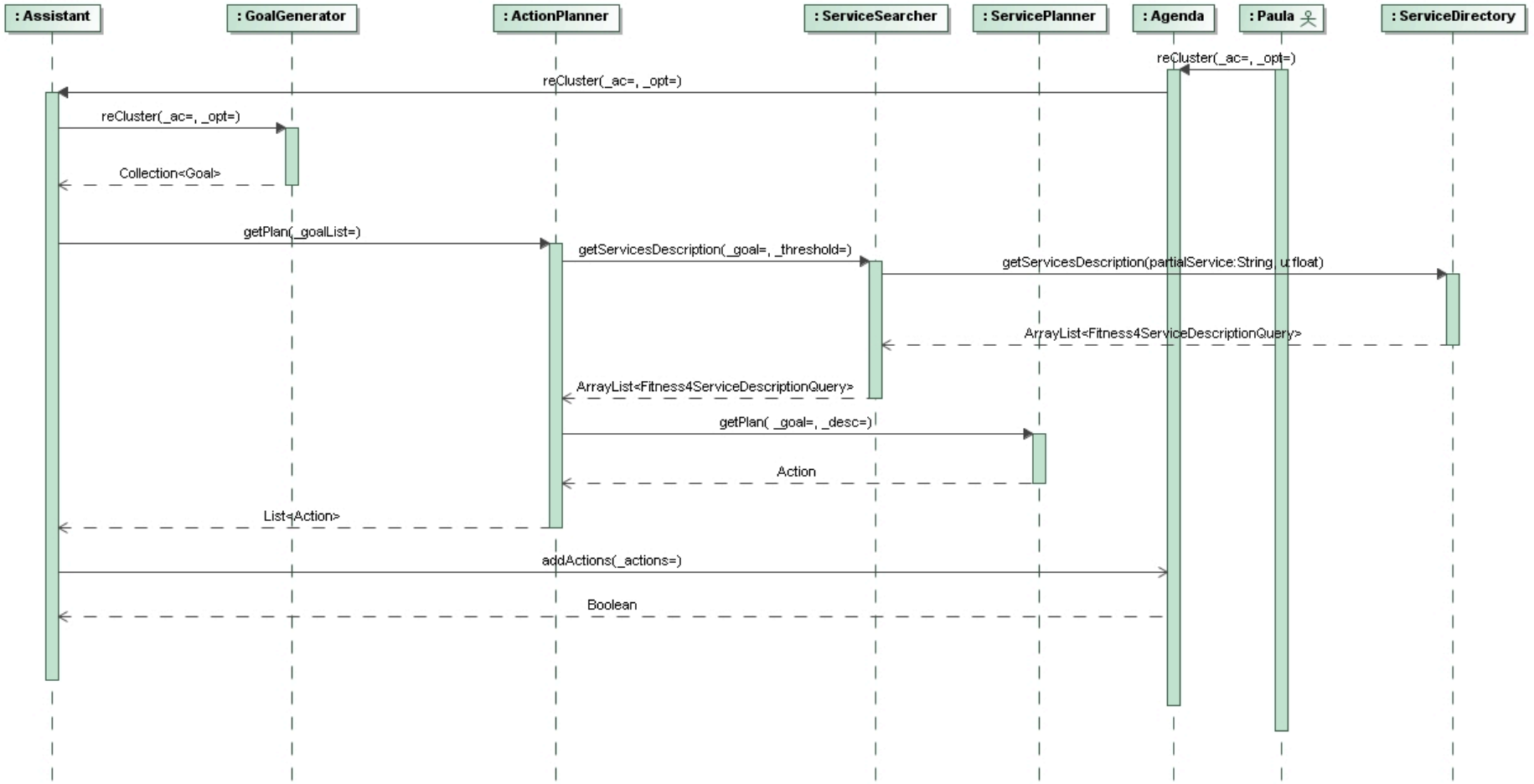

Figura 19 - Diagrama de secuencia de "Generalizar acción"

El caso de uso "*refinar propuesta*" proporciona la posibilidad de generalizar o particularizar cada una de las acciones ofrecidas por el asistente. De forma que, si así se desea, se pueden ejecutar acciones que sin ser las ofrecidas a priori por el asistente, están estrechamente relacionadas.

Este caso de uso siempre es iniciado por parte del usuario al interactuar con la agenda de acciones propuesta por el agente.

Supongamos el caso de la particularización de una acción, lo que supondría realizar un *zoom* de la acción seleccionada.

Lo que en realidad ocurre cuando se desea realizar el *zoom*, es que la acción que ha sido seleccionada por el usuario, vuelve al módulo de generación de objetivos. Más concretamente, con el método *reCluster* se consigue una nueva agrupación de los objetivos.

Cabecera del método	**reCluster(a: Action, type:int):List<Action>**
Invocado por	Agenda
Destino	Assistant
Parámetros	a: La acción que se desea particularizar o generalizar type: El entero indica cuál de las dos acciones se desea realizar (particularizar o generalizar).
Retorno	Una nueva lista de acciones agrupadas
Descripción	Con este método se realiza una nueva agrupación de las acciones que se añadirán a la Agenda

Cabecera del método	**reCluster(a: Action, type: int):List<Goal>**
Invocado por	Assistant
Destino	GoalGenerator

Parámetros	a: La acción que se desea particularizar o generalizar. type: El entero indica cuál de las dos acciones se desea realizar.
Retorno	Una nueva lista de objetivos a satisfacer
Descripción	Con este método se realiza una nueva agrupación de los objetivos, en función del parámetro type.

A partir de este punto, el flujo es el mismo que en el caso de *añadir una acción a la Agenda.*

Puesto que el refinamiento de acciones (particularización o generalización) supone la generación de nuevas acciones en función de las necesidades del usuario, al igual que en casos anteriores, para poder obtener las plantillas de servicio que satisfacen los objetivos del usuario, se debe realizar una llamada remota al servicio de directorio. A través de esta llamada distribuida, se obtendrán las plantillas de servicio acordes a la nueva agrupación de objetivos.

3.4.2. Interoperabilidad del agente con componentes externos

El agente asistente inteligente mIO!, debe interaccionar con componentes externos, que le ofrecen la información necesaria, para poder llevar a cabo la recomendación de servicios adaptados al contexto del usuario.

En este apartado, se explicarán las interacciones que dicho agente asistente presenta con el resto de componentes, con el objetivo de satisfacer las necesidades del usuario.

En el ámbito de este proyecto se ha desarrollado el agente asistente inteligente que interacciona con el resto de componentes externos que han sido desarrollados en otro Proyecto Fin de Carrera (Martínez, 2011).

3.4.2.1. Interoperabilidad con el gestor de contexto

Para que el asistente inteligente pueda percibir el contexto del usuario recomendado, es precisa la interacción con el módulo externo gestor de contexto.

Dicho gestor de contexto tiene las siguientes funciones (iSoco, Telefonica I+D, 2009):

- Publicación de información de contexto, su única responsabilidad será componer el objeto que contendrá la información de contexto a transmitir, incorporándole determinados metadatos (fuentes de contexto, datos temporales, etc.) que permitan obtener información sobre dicho contexto provisto.

- Consulta de información de contexto, el gestor de contexto actuará de enlace entre el consumidor de contexto y el proveedor de contexto, tratando de minimizar el hecho de que el consumidor deba conocer el lenguaje de consulta de contexto.

- Suscripción a información de contexto, se dará de forma asíncrona permitiendo que sea el consumidor el que establezca de manera más o menos precisa, un elemento de información sobre cuyos cambios quiere ser notificado.

El siguiente diagrama de clases (Figura 20) muestra la interacción del asistente con el gestor de contexto, que le permite acceder de esta forma a la información del entorno del usuario recomendado.

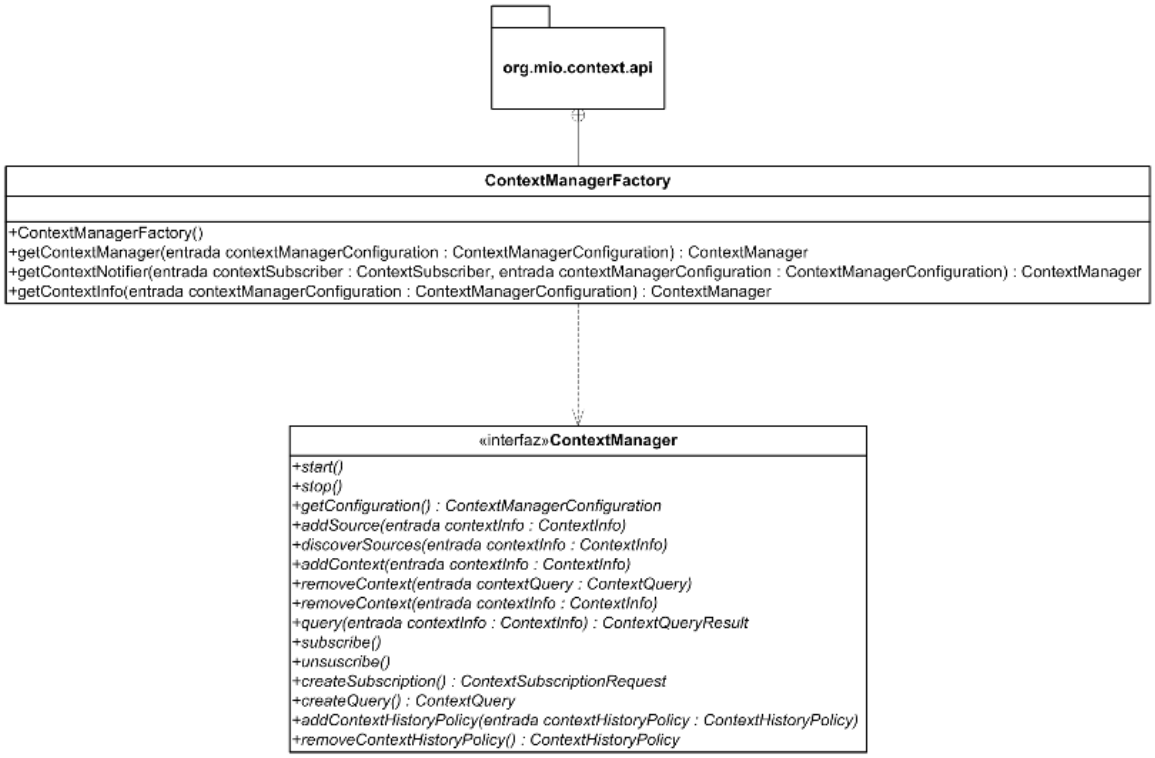

Figura 20 - API del gestor de contexto

Como se puede observar tiene los métodos necesarios para llevar a cabo la interacción con el contexto. En secciones anteriores se indicó que al igual que el directorio de servicios, el gestor de contexto es un elemento externo al agente asistente y que además presentaba un interfaz REST para su acceso remoto.

A continuación, se muestra una tabla que resume las URIs correspondientes a cada uno de los recursos a partir de los cuales se consume la información contextual. A efectos del presente proyecto Fin de Carrera tan sólo se ha tenido en cuenta el consumo de información de contexto.

URI	Operaciones	Descripción
/CMConsumer/query	Post	Recurso para la realización de consultas sobre información de contexto (dicha consulta será realizada en formato *SPARQL*).
/CMConsumer/subscriber	Post	Recurso para el establecimiento de suscripciones a eventos de contexto. Requiere recibir como parámetro un servicio *REST* al que enviar los eventos mediante método *POST*.

3.4.2.2. Interoperabilidad con el directorio de servicios

En apartados anteriores ya se expuso la necesidad de un directorio de servicios en el que se encontrasen alojadas tanto las plantillas como las instancias de servicio.

Dicho directorio de servicios se encuentra distribuido, por lo que para realizar la interacción entre el agente asistente y el directorio es preciso utilizar una forma de comunicación punto a punto, en este caso se ha utilizado la arquitectura REST (ver sección 1.2) que se basa en el uso del protocolo *HTTP*.

Puesto que el asistente interacciona con el directorio para dos propósitos diferentes, existen dos formas de interacción: una para la búsqueda de plantillas y otra para la búsqueda de instancias de servicio.

En ambos casos, la comunicación se ha realizado utilizando la arquitectura REST. Debido al carácter móvil del prototipo del proyecto, resulta ideal la utilización de REST al basarse en la utilización del protocolo *HTTP* para la realización de las comunicaciones. Además, no es necesario el uso de ninguna librería externa, puesto que bastaría con establecer una conexión *HTTP* a través de las librerías nativas de Java.

A pesar de ello, para el desarrollo se ha hecho uso de la librería *Restlet* (Restlet, 2006) para *Android*, que facilita el uso de dicho protocolo.

Búsqueda de plantillas de servicio

Una vez determinados los objetivos que se pretenden alcanzar para el usuario, se buscan las plantillas de servicio que cumplen dichos objetivos.

En la Figura 21 se puede observar la comunicación entre el asistente y el directorio de servicios.

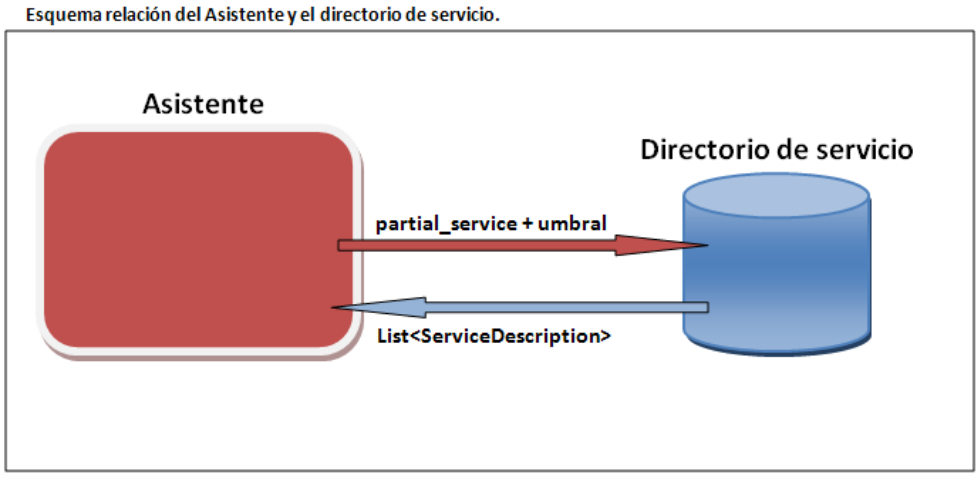

Figura 21 - Detalle de búsqueda de plantillas en el directorio de servicios

El asistente realiza una petición al directorio de servicios, con un servicio parcialmente lleno y un umbral mínimo de encaje y el directorio de servicios, acorde a dichos parámetros, obtiene una lista de plantillas de servicio.

