# ISMED: Intelligent Semantic Middleware for Embedded Devices

Informe científico anual 2008


## RESUMEN

Este informe contiene la memoria científica correspondiente al trabajo realizado durante 2008 para el proyecto de investigación ISMED. En tal proyecto cooperan la Universidad de Deusto (UD) y Mondragón Goi Eskola Politeknikoa (MGEP).

## HISTORIAL DE CAMBIOS

| Versión | Descripción | Autor | Fecha | Comentarios |
|---|---|---|---|---|
| V0.1 | Creación de las partes relativas al modelado y coordinación semántica | Aitor Gómez y Jorge Bastida | 12/11/2008 | |
| V0.2 | Creación de las partes relativas al aprendizaje | Asier Aztiria | 15/11/2008 | |
| V0.3 | Creación de las partes relativas al descubrimiento | Aitor Urbieta | 15/11/2008 | |
| V0.4 | Revisión de las partes relativas al modelado y coordinación semántica | Diego Lz. de Ipiña | 30/11/2008 | |
| V0.5 | Integración de las distintas contribuciones de UD y MGEP | Aitor Gómez y Jorge Bastida | 2/12/2008 | |
| V0.6 | Revisión del documento | Diego Lz. de Ipiña | 8/12/2008 | |
| V0.7 | Revisión del documento | Aitor Gómez, Jorge Bastida y Diego Lz. de Ipiña | 9/12/2008 | |

## TABLA DE CONTENIDOS

# 1 ANÁLISIS CRÍTICO DEL ESTADO DEL ARTE

## 1.1 Modelado y coordinación semántica

### 1.1.1 Introducción

Este documento analiza el estado del arte en modelado semántico y coordinación basada en modelos semánticos para dispositivos empotrados y móviles. Como resultado de este análisis se ha generado el diseño del módulo de modelado y coordinación semántica que constituirá el core de la plataforma ISMED en la sección 2.1.

### 1.1.2 El paradigma Triple Space computing

“Triple Space Computing” es un paradigma que trata de usar conocimiento expresado mediante semántica bajo el paradigma de “Space based computing” para lograr que distintas entidades se comuniquen mediante la publicación y consulta de su conocimiento en una memoria distribuida.

#### 1.1.2.1 Conceptos previos

Para comprender el “Triple Space Computing” es fundamental conocer de dónde proviene, por lo que en los siguientes epígrafes se detallarán los paradigmas en los que se fundamenta.

##### 1.1.2.1.1 Web semántica

La web semántica se basa en añadir información adicional (describiendo el contenido, significado y la relación de los datos) a la World Wide Web, para que sea posible evaluar dicha información de forma automática por parte de las máquinas. De esta forma, se mejorará Internet ampliando la interoperabilidad entre los sistemas informáticos, reduciendo la mediación de los operadores humanos.

Entre los conceptos a destacar dentro de la web semántica, se pueden enumerar los siguientes:

- RDF. Es un modelo de datos para expresar los recursos y las relaciones que se puedan establecer entre ellos. Aporta una semántica básica para este modelo de datos que puede representarse mediante XML.

- OWL. Añade más vocabulario para describir propiedades y clases: tales como relaciones entre clases (p.ej. disyunción), cardinalidad (por ejemplo "únicamente uno"), igualdad, tipologías de propiedades más complejas, caracterización de propiedades (por ejemplo simetría) o clases enumeradas.

- OWL-S. Es un lenguaje que proporciona un conjunto de construcciones para representar las relaciones presentes en una ontología, de forma que dichas ontologías puedan ser fácilmente procesables por máquinas.

- WSMO. El proyecto WSMO es una ontología para la descripción de servicios web semánticos, que no sólo pretende crear la especificación, sino que, también crea la arquitectura y facilita un conjunto de herramientas para soportar la especificación.

#### 1.1.2.1.2 Tuple spaces

Tuple spaces es una implementación del paradigma de memoria compartida para computación distribuida con el que se logra una gran escalabilidad. Este enfoque hace que se elimine la complejidad inherente del procesado de los mensajes que de otra forma tendrían que intercambiar los procesos, haciendo que los procesos se comuniquen de forma sencilla y desacoplada escribiendo, eliminando y leyendo tuplas en un espacio global persistente visible por todos ellos.