A continuación, se explica más detalladamente cómo se lleva a cabo la comunicación entre el asistente y el directorio usando REST, haciendo hincapié en los tipos de peticiones *HTTP* que se realizan, así como el contenido de éstas.

En la Figura 22 se puede observar cómo son los flujos de información existentes entre el asistente y el directorio de servicios, así como los métodos *HTTP* utilizados en éste.

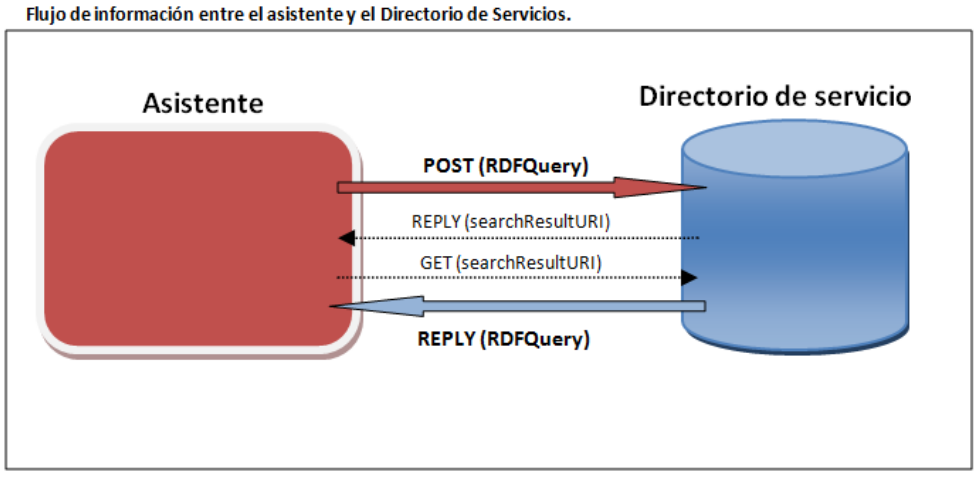

Figura 22 - Flujo de información entre el asistente y el Directorio de Servicios

En primer lugar, el asistente realizará una petición *POST* contra el directorio de servicios, más concretamente contra el recurso *RDFQuery* que hace referencia a la tecnología semántica *RDF* usada para la descripción de servicios. Esta petición *POST* llevará en el cuerpo de la petición: la consulta del servicio, que contendrá los objetivos que se pretenden satisfacer, junto con un umbral mínimo de encaje, para establecer en qué grado mínimo los servicios encontrados (en este caso plantillas) deben encajar con los objetivos

iniciales. El cuerpo del mensaje *POST* incluye un objeto *XML* que ha sido serializado utilizando la librería *XStream* (xStream, 2007) para convertir objetos Java a *XML*.

A continuación, el directorio de servicios tras recibir la petición de búsqueda de plantillas de servicio, pondrá en marcha el mecanismo de búsqueda y alineamiento de objetivos con la descripción de servicio. Una vez determinadas cuáles son las plantillas que encajan, devolverá en la cabecera *location* la URI del nuevo recurso *query* creado.

El asistente conoce la URI donde están alojados los recursos que hacen referencia a las plantillas de servicio que satisfacen los objetivos, por lo que tendrá que realizar una petición *GET* para obtener la representación de dichos recursos plantilla de servicio. El directorio de servicio al recibir la petición *GET*, devolverá dicha representación que contendrá las URIs que identifican la localización de las plantillas.

Por último, se especifica el árbol de URIs que contiene el directorio de servicios, así como que funcionalidad se espera de éste al realizar las distintas peticiones sobre sus recursos REST.

URI	**Operaciones**	**Descripción**
/template/RDF	Get	Devuelve la lista completa de servicios disponibles
/template/RDF/Search	Get	La búsqueda situada en la URI de la petición
	Post	Genera una búsqueda nueva
/template/RDF/Search/{numBusqueda}	-	
/template/RDF/Search/{numBusqueda}/Query	Get	Devuelve la consulta original de la búsqueda apuntada mediante la URI
/template/RDF/Search/{numBusqueda}/Result	Get	El resultado de la búsqueda
/template/RDF/Search/{numBusqueda}/Threshold	Get	El umbral para dicha búsqueda

En caso de usar otra tecnología semántica para la descripción de servicios, como es el caso de *OWL* en lugar de *RDF*, entonces existen otras URI diferentes para acceder a dichos servicios:

URI	Operaciones	Descripción
/template/OWL	Get	Devuelve la lista completa de servicios disponibles
/template/ OWL /Search	Get	La búsqueda situada en la URI de la petición
	Post	Genera una búsqueda nueva
/template/ OWL /Search/{numBusqueda}	-	
/template/ OWL /Search/{numBusqueda}/Query	Get	Devuelve la consulta original de la búsqueda apuntada mediante la URI
/template/ OWL /Search/{numBusqueda}/Result	Get	El resultado de la búsqueda
/template/ OWL /Search/{numBusqueda}/Threshold	Get	El umbral para dicha búsqueda

Búsqueda de instancias de servicio

Para el caso de la búsqueda de instancias de servicio en el directorio, el flujo comienza con la petición por parte del asistente, el cual pasa la URI que hace referencia al identificador de la plantilla de servicio y en función de éste, se realiza una búsqueda en el directorio de servicio que da como resultado una lista de instancias de servicio.

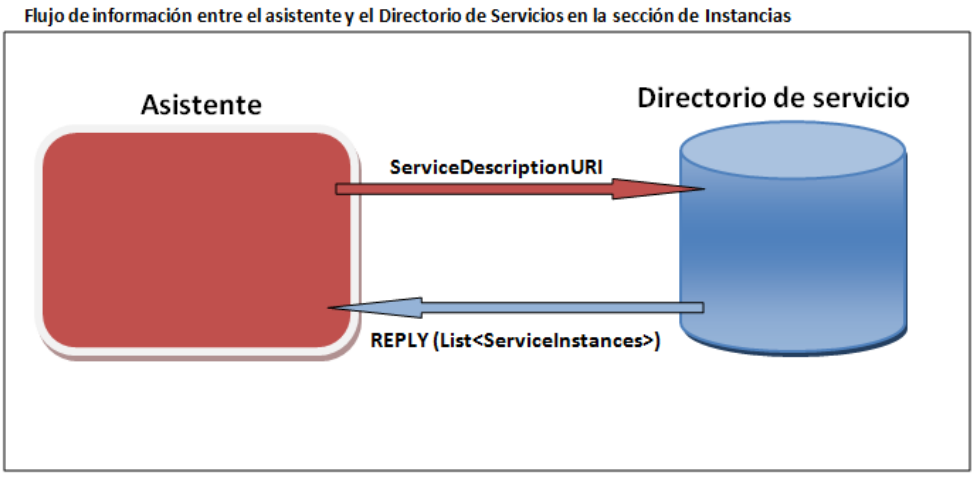

Figura 23 - Flujo de información entre el asistente y el Directorio de Servicios de instancias

La Figura 23 muestra a grandes rasgos la comunicación existente entre el servicio de directorio y el asistente inteligente, para la búsqueda de instancias de servicio.

En la siguiente figura, se puede ver dicha comunicación a más bajo nivel, así como qué información se intercambian para la búsqueda de instancias, y las peticiones REST necesarias.

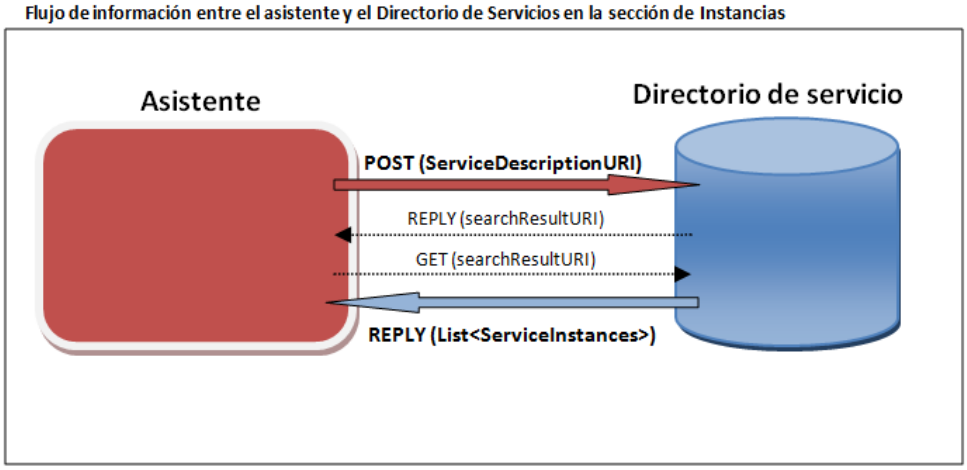

Figura 24 - Flujo de peticiones REST entre el asistente y el Directorio de Servicios de instancias

Para realizar la búsqueda de instancias de servicio, el asistente realiza en primer lugar una petición *POST* a la parte del directorio encargado de la búsqueda de instancias. Dicha petición se realizará contra el recurso encargado de la búsqueda de instancias acordes a la plantilla especificada como a través de la URI.

Tras realizar el *POST*, el directorio de servicios, buscará todas aquellas instancias de servicio que implementen la plantilla de servicio que se ha pasado en la petición. Como respuesta a la petición *POST*, se devolverá una URI que identifica el recurso que aglutina todas las instancias de servicio que implementan dicha plantilla.

Al igual que en el caso de las plantillas de servicio, gracias a la URI recibida, el asistente inteligente conoce donde se encuentra el recurso y podrá por tanto realizar una petición *GET* para obtener la representación de dicho recurso.

Al igual que en el caso de las plantillas, se muestra a continuación una tabla que contiene la información referente a todos los recursos y los métodos *HTTP* que son necesarios para poder acceder, así como una explicación de la funcionalidad de cada uno de éstos.

URI	Operación	Descripción
/instance	Get	Devuelve una lista completa de pares <proveedor, servicio>
/instance/Search	Post	Genera una nueva búsqueda

/instance/Search/{numBusqueda}	Get	Devuelve una búsqueda situada en la URI de la petición
/instance/Search/{numBusqueda}/Query	Get	Devuelve la consulta original de la búsqueda apuntada mediante la URI
/instance/Search/{numBusqueda}/Result	Get	El resultado de la búsqueda

3.4.2.3. Interoperabilidad con el Modelo de Confianza

Cuando se realiza una búsqueda en el directorio de servicios para la obtención de proveedores de servicio a través de una plantilla dada, pueden ser multitud de instancias de servicio las que encajen con ésta, y por tanto, existen muchos proveedores capaces de ofrecer el servicio buscado. Ante esta situación, se debe escoger de entre todos los proveedores de servicio, aquel que sea más idóneo.

Dicha idoneidad se debe buscar desde el punto de vista de la satisfacción de los objetivos del usuario.

El esquema general de interacción entre el agente y los módulos externos, se puede observar en la siguiente figura:

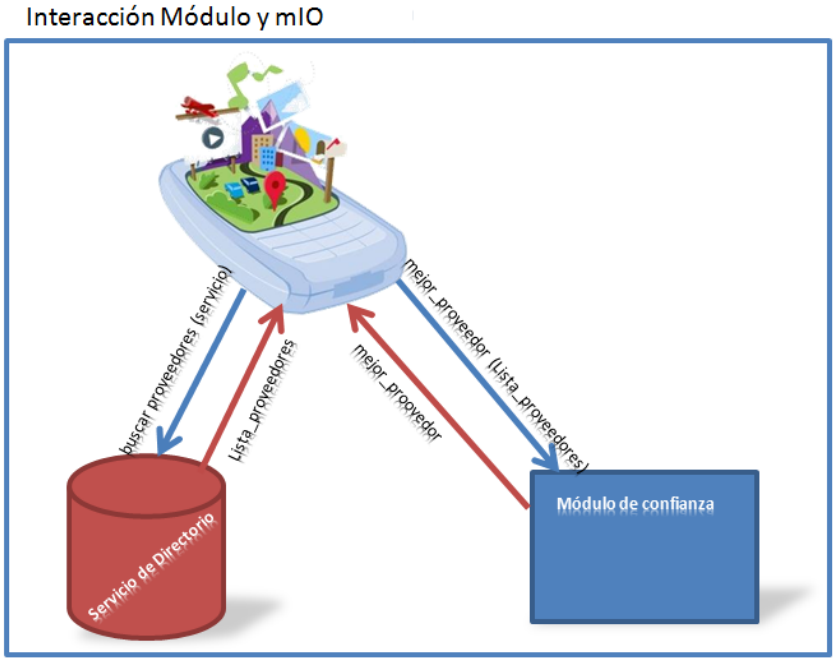

Figura 25 - Detalle de interacción agente con módulos externos

El flujo de interacción comienza cuando el asistente encarga al servicio de directorio la realización de la búsqueda de proveedores a partir de una plantilla dada.

Cuando el asistente obtiene la lista de proveedores potenciales, encarga al módulo de confianza la ordenación de éstos en términos de confianza, devolviendo así una lista ordenada de proveedores, de mayor a menor grado de confianza. El modelo de confianza que en este apartado se presenta, corresponde con el propuesto en (Billhardt, Hermoso, & Centeno, 2007) cuya implementación está especificada en el Proyecto Fin de Carrera "Directorios de servicio y modelos de confianza para la recomendación de servicios" (Martínez, 2011).

Dicho módulo de confianza, mantiene una tabla LIT (*Local Interaction Table*) que contiene una entrada por cada proveedor ofreciendo un determinado servicio (llamado dupla *<S, P>*, en referencia a *Service* y *Provider*).

Por cada una de estas entradas en la tabla, mantiene información relativa a (Billhardt, Hermoso, & Centeno, 2007):

- la dupla *<S, P>* que identifica unívocamente cada proveedor ofreciendo un servicio concreto.
- El valor de confianza ($c_{s,p}$, *confidence*) de la dupla *<S, P>* acumulado hasta el momento.
- La fiabilidad ($r_{s,p}$, *reliability*) con respecto al valor de confianza mostrado en la tabla.
- El número total de interacciones que ha tenido el usuario con el par *<S, P>*.

A partir de la información almacenada en dicha tabla, realiza la ordenación de los proveedores de servicio.

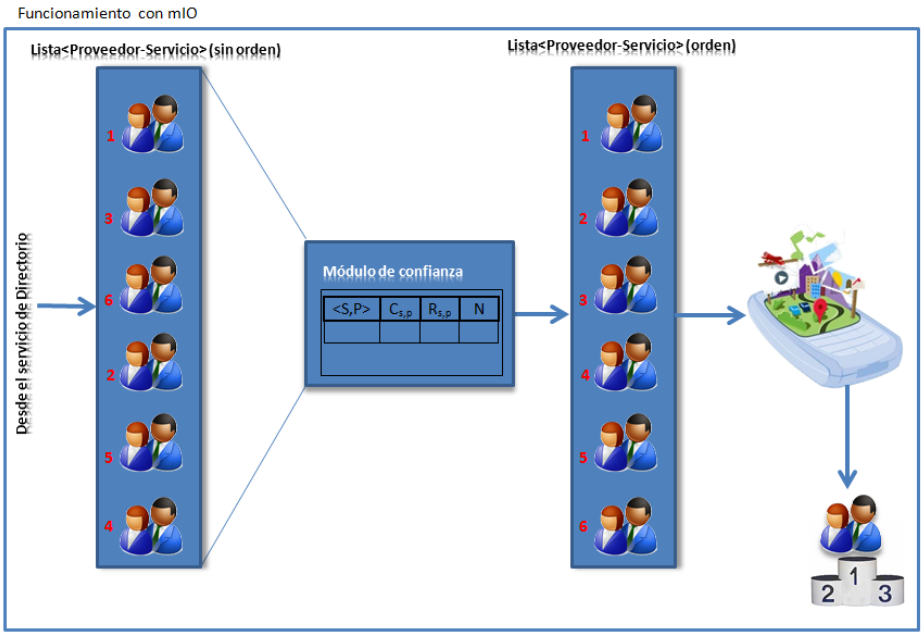

Figura 26 - Detalle de funcionamiento del módulo de confianza

Esta información almacenada en la *LIT* del módulo de confianza, deberá ser actualizada conforme el usuario recomendado tiene nuevas interacciones con un par *<S, P>.* Dicha actualización se realizará gracias a la valoración dada por el usuario tras la ejecución de un servicio. De forma que a través de esta retroalimentación y con un mayor volumen de información, el módulo de confianza sea capaz de ordenar en futuras ocasiones, de una manera más idónea los proveedores en función de las preferencias del usuario.

Además, esta tabla de interacciones *LIT*, debe ser local puesto que almacena información relativa al propio agente. Por otro lado, debe disponer de persistencia de datos para poder reutilizar la información disponible entre distintas interacciones.

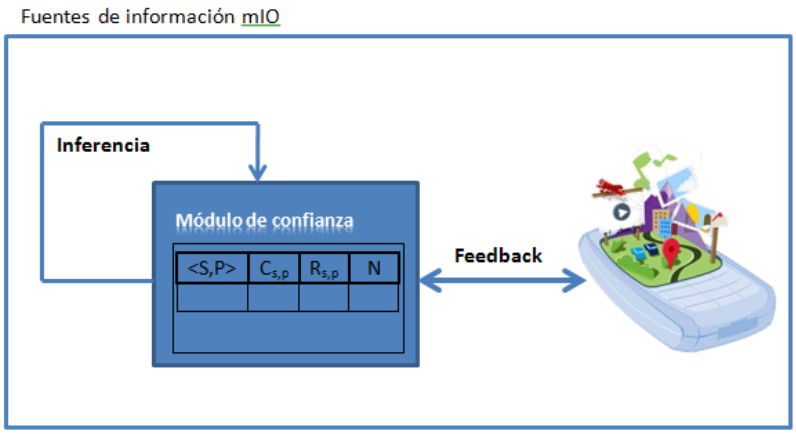

Figura 27 - Detalle de fuentes de información para el módulo de confianza

3.4.3. Agenda de acciones

El agente asistente inteligente mIO! está alojado en un dispositivo móvil, por lo que precisa de una interfaz de interacción con el usuario.

En este apartado, se explicará cómo es posible la interacción del usuario con el sistema a través de la Agenda, componente que hace de interfaz visible para éste.

3.4.3.1. Diagrama de clases de agenda de acciones

En la Figura 28 se muestra el diagrama de clases correspondiente a la agenda de acciones. Se trata de un diagrama sencillo que pone de manifiesto la estructura de la agenda, así como las relaciones que esta mantiene con el resto de componentes.

La agenda esta compuesta por una lista de acciones que satisfacen los objetivos (*Goals*) que ha generado el agente recomendador a través de la información contextual del usuario (*Context*).

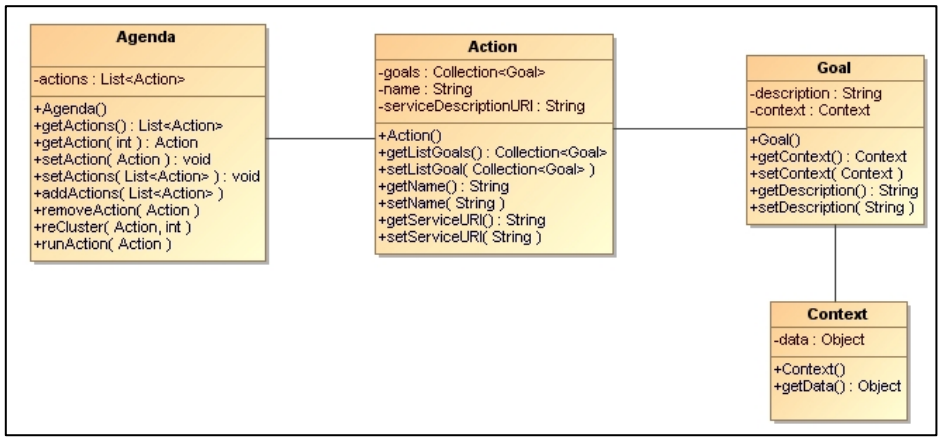

Figura 28 - Diagrama de clases de la Agenda

Clases

Nombre de clase**: Agenda**

- Funcionalidad: Es la clase que representa la Agenda del usuario, que contiene una lista de acciones que pretenden satisfacer los objetivos del usuario recomendado. Hace de conexión con el usuario (a través de la interfaz).

  Además, contiene todos aquellos métodos que permiten llevar a cabo la funcionalidad que ha sido descrita en los casos de uso de la agenda, tales como la ejecución de una acción, eliminación y refinamiento de las acciones entre otros.

- Relaciones: Se relaciona con la clase *Action*, puesto que como se ha mencionado, contiene una lista de éstas.

Nombre de clase**: Action**

- Funcionalidad: Instancias de la clase acción (*Action)* son ofrecidas al usuario para satisfacer sus objetivos.

- Relaciones: Se relaciona con la clase objetivo (*Goal*), puesto que las acciones satisfacen los objetivos detectados para el usuario.

Nombre de clase**: Goal**

- Funcionalidad: La clase *Goal* representa cada uno de los objetivos que se han generado a partir de la información contextual.

- Relaciones: Se relaciona con la clase *Context*, puesto que los objetivos son generados a partir de la información percibida del contexto del usuario recomendado.

Nombre de clase**: Context**

- Funcionalidad: La clase *Context* representa el estado relevante del mundo para el usuario recomendado en un momento dado.

- Relaciones: Se relaciona con la clase *Goal*, ya que esta información contextual es aglutinada en forma de objetivos para satisfacer las necesidades del usuario.

3.4.3.2. Diagrama de secuencia de la agenda de acciones

Este apartado muestra qué ocurre en la agenda de acciones para cada una de las interacciones que permite la aplicación.

Por cada uno de estos casos de uso, se presenta el diagrama de secuencia que contiene los métodos que entran en ejecución en la agenda, cuando se produce cada una de las llamadas.

Ejecutar una acción

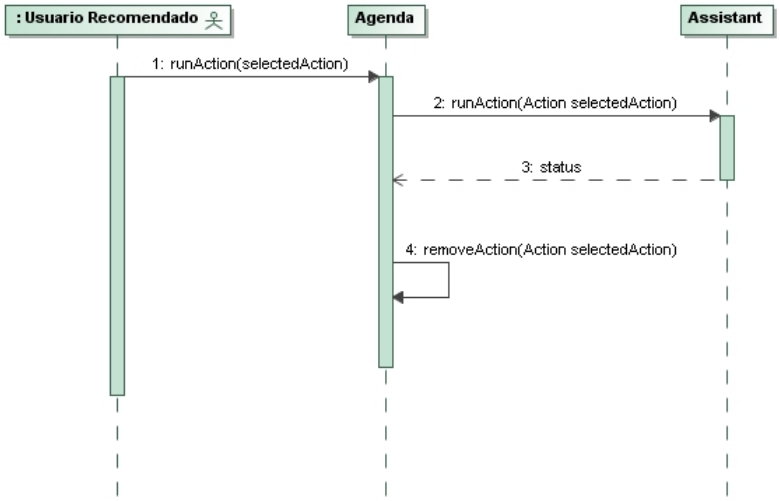

Figura 29 - Diagrama de secuencia de "Ejecutar acción"

En la Figura 29, se puede observar el flujo necesario para poder llevar a cabo la ejecución de una acción.

Cabecera del método	**runAction(Action selectedAction): void**
Invocado por	Usuario recomendado
Destino	Agenda
Parámetros	actionSelected: la acción seleccionada por el usuario

Retorno	void
Descripción	El usuario a través de la interfaz gráfica escoge la acción para su ejecución.

Cabecera del método	**runAction(Action selectedAction): void**
Invocado por	Agenda
Destino	Assistant
Parámetros	actionSelected: la acción seleccionada por el usuario
Retorno	void
Descripción	La agenda invoca al agente asistente para la ejecución de la acción seleccionada.

Cabecera del método	**removeAction(Action selectedAction): void**
Invocado por	Agenda
Destino	Agenda
Parámetros	actionSelected: la acción seleccionada por el usuario
Retorno	void
Descripción	La agenda tras haber sido ejecutado el servicio asociado a la acción seleccionada, borra dicha acción de la agenda.

Refinar acción

Como ya se explicó anteriormente, una de las posibilidades a realizar con una acción, es su refinamiento. Existen dos posibilidades:

- La particularización de dicha acción, lo que implica que la acción mostrada al usuario es demasiado genérica, y se precisa su división en diversas acciones que atiendan a objetivos más concretos.

- La generalización de una acción, al contrario del caso anterior, la acción mostrada en la agenda es demasiado particular y se precisa aglutinar varias acciones en una única que englobe los mismos objetivos que las acciones que se desean generalizar.

A continuación, se refleja en forma de diagrama de secuencia, qué ocurre en la agenda cuando se realiza cualquiera de éstas interacciones.

Particularizar acción

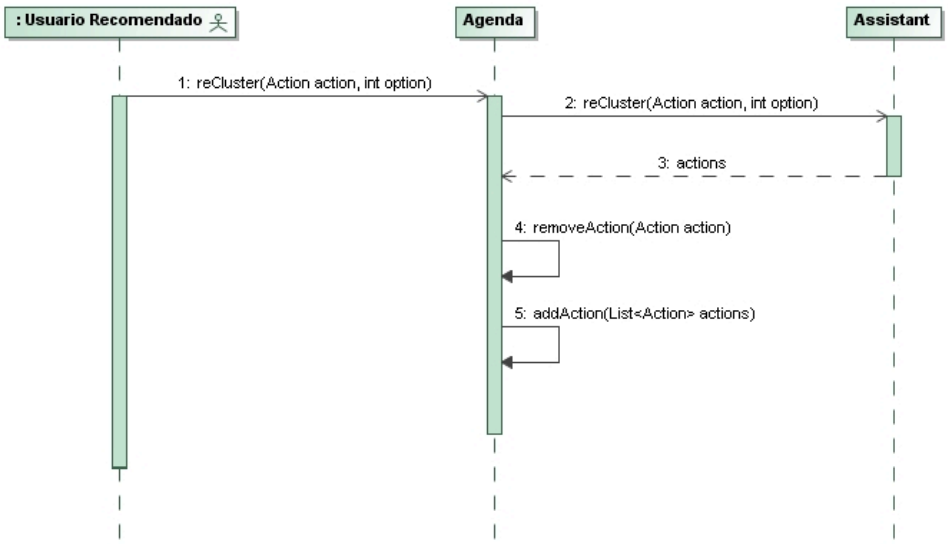

Figura 30 - Diagrama de secuencia de "Particular acción"

Cabecera del método	**reCluster(Action selectedAction, int option): void**
Invocado por	Usuario recomendado
Destino	Agenda

Parámetros	actionSelected: la acción seleccionada por el usuario. option: un entero que indica que la interacción deseada es particularizar acción.
Retorno	void
Descripción	El usuario a través de la interfaz indica que desea particularizar la acción seleccionada.

Cabecera del método	**reCluster(Action selectedAction, int option): List<Action>**
Invocado por	Agenda
Destino	Assistant
Parámetros	actionSelected: la acción seleccionada por el usuario. option: un entero que indica que la interacción deseada es particularizar acción.
Retorno	actions: una lista de acciones particularizadas.
Descripción	La agenda invoca el método del asistente encargado de la particularización de acciones.

Esta llamada al método *reCluster* se traduce en la agenda en la eliminación de la acción seleccionada por el usuario, y a continuación, una vez generadas las acciones nuevas (que cumplen los mismos objetivos que la acción eliminada), éstas se deberán añadir a la agenda para poder ser mostradas al usuario.

Cabecera del método	**removeAction (Action selectedAction): void**
Invocado por	Agenda

Destino	Agenda
Parámetros	actionSelected: la acción particularizada.
Retorno	void
Descripción	La agenda elimina de la lista de acciones, la que ha sido recién particularizada.