De esta forma, por un lado existirán unas entidades denominadas productores que envían sus datos en forma de tuplas (grupos de campos ordenados) a una memoria distribuida (repositorio de tuplas) a la que se puede acceder de forma concurrente desde otros productores o consumidores de tuplas. Y por otro lado, los denominados consumidores pueden consultar o coger la información existente en el espacio en base a un determinado patrón de consulta.

Así, las entidades que siguen el paradigma explicado, se comunican y coordinan compartiendo su estado en un repositorio común, siguiendo el patrón conocido como Master-Worker [1].

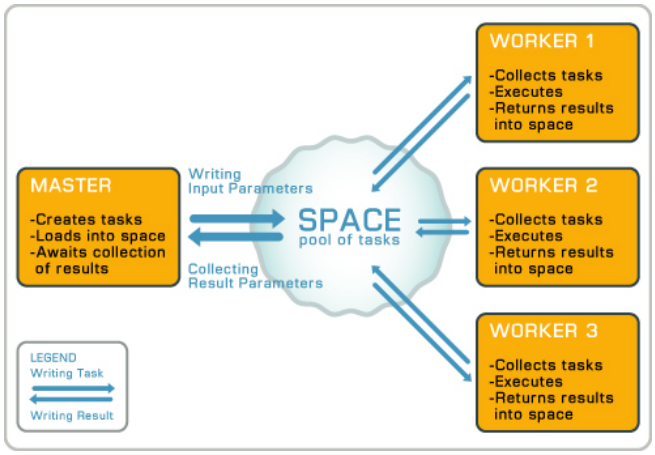


**Ilustración 1. Esquema del patrón Master-Worker.**

En él, un maestro deja unidades de trabajo en un espacio. Los trabajadores leen dicho espacio, procesan el trabajo y escriben de nuevo en dicho espacio los resultados del trabajo una vez completado el mismo. Finalmente el maestro consulta los resultados de los trabajos que han realizados los trabajadores. En un entorno típico, habría varios espacios, varios maestros y muchos trabajadores.

Añadiendo una capa de coordinación, este paradigma se puede convertir en un paradigma de comunicación entre procesos que no tienen por qué estar en el mismo sistema.

La principal **ventaja** de dicho paradigma consiste en que logra procesos completamente desacoplados en tres dimensiones distintas referencia, tiempo y espacio.

- *Referencia*: los procesos que se comunican entre sí, no necesitan saber explícitamente de su existencia mutua. No necesitan establecer un canal comunicación entre sí.

- *Tiempo*: La comunicación puede ser completamente asíncrona porque el repositorio de tuplas garantiza el almacenamiento de los datos mediante los que se comunican las entidades.

- *Espacio*: los procesos pueden ejecutarse en entornos computacionales completamente distintos dado que todos ellos podrán acceder al mismo repositorio de tuplas.

Con dicho desacoplamiento se logran avances de diseño para definir aplicaciones reusables, distribuidas, heterogéneas y de cambios rápidos [2].

Además, mediante tuple spaces se puede lograr la exclusión mutua entre productores y consumidores de la información de forma sencilla si cada vez que se accede a un dato en el repositorio se elimina del mismo y se vuelve a depositar una vez se finalice su uso.

El paradigma de “Tuple spaces” surgió de forma teórica a raíz del lenguaje de programación paralela Linda, desarrollado en la Universidad de Yale, pero hoy en día existen distintas implementaciones de dicho paradigma en distintos lenguajes, entre los que cabe destacar la realizada para el lenguaje Java denominada JavaSpaces.

#### 1.1.2.1.3 Servicios web

De acuerdo a la definición de la W3C, "un servicio web es un sistema de software identificado por una URI, cuyos interfaces públicos y vínculos se definen y describen mediante XML. Otros sistemas de software pueden descubrir su definición, y estos pueden interactuar con dicho servicio web de la forma indicada en su definición, usando mensajes basados en XML transportados mediante los protocolos de Internet."