Cabecera del método	**addAction (List<Action> ations): void**
Invocado por	Agenda
Destino	Agenda
Parámetros	actions: lista de acciones a añadir.
Retorno	void
Descripción	La agenda añade la nueva lista de acciones que cumplen los mismos objetivos que la recién eliminada.

Generalizar acción

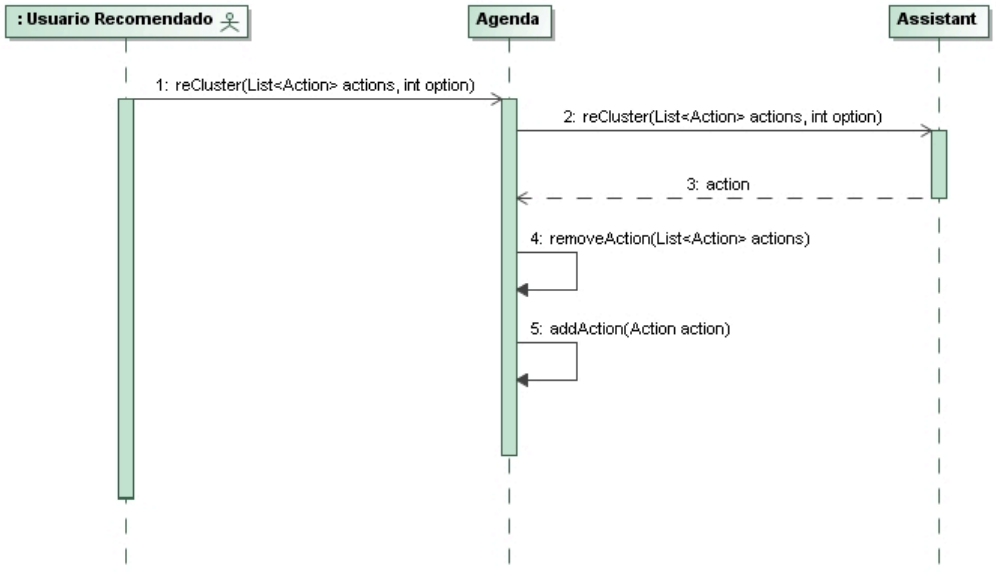

Figura 31 - Diagrama de secuencia "Generalizar acción"

La generalización de las acciones, implica que dado un grupo de acciones, éstas se aglutinan en una única acción que englobe los objetivos de las anteriores.

Cabecera del método	**reCluster(List<Action> actions, int option): void**
Invocado por	Usuario recomendado
Destino	Agenda
Parámetros	actionSelected: la lista de acciones seleccionadas por el usuario. option: un entero que indica que la interacción deseada es generalizar acción.
Retorno	void
Descripción	El usuario a través de la interfaz indica que desea generalizar las acciones seleccionadas.

Cabecera del método	**reCluster(List<Action> actions, int option): Action**
Invocado por	Agenda
Destino	Assistant
Parámetros	actionSelected: la lista de acciones seleccionadas por el usuario. option: un entero que indica que la interacción deseada es generalizar acción.
Retorno	action: una acción generalizada.
Descripción	La agenda invoca el método del asistente encargado de la generalización de acciones.

Cabecera del método	**removeAction (List<Action> actions): void**
Invocado por	Agenda
Destino	Agenda
Parámetros	actions: la lista de acciones generalizadas.
Retorno	void
Descripción	La agenda elimina de la lista de acciones, aquellas que acaban de ser generalizadas.

Cabecera del método	**addAction (Action ation): void**
Invocado por	Agenda
Destino	Agenda
Parámetros	action: acción a añadir.
Retorno	void
Descripción	La agenda añade la nueva acción que cumple los mismos objetivos que las recién eliminadas.

Borrar una acción de la agenda

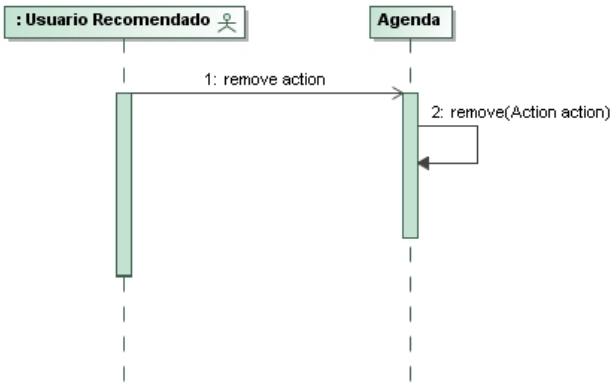

Figura 32 - Diagrama de secuencia de "Borrar acción"

En interacciones anteriores con la agenda, se ha mostrado la necesidad de eliminar acciones de la lista contenida en la agenda. En este caso, la eliminación de la acción se realiza bajo petición del propio usuario.

Cabecera del método	**remove (Action action): void**
Invocado por	Agenda
Destino	Agenda
Parámetros	action: la acción seleccionada.
Retorno	void
Descripción	La agenda elimina la acción de la lista

Añadir acción a la agenda

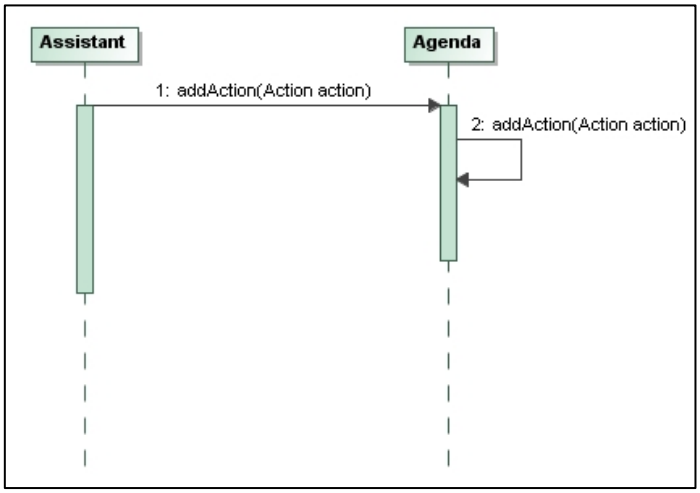

Figura 33 - Diagrama de secuencia de "Añadir acción a la agenda"

Al igual que en el caso de borrar acciones de la agenda, en interacciones anteriores ha quedado patente la necesidad de poder añadir nuevas acciones a la agenda.

Puesto que la agenda no es más que una lista de acciones (con una interfaz) el comportamiento mostrado por la agenda a la hora de recibir una llamada *addAction* es la de añadir a la lista de acciones, la nueva acción que recibe como parámetro en el método.

Cabecera del método	**addAction (Action action): void**
Invocado por	Assistant

Destino	Agenda
Parámetros	action: la acción a añadir
Retorno	void
Descripción	Se añade la nueva acción a la lista de la agenda

Ver detalle de una acción

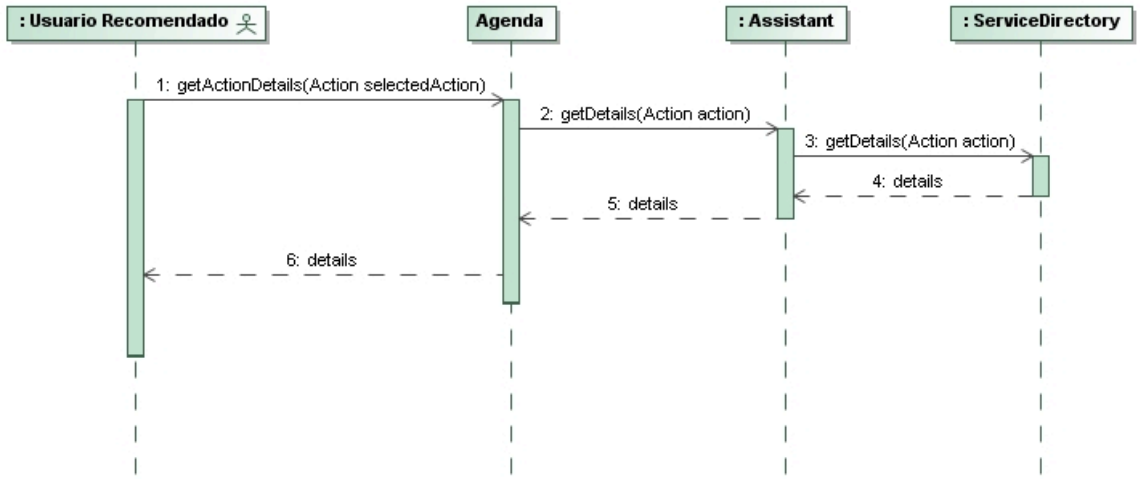

Figura 34 - Diagrama de secuencia de "Ver detalle de acción"

Es posible que el usuario desee obtener información adicional sobre cualquiera de las acciones mostradas en la agenda.

Cabecera del método	**getActionDetails (Action action): String**
Invocado por	Usuario recomendado
Destino	Agenda
Parámetros	action: acción de la que se desea obtener información adicional.
Retorno	Una cadena de texto con los detalles de la acción seleccionada.
Descripción	Obtiene los detalles de una acción para mostrárselos al usuario a través de la interfaz.

Cabecera del método	getActionDetails (Action action): String
Invocado por	Agenda
Destino	Assistant
Parámetros	action: acción de la que se desea obtener información adicional.
Retorno	Una cadena de texto con los detalles de la acción seleccionada.
Descripción	Obtiene los detalles de una acción.

Cabecera del método	**getActionDetails (Action action): String**
Invocado por	Assistant
Destino	ServiceDirectory
Parámetros	action: acción de la que se desea obtener información adicional.
Retorno	Una cadena de texto con los detalles de la acción seleccionada.
Descripción	Se trata de una llamada remota, puesto que se realiza al directorio de servicios, para la obtención de los detalles de una acción.

3.4.4. Diseño de la interfaz en dispositivos móviles

Recordemos, tal y como se comentaba en el apartado 2.2, uno de los objetivos del proyecto es el diseño de la interfaz de usuario del agente recomendador centrado en plataformas móviles. Dado que el proyecto se basa en la idea de movilidad, el agente inteligente debe tener una interfaz gráfica en el dispositivo móvil para poder interaccionar con el usuario.

En esta sección se explicará el proceso llevado a cabo para diseñar la interfaz del agente inteligente en dispositivos móviles, utilizando la plataforma móvil Android.

3.4.4.1. Introducción a Android

Android es una plataforma para dispositivos móviles que ha sido utilizada para llevar a cabo el desarrollo de la interfaz del agente.

Esta plataforma proporciona diferentes componentes específicos para aplicaciones e interfaces gráficas. Para la implementación concreta de esta aplicación para dispositivos móviles, se han usado los siguientes:

- *Activity*: una actividad representa una pantalla individual en el terminal y presenta una interfaz gráfica de usuario, que a su vez está formada por un conjunto de vistas (*Views*) que responden a diferentes eventos, fruto de la interacción con el usuario.

- *Intent*: proporciona las herramientas necesarias para realizar el enlace en tiempo de ejecución entre el código de diferentes componentes. Se trata de un sistema de comunicación entre los diferentes componentes Android.

- *Application*: Se trata de la clase base, que permite mantener el estado de la aplicación que se está desarrollando. Por otra parte, la aplicación es el componente central, hace referencia al conjunto del software agrupado en un paquete (*Apk*). Este archivo permite ser instalado y ejecutado en los terminales.

3.4.4.2. Diagrama de clases de la interfaz en dispositivos móviles Android

En la Figura 35 se puede observar el diagrama de clases correspondiente a la interfaz para dispositivos móviles diseñada teniendo en cuenta los componentes que proporciona la plataforma móvil Android.

En este diagrama se puede ver el paquete general que aglutina todas las clases que intervienen en la interfaz gráfica.

Existen dos tipos de clases representadas en éste: las clases Java genéricas y otras clases que han sido estereotipadas con *«Activity»* haciendo referencia al componente *Activity* que representa una pantalla en la plataforma Android.

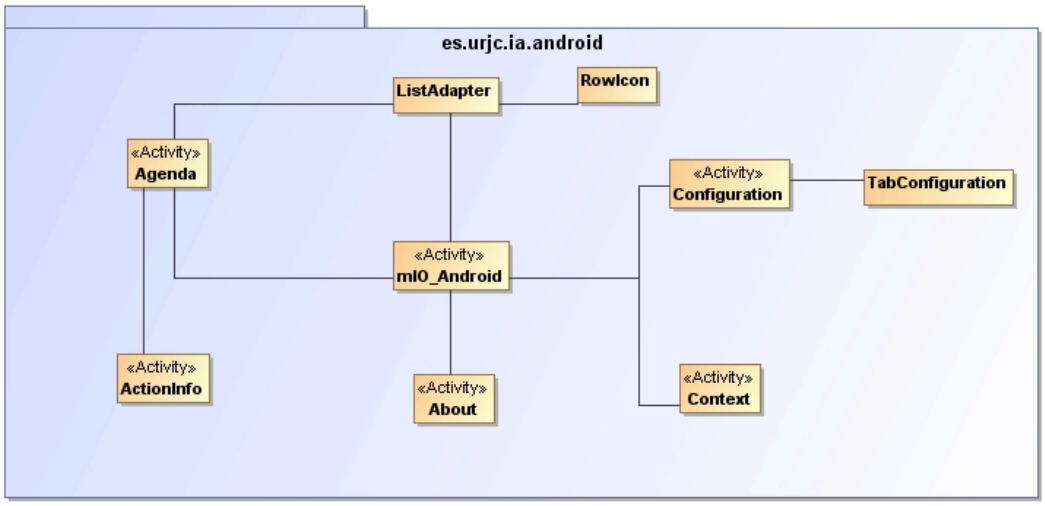

Figura 35 - Diagrama de clases de las actividades de la interfaz

A continuación, se explica cada una de estas clases así como su participación en la interfaz diseñada.

mIO_Android

Se trata de la actividad principal. Representa la pantalla inicial de la aplicación. Esta actividad tiene relación con la clase Java *ListAdapter*, porque el aspecto que muestra esta pantalla es el de una lista de ítems en forma de enumeración.

Además, esta actividad principal, tiene relación con todas las demás actividades mostradas en el diagrama, puesto que cada una de estas actividades se corresponde con una pantalla a la que es posible navegar a través de la principal.

Agenda

La actividad *Agenda* (Figura 35) representa la interfaz de la agenda de acciones. Esta actividad al igual que la principal, tiene relación directa con *ListAdapter* puesto que muestra en forma de lista todas las acciones posibles en un momento dado para el usuario.

Además, también tiene una relación directa con la pantalla principal, puesto que es a partir de ésta desde la que se llega a la actividad *Agenda*.

ActionInfo

Cada una de las acciones que son mostradas en la actividad *Agenda* (Figura 35), como ya se ha mencionado anteriormente tiene una serie de interacciones posibles: borrar, refinar, poder consultar información o ver detalles de la acción. La actividad *ActionInfo* es la encargada de mostrar los detalles adicionales asociados a una acción concreta.

Es precisamente desde la actividad *Agenda*, donde se puede acceder a la información específica de cada una de estas acciones.

About

Se trata de una actividad que muestra información relativa al proyecto en el cual está enmarcado el presente Proyecto Fin de Carrera.

Al igual que la *Agenda*, se accede a esta actividad a través de la pantalla principal de la aplicación.

Context

La actividad *Context* existe tan sólo en el ámbito del prototipo que se muestra en este Proyecto Fin de Carrera.

Puesto que generar la información contextual no es objetivo de este proyecto, existe una pantalla representada a través de la actividad *Context* que nos permite simular cambios de contexto en el entorno del usuario. Por lo que, es tan sólo una actividad que simula el contexto de la aplicación, pudiendo cambiar entre dos contextos diferentes dados y que permiten demostrar la ejecución de la aplicación en dos escenarios diferentes.

Esta actividad ha sido implementada en esta versión del prototipo con fines de prueba.

Configuration

Al igual que la pantalla referente al contexto, se trata de una actividad que existe únicamente con fin de poder probar el prototipo desarrollado.

La actividad dedicada a la configuración, consiste en una pantalla que contiene todos los parámetros que son necesarios para poder probar el prototipo diseñado en el actual proyecto. Contiene información del tipo: lugar en el que se encuentra el directorio de servicios (identificado a través de dirección IP y puerto), umbral mínimo aceptado para el *matchmaking* de servicios y la URI del recurso *REST* que atiende las peticiones.

Además, la clase *Configuration* tiene relación con la clase Java *TabConfiguration* puesto que el aspecto de la actividad incluye un gestor de pestañas. La clase *TabConfiguration* es la que se encarga por tanto, de la gestión de las pestañas.

A la pantalla de configuración se accede a través de la pantalla principal de la aplicación.

3.4.4.3. Diagrama de instancias de la interfaz de dispositivos móviles Android

En el apartado anterior se ha explicado el diagrama de clases asociado a la interfaz para dispositivos móviles Android.

En el presente apartado, se muestra el diagrama de clases asociado a las instancias de las clases explicadas en el apartado anterior, con el fin de mostrar cómo se crean las instancias de las clases que se han desarrollado para la demostración de la aplicación que rodea a este Proyecto Fin de Carrera.

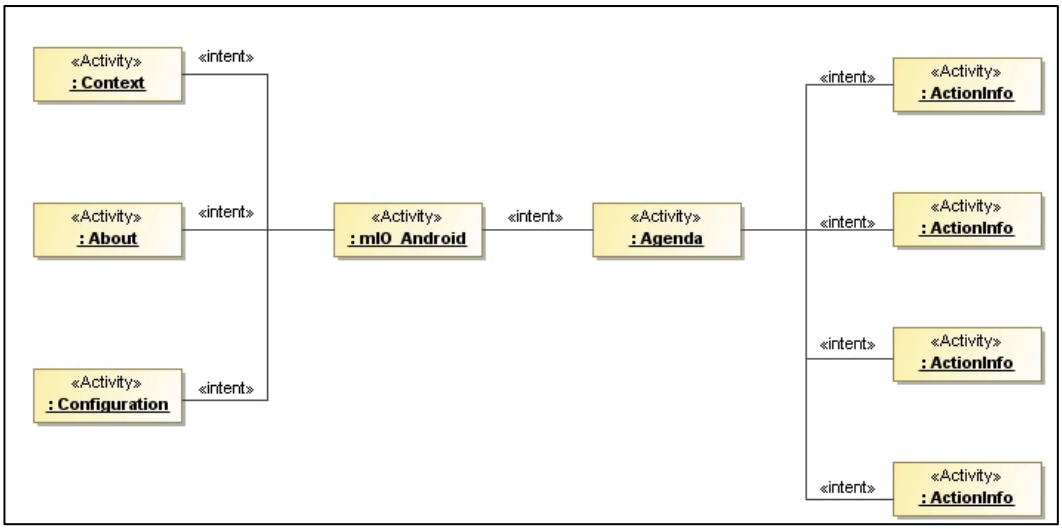

Figura 36 - Diagrama de instancias de la interfaz

Tal y como se puede observar en la Figura 36 existen dos tipos de estereotipos asociados a las clases mostradas en el diagrama:

- el estereotipo *«Activity»*, hace referencia al componente *Activity* de la plataforma Android. En este caso, este estereotipo indica que se trata de clases Java que en el ámbito de Android se traduce en interfaces gráficas.

- Por otro lado, el estereotipo *«intent»* hace referencia al componente asociado a la plataforma Android y que fue explicado anteriormente en el apartado dedicado a la plataforma Android. Este componente permite la navegación entre distintos componentes y el intercambio de información entre éstos.

El diagrama de la Figura 36 refleja las instancias que son creadas en la ejecución de uno de los escenarios diseñados para el prototipo y expuesto en la sección 2.1 dedicada al caso de estudio.

Al ser un diagrama de instancias, refleja el número concreto de éstas que son creadas durante la ejecución de la aplicación. Existe una única instancia de determinadas clases que son creadas una vez y utilizadas durante el resto de la ejecución (Context, About, Agenda, etc.), pero existen sin embargo, multitud de instancias del componente *intent* de Android, así como de la actividad *ActionInfo*. Estos *intent* son utilizados para la navegación entre componentes, en este caso entre actividades Android, por lo que es necesario crear diferentes instancias para poder intercambiar distintos parámetros entre éstos. Por último, existen además diferentes instancias de la actividad *ActionInfo*, ya que se crea una por cada acción que es mostrada en el escenario, tal y como veremos en la sección 3.6.

3.5. Implementación

En este apartado de implementación, se distribuye el sistema en términos de componentes, que son empaquetamientos físicos de uno o varios elementos del modelo de diseño.

A través de los diagramas de despliegue, se va a especificar cómo se organizan estos componentes y cómo dependen unos de otros dentro del sistema implementado.

Esa organización de los componentes, junto con los detalles más importantes de implementación llevados a cabo y las herramientas utilizadas para desarrollar el proyecto, son explicados en esta sección.

3.5.1. Tecnologías utilizadas

A continuación, se explican cada una de las tecnologías que han sido utilizadas para el diseño e implementación de este proyecto. Además, se hará referencia a las herramientas que se han empleado para facilitar este proceso de desarrollo.

3.5.1.1. Java

Java es un lenguaje de programación y plataforma informática, lanzado por primera vez por *Sun Microsystems* en 1995. Se trata de un lenguaje de programación que sigue el paradigma de orientación a objetos.

Más concretamente, se ha utilizado la versión Java 1.6.

3.5.1.2. Eclipse

Eclipse es el IDE (*Integrated Development Environment*) utilizado para el desarrollo del código fuente del proyecto. Éste brinda las herramientas necesarias para la construcción, despliegue y gestión del software a lo largo de su ciclo de vida.

Más concretamente, se ha utilizado la versión Eclipse Helios 3.6

3.5.1.1. Android

La implementación del agente en el dispositivo móvil se ha llevado a cabo utilizando la plataforma Android en su versión 1.5.

Android es una plataforma de código abierto para dispositivos móviles que está basada en Linux, fue desarrollado por la *Open Handset Alliance*, un conglomerado de fabricantes y desarrolladores de hardware, software y operadores de servicio.

Para el desarrollo de la parte relacionada con el dispositivo móvil, se ha utilizado además el *plugin Android Developer Tools* (ADT) para *Eclipse IDE*, diseñado para ofrecer un poderoso entorno integrado para construir aplicaciones *Android*.

3.5.1.2. Svn

Subversion es un sistema de control de versiones que ha permitido una gestión eficiente de la configuración en el sistema construido en el presente proyecto.

Permite, la gestión de las versiones de los distintos documentos generados, a través de un sistema de repositorio.

3.5.1.3. Trac

Trac es una *wiki* y un sistema de seguimiento para los proyectos de desarrollo software. Utiliza una enfoque para la gestión de proyectos basado en web, tratando de proporcionar a los desarrolladores, herramientas para el seguimiento de los proyectos, sin imponer procesos o políticas establecidas.

Permite además la integración con el sistema *Subversion*.

3.5.1.4. Log4J

Es una *API* de código abierto desarrollada por *Apache Software Foundation* para manejar el registro de trazas en las aplicaciones Java. Provee un *framework* robusto, extensible, configurable y sencillo para el registro de trazas en aplicaciones Java.

Se ha utilizado la versión integrada de *Log4J* con *Eclipse*, a través del *plugin Log4E* que permite la gestión de trazas sobre diversos componentes.

3.5.1.5. Restlet

Es un *framework REST* de código abierto, para la plataforma Java. Permite por tanto, trabajar con la arquitectura REST (basada en *HTTP*) a través de código Java.

La versión que se ha utilizado, es *Restlet* 1.2

3.5.1.6. xStream

Librería de código abierto que permite la serialización y desearilización de objetos a *XML*. La versión utilizada ha sido la 1.3.1

3.5.1.7. Db40

Db4O (*db4objects*) ofrece una base de datos de objetos de código abierto, ha sido utilizada para almacenar información en el dispositivo móvil. La versión del software utilizada ha sido la 7.12.

3.5.1.8. MagicDraw

Software de soporte utilizado para el análisis y diseño del sistema, permitiendo realizar el modelado del sistema a través del uso de UML.

Más concretamente, la versión utilizada ha sido *MagicDraw* 16.6

3.5.1.9. Mircrosoft® Word

Procesador de textos utilizado para la redacción de la memoria del Proyecto Fin de Carrera.

La versión utilizada ha sido Word 2011 para Mac 14.0.0

3.5.2. Diagrama de despliegue

Para dar una vista física de todo el proyecto, se presenta un diagrama de despliegue (Figura 37) cuya finalidad es la de mostrar los nodos individuales del sistema, así como los enlaces y componentes que ejecutan en cada uno de éstos.

En la figura se pueden observar dos nodos principales:

- el nodo correspondiente al dispositivo móvil
- el nodo que representa el servidor en el que se encuentra el directorio de servicios

También se puede observar que existe un componente externo relacionado con el resto de nodos del sistema y que contiene a su vez las librerías que son necesarias para la ejecución del resto de componentes del sistema.

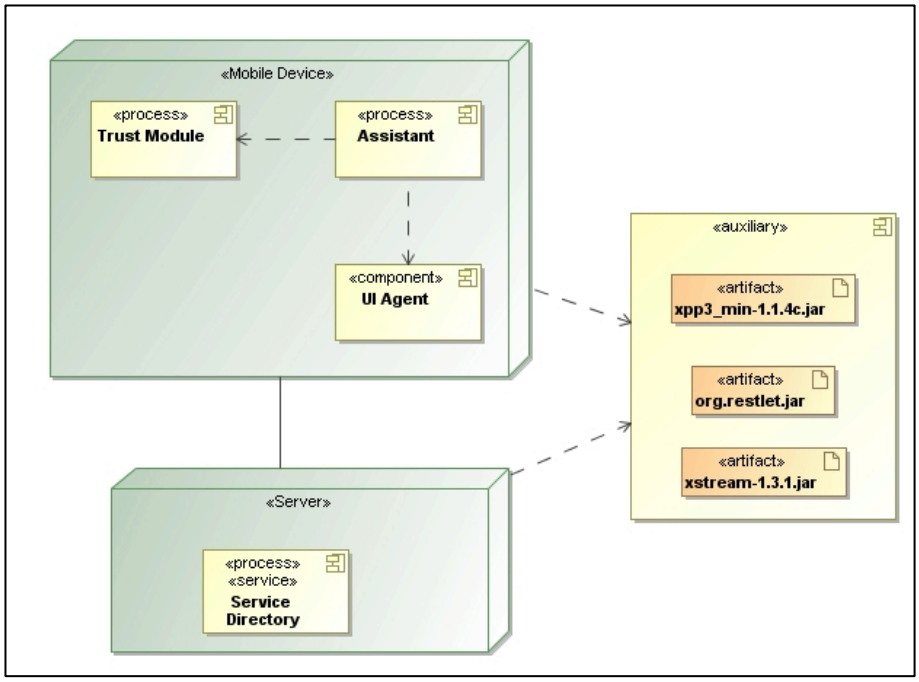

Figura 37 - Diagrama de despliegue

En el diagrama de despliegue anterior se muestra cómo se trata de un sistema distribuido, formado por tres diferentes nodos que interactúan entre sí.

El nodo correspondiente al dispositivo móvil contiene tres componentes fundamentales: *Assistant, Trust Module y UI Agent*. El asistente (*Assistant*) que representa al agente

encargado de la recomendación, el cual interactúa con el componente llamado *Trust Module* es el encargado de llevar a cabo la ordenación de los proveedores de servicios en términos de confianza. Por último, existe un último componente representado dentro del nodo del dispositivo móvil y que hace referencia a la interfaz gráfica de usuario que se ha desarrollado en la plataforma Android y que hace las veces de interfaz de la agenda de acciones.

El segundo nodo del diagrama de despliegue, el directorio de servicios que, como ya se ha comentado en ocasiones anteriores, se trata de un módulo externo con el que interactúa el agente para la obtención de la información referente a los servicios a ejecutar. Además, la interacción del móvil con el servicio de directorio se realiza a través de la invocación a un servicio web, que provee de información relativa a los servicios (tanto plantillas, como instancias).

Por otra parte, existe un componente externo que aglutina las librerías que hacen falta para la correcta ejecución del resto de componentes de los nodos representados. Dichas librerías son necesarias para la comunicación entre los nodos remotos, tanto la librería *xStream* encargada del aplanamiento de la información para llevar a cabo la comunicación en formato texto, como la librería *restlet* que ha sido utilizada para la comunicación entre nodos siguiendo la arquitectura *REST*.

Pese a que en el diagrama de despliegue representado anteriormente, hace referencia a la existencia de dos nodos: el dispositivo móvil y el servicio de directorio, sería totalmente factible que el nodo correspondiente al dispositivo móvil, fuese sustituido por un nodo que ejecutase en un dispositivo no móvil, es decir, en cualquier tipo de plataforma de computación no móvil.

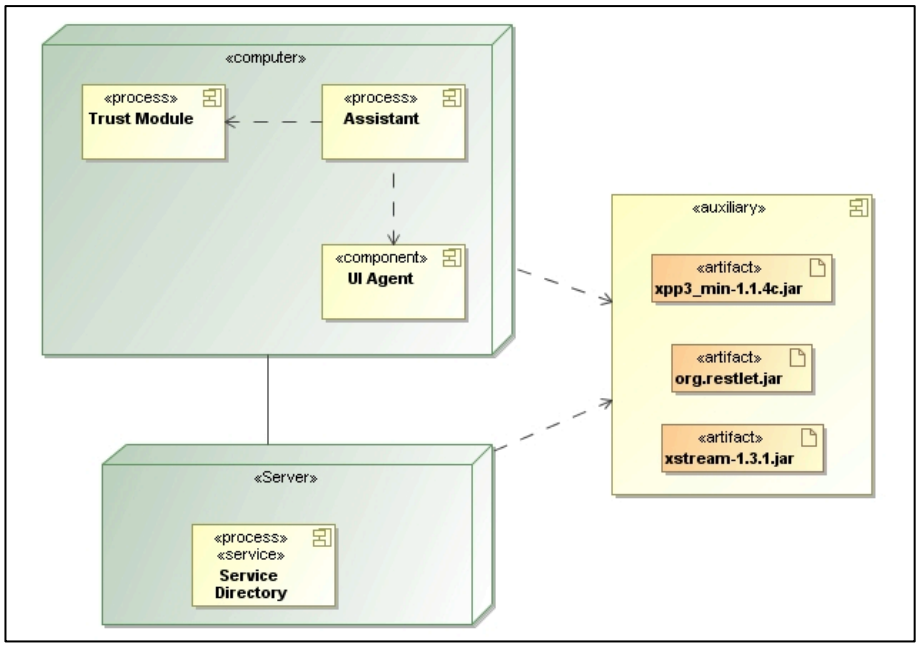

Figura 38 - Diagrama de despliegue no móvil

En los objetivos del presente proyecto se puntualizó que la arquitectura propuesta para la recomendación de servicios, podría ser igualmente utilizada en dispositivos no móviles.

En la Figura 38 se presenta cómo quedaría la distribución física de los elementos computacionales, en caso de que ejecutarse en un dispositivo no móvil.

Si comparamos ambos diagramas de despliegue, se comprobaría que no existe diferencia en cuanto a la distribución de los elementos en los nodos, por lo que, queda demostrado que la arquitectura propuesta es igualmente válida para la ejecución en ambos tipos de dispositivos.

3.5.3. Detalles de implementación

En este apartado se da una visión global de cuáles son los detalles de implementación, todas aquellas decisiones y problemas que han ido surgiendo a lo largo de la implementación del asistente inteligente.

También se muestra una vista física del sistema, a través de capturas de pantalla y fotografías del dispositivo en ejecución.

3.5.3.1. Gestión de los parámetros de configuración del agente

Para poder llevar a cabo el proceso de recomendación, el agente asistente precisa de cierta información que se le debe proveer en la configuración de la aplicación.

Estos parámetros de configuración que son necesarios establecer, son los relativos al Directorio de Servicios:

- *Host y puerto*: información relativa a los parámetros de conexión con el Directorio de Servicios
- *Recurso*: hace referencia a la dirección del recurso (la URI) sobre la que se pueden realizar las peticiones REST al Directorio de Servicios.
- *Umbral (threshold)*: umbral mínimo de encaje exigido para los servicios del Directorio.

3.5.3.2. Gestión de la falta de conectividad

El agente asistente inteligente que se encuentra alojado en el dispositivo móvil precisa de conexión a internet para poder llevar a cabo su función.

La tecnología ha avanzado mucho en los últimos años, tanto que ahora concebimos la idea de estar conectados de manera continua a través de un dispositivo móvil. El prototipo de asistente inteligente móvil se basa en esta idea, por lo que se hace necesario comprobar la posible falta de conectividad producida por algún tipo de problema.

Para ello, el agente alojado en el dispositivo móvil al iniciar la ejecución, detecta la existencia o ausencia de conectividad a través de la realización de una conexión *HTTP*: una petición *GET* a la dirección (identificada por *IP* y puerto) que se ha facilitado en los parámetros de configuración.

El controlar este tipo de situaciones nos da la posibilidad de recuperar a la aplicación de un posible problema de falta de conectividad (dotarle de robustez), informando al usuario en caso de que se produjera esa situación.

3.5.3.3. Detalles de persistencia de datos en el dispositivo móvil

En apartados anteriores se ha puesto de manifiesto la necesidad del almacenamiento permanente de datos, como es el caso de la información de confianza.

Dicha información de confianza almacenada de forma local en el módulo de confianza del agente asistente, es preciso que se encuentre disponible con el paso del tiempo y de forma permanente, para ello se ha utilizado una Base de Datos que aloja la *LIT* que recoge la información de las interacciones pasadas el usuario con distintos proveedores de servicios.

La base de datos *db4o* (*data base for objects*) ha permitido el almacenamiento permanente en el dispositivo móvil. El uso de esta tecnología junto con la plataforma *Android* generó ciertos problemas en cuanto a las limitaciones que la base de datos originaba.

Más concretamente, todas las transacciones que realiza el agente con la base de datos debían ser llevadas a cabo en el contexto de una *Activity* de *Android*. Es decir la apertura, volcado de información y cierre de la base de datos, debía ser realizada en un componente de tipo *Activity*, lo que planteaba ciertas limitaciones.

Es por esto que la comunicación con la base de datos se realiza al inicio y final de la interacción con el usuario, usando mientras tanto almacenamiento local (una *caché*) de dicha información.

3.5.3.4. Traslado de la lógica de búsqueda de servicios a la plataforma

Puesto que la implementación se ha realizado sobre la plataforma *Android*, esto presenta ciertas limitaciones en cuanto a capacidad de cómputo se refiere, que se pasan a exponer en este apartado.

Los dispositivos móviles con el paso del tiempo han ido aumentando su capacidad de cómputo, aunque, existen restricciones (sobre todo relacionadas con la batería y el diseño físico del dispositivo) que hacen que no sea equiparable a un dispositivo de sobremesa.

La arquitectura fue inicialmente pensada para ser implementada en dispositivos móviles, aunque fuera probada en otros tipos de dispositivos también. El estar implementado sobre *Android* ha hecho que existan ciertas restricciones, como es el hecho de que no ha sido posible utilizar ciertas librerías (en forma de *JAR*) en los proyectos de tipo *Android Project* del entorno de desarrollo *Eclipse*.

Una vez detectada la problemática relacionada con la incompatibilidad de las librerías *JAR* (*Java Archive*) con la plataforma *Android*, se trató de llevar a cabo dos soluciones:

- En primer lugar, se trataron de recompilar las librerías con un compilador más moderno (Java 6) e intentar incluir esta nueva versión de las librerías en el proyecto, pero esta solución no solventó los problemas que habían surgido hasta entonces.

- La segunda solución, fue tratar de generar paquetes en el proyecto de *Android* para Eclipse, para así poder añadir el código asociado a estas librerías, pero esta solución tampoco fue posible puesto que había librerías como *log4Java* que presentaban problemas.

La imposibilidad de incluir librerías *JAR* hizo que parte de la lógica del proyecto, tuviera que ser trasladada hacia el directorio de servicios. Este fue el caso de las librerías relacionadas con semantización de los servicios (la librería semántica *Jena*), lo que implicó que toda la lógica de búsqueda y *matching* de servicios se realizara en el propio directorio de servicios, en lugar de en el dispositivo móvil, así como la ejecución propiamente dicha.

Para evitar la presencia de las librerías de semantización de servicios, en el asistente del dispositivo móvil, la solución por la que se optó, es que todo el tráfico entre el asistente móvil y el directorio de servicios, se hiciera en un formato comprensible para ambos extremos, de ahí que se optara por que la comunicación se realizase en texto plano. De forma que el asistente, a la hora de realizar búsquedas en el directorio, debería pasar como parámetros de búsqueda la consulta y un umbral mínimo de encaje.

Dentro del directorio se lleva a cabo todo el proceso de búsqueda de servicios, tanto plantillas como instancias, que encajen con el dado, en un nivel mínimo considerado a través del umbral.

3.5.3.5. Detalle de la Interfaz en dispositivo móvil

A continuación, mediante capturas de pantalla de la interfaz del dispositivo móvil, se explicará brevemente cada una de las funcionalidades relevantes de la aplicación.

En la Figura 39, se muestra la pantalla principal de la aplicación. En la parte superior de la pantalla se muestra el nombre de la aplicación, *mIO! Personal Assistant*, junto con el logotipo. En la parte inferior, una lista de todas las posibilidades que nos brinda al inicio de la aplicación.

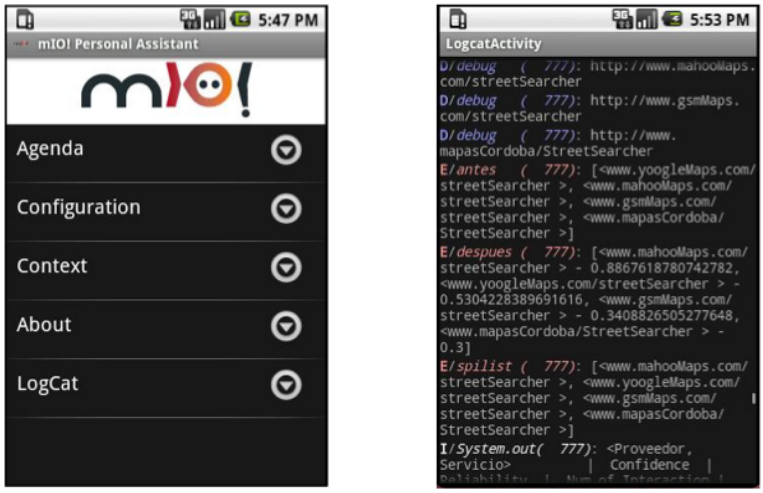

Figura 39 - Captura de la pantalla principal y el logger de la aplicación

LogCat

El nombre "*LogCat*" está referido a la herramienta de monitorización que ofrece la plataforma de *Android*, de forma que se ha aprovechado para poder analizar en todo momento, qué está realizando el dispositivo.

Configuration

Esa sección está destinada a configurar parámetros propios de la aplicación. En particular, está dedicado a la configuración de la localización de los distintos componentes que puedan estar distribuidos en la red y que no estén dentro de la arquitectura de los agentes inteligentes asesores. Estos parámetros pueden observarse en la Figura 40.

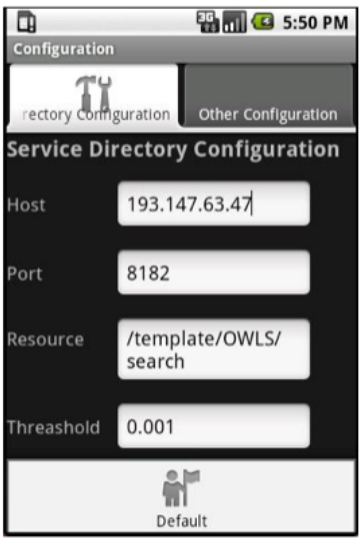

Figura 40 - Captura de pantalla de configuración

Agenda de acciones

La interfaz principal de los agentes inteligentes con el usuario, es la agenda de acciones. En la Figura 41 que se puede ver a continuación, se muestra el estado de la agenda para el escenario "Turismo en Córdoba".

Dentro de cada fila de la lista, se pueden observar dos ítems, por una parte el nombre de la acción a realizar, y en segundo lugar, se ofrece una breve descripción de qué consiste la acción a realizar.

Cada una de estas acciones tiene una serie de interacciones asociadas, esto es, todas las posibilidades que existen a realizar con dicha acción. Si se pulsa de forma continuada sobre cualquiera de ellas, aparecerá un menú desplegable con las siguientes acciones:

- ***Run***: ejecutar la acción seleccionada
- ***Remove***: eliminar de la agenda la acción seleccionada (y evitar de alguna forma que se pueda volver a generar ese tipo de acción en el futuro).
- ***Zoom in***: particularizar la acción, en el caso de que se trate de una acción relacionada con los deseos del usuario, pero no concretamente esa acción, entonces se procede a una redefinición en acciones más concretas.
- ***Zoom out***: este es el caso contrario al anterior, en caso de ofrecer al usuario una acción demasiado específica, se redefine tratando de hacerla más genérica.

Este es el menú desplegable que aparece al pulsar de forma continuada sobre una acción, sea cual sea la interacción deseada, se abrirá una nueva pantalla, en la que se mostrará información más específica sobre dicha acción, para poder dar al usuario la oportunidad de informarse sobre ella, antes de tomar una decisión.

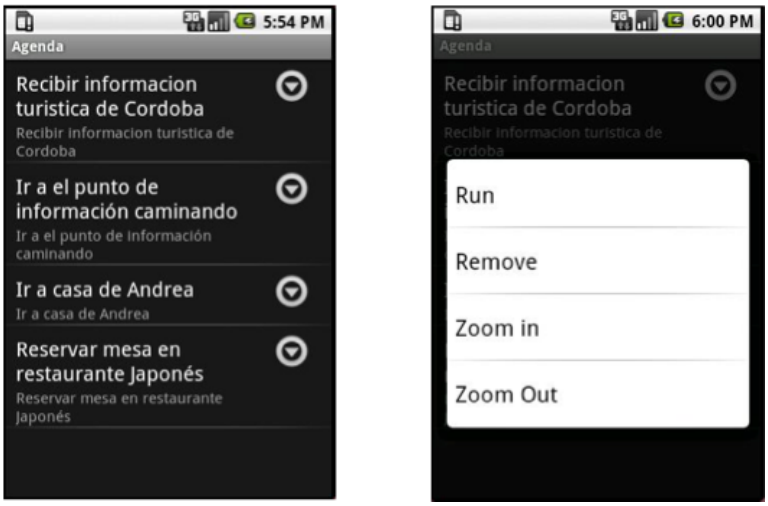

Figura 41 - Captura de pantalla de la agenda y del menú desplegable de una acción

En el caso de seleccionar una de las acciones, se ofrecen más detalles de la acción indicada y se ofrecen las distintas opciones a realizar sobre una acción: las de refinamiento y la acciones directas de ejecutar la acción o eliminarla.

Figura 42 - Captura de pantalla de detalles de una acción y menú desplegable de una acción

Al solicitar la ejecución de una de las acciones, en primer lugar se pide la confirmación al usuario y tras su ejecución, se solicita la evaluación subjetiva de su satisfacción con la ejecución de dicho servicio.

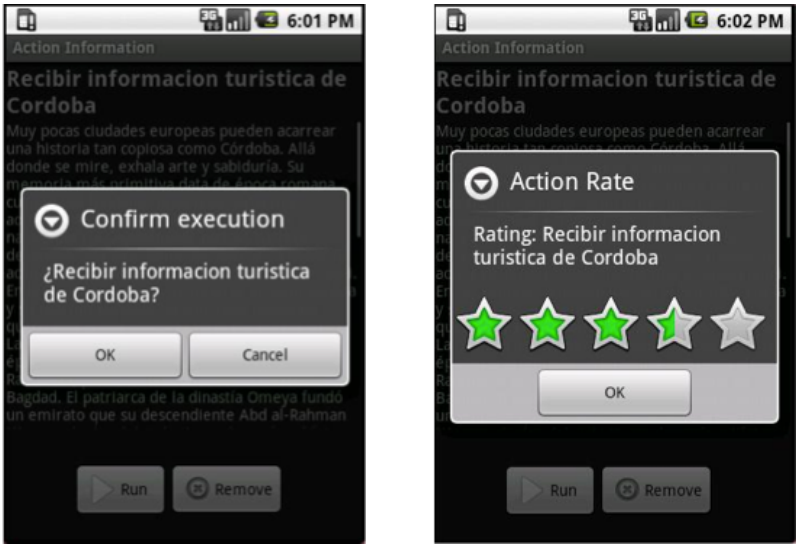

Figura 43 - Captura de pantalla de confirmación de ejecución y valoración de una acción

3.5.3.6. Implementación para otros dispositivos

Pese a que el proyecto está orientado a la ejecución en un dispositivo móvil, la arquitectura ha sido diseñada pensando en la ejecución en cualquier tipo de dispositivo.

Tanto es así, que para poder demostrar el funcionamiento correcto de la arquitectura diseñada, se hizo una versión para *Java* y se implementó una interfaz *Swing (*Figura 44*)*. Además, para demostrar su correcto funcionamiento, se ejecutó este prototipo en un dispositivo móvil *HTC Shift* (dispositivo de tamaño e interfaz móvil, pero con software de sobremesa).

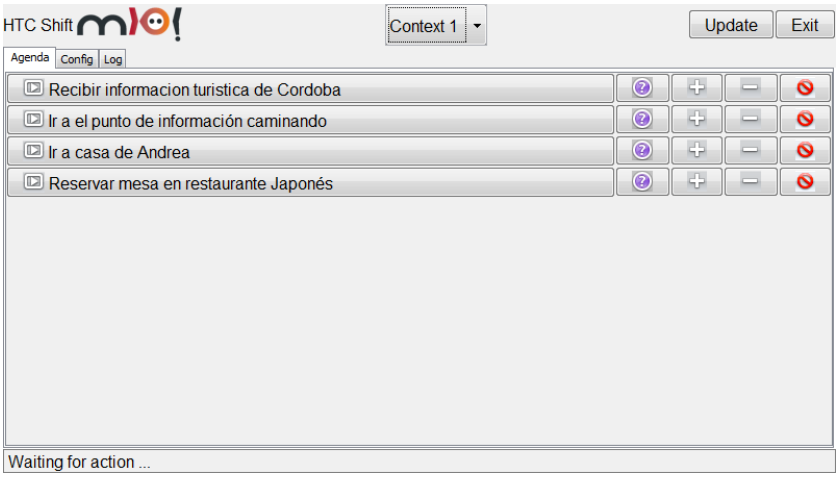

Figura 44 - Captura de agenda de acciones en interfaz Swing

En la Figura 45 se observa el equivalente a la pantalla principal desarrollada para el dispositivo móvil. En este caso, se muestran en forma de lista todas las acciones disponibles para el usuario recomendado en ese momento.

Además por cada una de las acciones de la lista, se permite realizar las interacciones correspondientes a la agenda de acciones y explicadas a través de casos de uso (refinar, ejecutar, ver detalles de la acción).

En la siguiente figura se puede observar la interfaz *Swing* cuando se consultan los detalles de una de las acciones provistas en la lista:

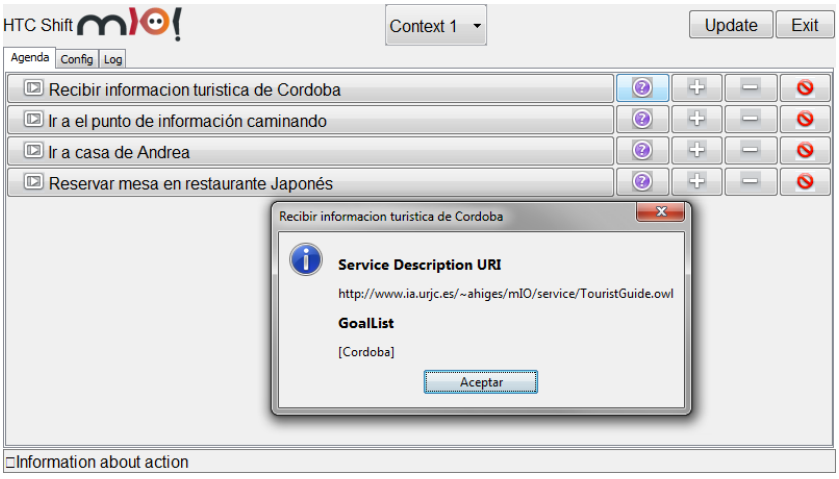

Figura 45 - Captura de detalles de acción en Swing

Otra de las posibles interacciones permitidas es la ejecución del servicio, en este caso al tratarse de un prototipo, la ejecución del servicio ofrecido por el proveedor es simulado y así se indica en la interfaz.

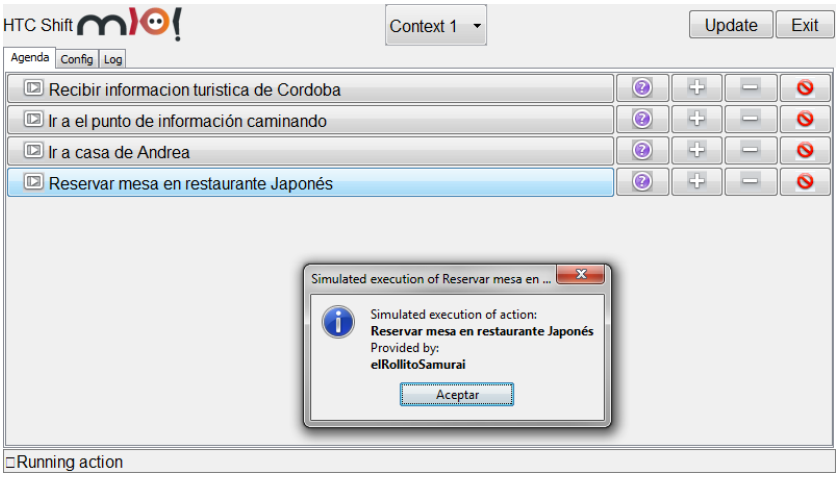

Figura 46 - Captura de simulación de ejecución en interfaz Swing

En la ejecución de la acción, se indica qué proveedor exacto ofrece el servicio.

Tras la ejecución del servicio, al igual que en el caso del dispositivo móvil, es necesaria la valoración del proveedor de servicio por parte del usuario recomendado, proporcionando la necesaria retroalimentación al módulo de confianza.

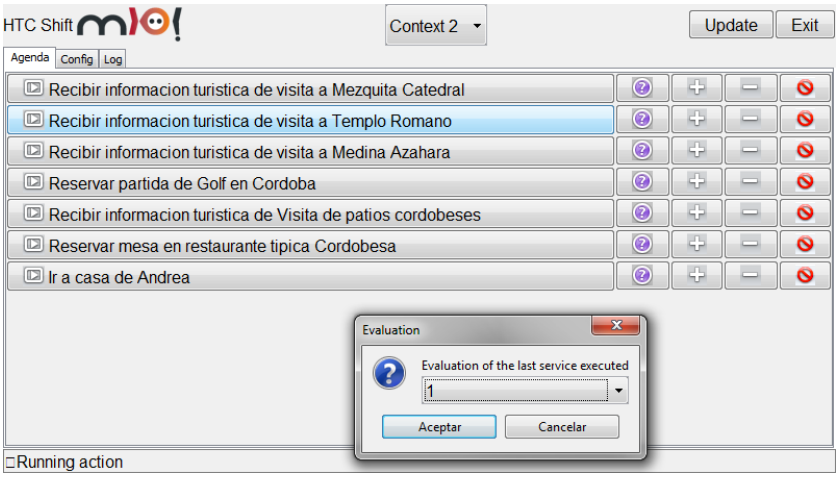

Figura 47 - Captura de valoración de ejecución en interfaz Swing

La valoración dada por parte del usuario debe estar comprendida entre los valores 0 a 5, siendo valores bajos una mala valoración del proveedor seleccionado. Tras ser valorado, el servicio será eliminado automáticamente de la lista de acciones.

3.5.3.7. Detalle de ejecución en dispositivos móviles físicos

En este apartado se demuestra cómo la ejecución se ha llevado a cabo no sólo en el simulador que ofrece la herramienta de *Android* para *Eclipse* (*ADT plugin*), sino que la

aplicación se ha probado en diversos dispositivos físicos reales como se puede ver a continuación:

Figura 48 - Ejecución en dispositivos físicos

Además, se ha comprobado cómo el diseño inicial de la interfaz gráfica del agente asistente realizado para la versión 1.5 de Android, es totalmente funcional para versiones posteriores como la 1.6, 2.2 y 2.3 de este sistema operativo. Las capturas de la pantalla anteriores, muestran el detalle de ejecución de la aplicación en distintas versiones del sistema operativo.

3.6. Evaluación

En este apartado se realizará la evaluación del prototipo construido, a partir de la demostración del funcionamiento del sistema.

A partir de capturas de pantalla del dispositivo móvil se irá mostrando cuál es el estado del sistema en cada uno de los estados de la recomendación, siguiendo el caso de estudio que ha sido planteado en la sección 2.1"Turismo en la ciudad de Córdoba".

Cuando el usuario recomendado, en este caso Paula, abre la aplicación móvil se le muestra la pantalla principal, en la que se encuentra una lista con todas las acciones que puede realizar ésta en ese momento.

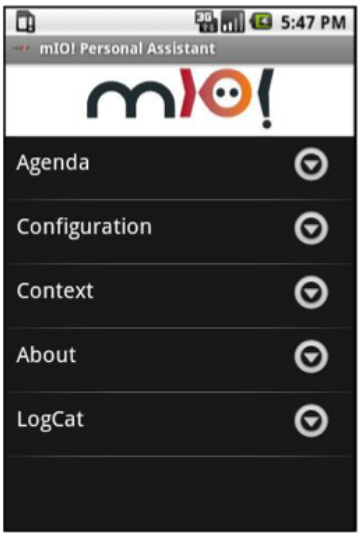

Figura 49 - Pantalla principal de la aplicación

Puesto que se encuentra haciendo turismo por la ciudad de Córdoba, y es la primera vez que la visita, consulta la agenda de acciones para que así el asistente recomendador, le sugiera algún sitio o ruta para visitar la ciudad.

Al pulsar sobre "*Agenda*" en la lista de la pantalla principal, el agente recomendador arrancará el proceso de recomendación, consultando información del entorno (señal *GPS*, hora, temperatura ambiente…) para determinar los objetivos de Paula y poder así construir las acciones que aparecerán en la pantalla de la Agenda, como muestra en la Figura 50.

Figura 50 - Agenda de acciones ofrecida a Paula

Una vez mostrada la lista de acciones, vemos cómo el asistente ha detectado la nueva localización de Paula, y las acciones mostradas son:

- Recibir información turística de "Tour de arte y cultura en Córdoba"
- Reservar partida de Golf en Córdoba
- Recibir información turística de "Vista a patios Cordobeses".

- Reservar mesa en un restaurante de comida típica de Córdoba.
- Ir a casa de Andrea, una amiga que vive en Córdoba.

Puesto que acaba de llegar al hotel, le gustaría comenzar por visitar un poco la ciudad antes de comer, es por eso que elije la opción de "Recibir información turística de Tour de arte y cultura de Córdoba".

Además, al pulsar encima de la acción seleccionada, la interfaz de su dispositivo móvil le da más información acerca de la acción. Dicha descripción, no es más que una guía turística en formato digital de la ciudad de Córdoba, como muestra la Figura 51.

Figura 51 - Detalle de la acción seleccionada

Puesto que es la primera vez que visita la ciudad de Córdoba, le gustaría que su asistente personal pudiera guiarle de forma más específica por los monumentos y sitios turísticos más conocidos de la ciudad, por eso decide pedir a su asistente la particularización de dicha acción, convirtiéndola en varias acciones que en conjunto cumplan los mismos objetivos.

Figura 52 - Nuevas acciones tras particularización

Tras la particularización de la acción "Recibir información turística de Tour de arte y cultura en Córdoba" se muestran las siguientes tres acciones que la sustituyen en la lista de acciones mostrada por la agenda:

- Recibir información turística de visita a Mezquita Catedral.
- Recibir información turística de visita a Templo Romano.
- Recibir información turística de visita a Medina Azahara.

En este caso, Paula escoge empezar la visita por la Mezquita Catedral, presionando encima de la acción y pulsando al botón ejecutar.

Figura 53 - Confirmación ejecución de una acción

Figura 54 - Valoración del proveedor seleccionado

Tras pedir la ejecución de la acción y llevar ésta a cabo, Paula recibe información en su dispositivo móvil acerca de cómo llegar hasta la Mezquita Catedral, así como información turística sobre ésta.

Una vez finalizada la ejecución de dicho servicio, Paula deberá valorar al proveedor que le ha provisto el servicio. Para ello, el dispositivo móvil muestra una pantalla de valoración (Figura 54), a partir de la cual podrá establecer un valor entre el 0 y el 5 para el citado proveedor.

Una vez valorado el servicio ofrecido por el proveedor, esta información será actualizada en el módulo de confianza interno del asistente para enriquecer el asesoramiento en futuras interacciones con éste.

Finalmente, la acción que ha sido recientemente ejecutada, se elimina de la lista.

Figura 55 - Lista de acciones tras la eliminación de la ejecutada

4. Conclusiones

El principal hito obtenido en el proyecto ha sido el desarrollo de un primer prototipo de agente asistente inteligente en el marco del proyecto mIO!, capaz de llevar a cabo el proceso de recomendación de servicios de valor para el usuario adaptado a la información de contexto.

Como consecuencia de dicho logro, se han conseguido distintas metas u objetivos que se planteaba al inicio del proyecto y se pasan a resumir a continuación.

A partir de una arquitectura de agentes dada, se ha llevado a cabo el diseño e implementación concreto del asistente inteligente, basado en la idea de la recomendación proactiva de servicios acordes al contexto del usuario. Además, se ha demostrado que éste es independiente del tipo de plataforma utilizada, pudiendo residir tanto en un dispositivo móvil, como en un dispositivo de sobremesa.

Para poder cumplir el objetivo anterior, ha sido necesario realizar un análisis de las distintas fases de las que consta el proceso de recomendación, desde la percepción del contexto del usuario, hasta la recomendación de servicios del valor para el usuario.

El agente asistente inteligente interacciona con el usuario con el fin de ofrecerle servicios de valor que satisfagan sus objetivos. Para ello, ha sido necesario analizar la interacción del agente recomendador con el usuario, dotando a éste de una interfaz que le permite poder llevar a cabo la interacción. Dicha interfaz detectada, es la agenda de acciones y puede ser implementada para cualquier tipo de dispositivos.

Puesto que el proyecto se basa en la idea de movilidad, se ha diseñado un prototipo adaptado para dispositivos móviles, donde reside el agente asistente inteligente, así como una interfaz de usuario para la interacción con éste. En este caso, se ha realizado sobre la plataforma móvil *Android*.

Con todo ello, se ha creado una implementación de referencia a partir de una arquitectura de agentes inteligentes, que da solución al problema de la recomendación de servicios adaptados a las necesidades de los usuarios.

Además, se ha presentado un caso de estudio para validar el prototipo desarrollado.

Por otra parte, indicar que este proyecto se inició en Junio de 2009 en el contexto de una beca de Investigación en el grupo de Inteligencia Artificial de la Universidad Rey Juan Carlos y fue finalizado en Julio de 2010 aproximadamente, durante este tiempo han existido diferentes etapas en la realización de éste. La estimación de esfuerzos del presente proyecto, puede ser observada en el gráfico que se muestra a continuación:

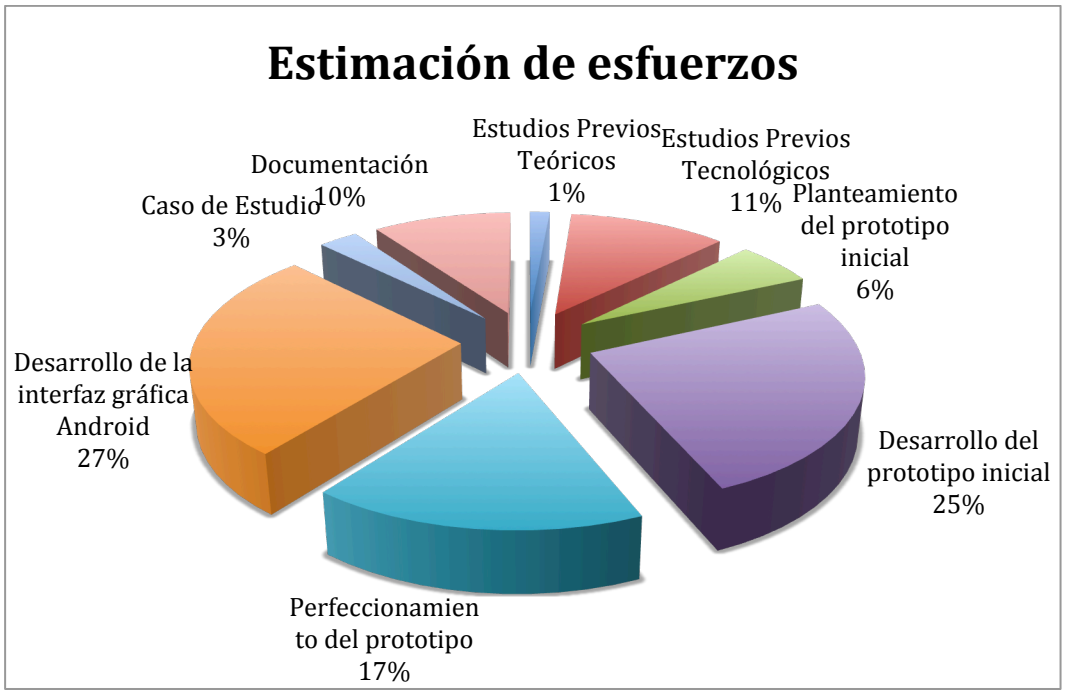

Figura 56 - Estimación de esfuerzos

Como se puede observar, un alto porcentaje del tiempo se ha dedicado a la realización por una parte del estudio previo tecnológico y por otra parte, al desarrollo en sí de la interfaz gráfica del agente en la plataforma Android. Como se ha comentado anteriormente, este proyecto comenzó a desarrollarse en el año 2009, por lo que había tecnologías (como es el caso de Android) que eran muy recientes y por tanto, estaban poco documentadas. Es por esto que fue necesario dedicar un esfuerzo extraordinario en la parte de estudio previo tecnológico, así como el desarrollo en sí mismo de la interfaz del agente para el dispositivo móvil.

Por último, y desde un punto de vista más personal, quisiera destacar los aspectos positivos que me ha aportado la realización de este proyecto. Por una parte, un acercamiento a multitud de tecnologías desconocidas para mí, así como una primera aproximación al desarrollo de proyectos en un ámbito empresarial más que académico. Así mismo, por otra parte, formar parte de un grupo de investigación, me ha permitido adquirir ciertos hábitos y aprender nuevas técnicas de trabajo que me serán de gran utilidad para mi futuro profesional: afrontar de forma madura problemas desconocidos, la necesidad constante de documentar, el trabajo en grupo, y sobre todo, el aprendizaje autodidacta sobre temas a priori poco documentados.

4.1. Líneas Futuras

Así mismo, pese a la consecución de los objetivos anteriormente mencionados, existen ciertas líneas de investigación sobre las que sería interesante trabajar.

- **Reputación**

  Para la determinación del mejor proveedor, en el presente prototipo tan sólo se ha tenido en cuenta la información almacenada internamente y asociada al usuario recomendado. El tener en cuenta sólo la opinión personal o interna, tiene sus limitaciones, por lo que una línea de trabajo podría ser la inclusión de un módulo de reputación. Esto es, introducir el concepto de red social pudiendo así consultar no sólo la opinión personal del usuario, sino también la de otros asistentes inteligentes.

- **Personalización**

  El término personalización hace referencia a la adaptación a los gustos y necesidades del usuario en cuanto a la ordenación de las acciones que son mostradas en la agenda. De forma que, teniendo en cuenta las preferencias del usuario, se mostrarían las acciones en la agenda en orden de importancia o impacto para éste.

- **Mejoras sobre el prototipo actual**

  Por una parte, dotar de robustez al prototipo. Controlando la falta de conectividad no sólo al inicio, sino durante toda la ejecución de la aplicación.

  Por otra parte, dado que el agente asistente inteligente mIO! esta ideado para la recomendación del usuario, permitir a éste la personalización de la interfaz de usuario acorde a sus preferencias.

5. Bibliografía

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web. *Scientific American , 284*, 34-43.

Billhardt, H., Hermoso, R., & Centeno, R. (2007). Trust-based Service Provider Selection in Open Environments. *SAC'07.* Korea.

Booch, G., Rumbaugh, J., & Jacobson, I. (1999). *El Lenguaje Unificado de Modelado, UML.* Addison Wesley.

Cebrián, H., Higes, A., Hermoso, R., & Billhardt, H. (2010). A trust-based provider recommender for mobile devices in semantic environments. *DCAI.* Valencia.

CDTI. (s.f.). *Cenit mIO!* Obtenido de http://www.cenitmio.es

Coppola, P. (2005). MoBe: Context-aware mobile applications on mobile devices for mobile users. *Workshop on exploiting context histories in smart environments.* Munich.

Dignum, F. y. (1998). Intentional Agents and Goal Formation. *Springer Verlag , Intelligent Agents IV*, 231-244.

Fernandez A., B. A. (2010). *Abstract architecture for service discovery.*

Fielding, R. (2000). *Architectural Styles and the Design of Network-based Software Architecture*. Obtenido de https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Fielding, R., & Taylor, R. (2002). ACM Transactions on Internet Technology. En *Principled Design of the Modern Web Architecture* (Vol. 2, págs. 115-150).

Google. (2011). *Nexus One*. Obtenido de http://www.google.com/phone/detail/nexus-one

Google. (s.f.). *Samsung Galaxy S*. Recuperado el 2011, de http://www.google.com/phone/detail/samsung-showcase-a-galaxy-s-phone

Gosling, J., Joy, B., Steele, G., & Bracha, G. (2005). *The Java (TM) Language Specification.* Addison-Wesley Professional.

Inc, G. (s.f.). *Android.com*. Obtenido de http://www.android.com

iSoco, & URJC. (2010). *Descripción de mecanismos de localización y recomendación de servicios basados en contexto y en las preferencias de los usuarios.* Madrid.

iSoco, Telefonica I+D. (2009). *Diseño de la infraestructura de gestión de contexto.*

HTC. (s.f.). *HTC Hero*. Obtenido de http://www.htc.com/es/product/hero/overview.html

HTC. (s.f.). *HTC Shift*. Obtenido de http://www.htc.com/es/product.aspx?id=14914

Martínez, A. (2011). *Directorios de servicio y modelos de confianza para la recomendación de servicios.*

mIO!, P. (2008). *E1.2.1 Informe de requisitos de gestión de contexto en un entorno móvil.*

Penela, V. (2010). Gestión semántica de contexto en entorno móviles. Madrid, España.

Pressman, R. (1998). *Ingeniería del Software. Un enfoque práctico.* McGraw-Hill.

Restlet. (2006). *RESTful web framework for Java*. Obtenido de http://www.restlet.org/

Saugar, S. (2010). Características de la Arquitectura del Entorno Restlet. Móstoles.

Samsung. (s.f.). *Samsung i7500 Galaxy*. Obtenido de http://www.samsungmobile.es/moviles/samsung-i7500-galaxy

URJC, G. (s.f.). *Wiki del proyecto mIO!* Obtenido de http://www.ia.urjc.es/trac/mIO/wiki

W3C. (s.f.). *OWL-S: Semantic Markup for Web Services*. Obtenido de http://www.w3.org/Submission/OWL-S

Wooldridge, M. y. (1995). *Intelligent agents: theory and practice.*

xStream. (2007). Obtenido de https://xstream.codehaus.org/

Anexo

Características técnicas de los dispositivos móviles

Para llevar a cabo las pruebas físicas de ejecución del prototipo se ha hecho uso de varios dispositivos móviles físicos, cuya descripción se detalla a continuación:

HTC Shift

Este dispositivo móvil de tamaño medio de capacidad de computación limitada, así como una interfaz adaptada al uso táctil por parte del usuario.

- Procesador Intel® A110, 800Mhz
- Sistema operativo Windows Vista
- Memoria RAM de 1GB
- Disco duro de 60GB
- Peso de 800 gramos (con batería)
- Pantalla táctil TFT-LCD de 7 pulgadas, 800x480 pixeles
- Conectividad HSDPA, UMTS, GSM, GPRS, EDGE y IEEE 802.11b/g

Este dispositivo se ha utilizado para la ejecución de la versión Swing del prototipo.

Dispositivos móviles Android

A continuación, se detallan los dispositivos Android específicos en los que se ha implantado el agente inteligente, incluyendo sus características más relevantes:

Samsung i7500 Galaxy

- Sistema operativo Android 1.5 (Cupcake)
- Procesador Qualcomm® MSM7200A™, 528 MHz
- Memoria RAM 192MB
- Pantalla AMOLED 3, 2'' HVGA (320x480 pixeles)
- Conectividad mediante HSDPA,UMTS,GPRS, EDGE y IEEE 802.11b/g
- A-GPS incorporado

HTC Hero

- Sistema operativo Android 1.5 (Cupcake)
- Procesador Qualcomm® MSM7200A™, 528 MHz
- Memoria RAM 256MB
- Pantalla TFT-LCD 3, 2'' HVGA (320x480 pixeles)
- Conectividad mediante HSDPA,UMTS,GPRS, EDGE y IEEE 802.11b/g
- A-GPS incorporado

Samsung Galaxy S

- Sistema operativo Android 2.1
- Procesador ARM® Cortex A8, 1 GHz
- Memoria RAM 512MB
- Pantalla Super AMOLED 4'' HVGA (480x800 pixeles)
- Conectividad mediante HSDPA,UMTS,GPRS, EDGE y IEEE 802.11b/g
- A-GPS incorporado

Nexus One

- Sistema operativo Android 2.3.3
- Procesador Qualcomm Snapdragon QSD 8250, 1 GHz
- Memoria RAM 512MB
- Pantalla AMOLED 3.7'' WVGA
- Conectividad mediante HSDPA,UMTS,GPRS, EDGE y IEEE 802.11b/g
- A-GPS incorporado