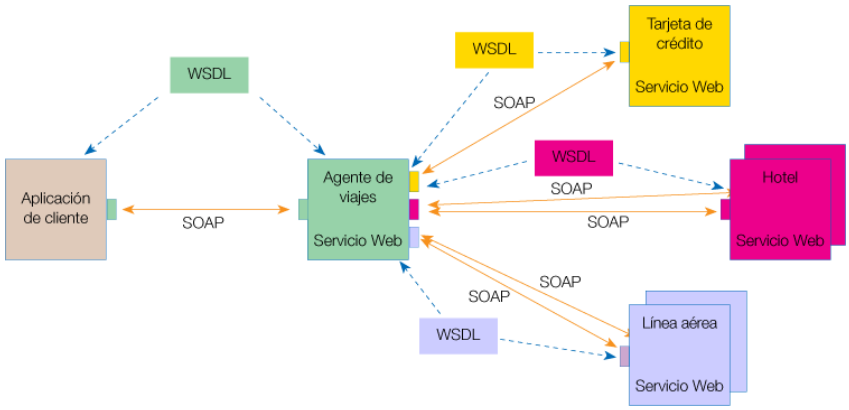


**Ilustración 2. Ejemplo de uso de servicios web y protocolos que usa**

“Dichos servicios proporcionan mecanismos de comunicación estándares entre diferentes aplicaciones, que interactúan entre sí para presentar información dinámica al usuario. Para proporcionar interoperabilidad y extensibilidad entre estas aplicaciones, y que al mismo tiempo sea posible su combinación para realizar operaciones complejas, es necesaria una arquitectura de referencia estándar”. [3]

#### 1.1.2.1.4 Servicios web semánticos

Los servicios web semánticos [4] son una nueva infraestructura para los Servicios Web. Mientras que en los servicios web tradicionales las interfaces del servicio se definen sólo a nivel sintáctico, los servicios web semánticos proveen y extienden la descripción formal de sus interfaces mediante la notación de web semántica.

Así, la aproximación de servicios web semánticos permite la integración de la Web Semántica y los Servicios Web como tecnologías complementarias. Esta nueva infraestructura permite realizar buscadores basados en Web Semántica para Servicios Web, pudiendo realizar búsquedas basadas en restricciones complejas.

**Servicios Web (Web de aplicaciones) + Web Semántica (Web de datos) = Servicios Web Semánticos (Web de datos y aplicaciones)**

De está forma, los servicios semánticos permiten usar la web semántica de tal forma que mediante las ontologías se describan los servicios para que las máquinas puedan procesarlos con la mínima intervención humana.

No obstante, la actual infraestructura de registro de servicios web impide a las tecnologías de servicios web semánticos lograr plenamente sus objetivos porque el almacenamiento de los datos semánticos queda restringido a un entorno cerrado. Por eso, el usuario necesita saber explícitamente la localización de los registros al igual que el modo de acceso a los mismos. Así es cómo actualmente pese a que los servicios web se construyen y despliegan de forma independiente, habitualmente se descubren usando un registro centralizado [5].

Esta aproximación no permite mucha escalabilidad en cuanto aumentan el número de consultas y descripciones de servicios web, mermando claramente la flexibilidad de un sistema que use registros para especificar los servicios que ofrece.

Además, la comunicación mediante dichos servicios sigue el paradigma de intercambio de mensajes de forma que la comunicación entre el proveedor y quien pide los datos, se lleva a cabo sin estado y de forma fugaz y síncrona.

El paradigma de Triplespace se hace especialmente útil en este momento, dado que, como se verá posteriormente, permite mejorar los servicios webs tradicionales haciendo que adopten una forma flexible, asíncrona y semánticamente mejorada.

#### 1.1.2.2 Triple space computing

Triple Space Computing es un nuevo paradigma de comunicación y coordinación entre máquinas que trata de proveer de una infraestructura semántica de comunicación que posibilite la interacción entre las mismas. Es una extensión del paradigma tuple-spaces a la que en esencia se le han añadido todas las funcionalidades provistas por la web semántica (RDF).

En palabras de los creadores de Tripcom, una de las primeras implementaciones en desarrollo de este paradigma, “Triple Space se convertirá en la web para las máquinas igual que la Web basada en HTML se convirtió en la Web para los humanos”.

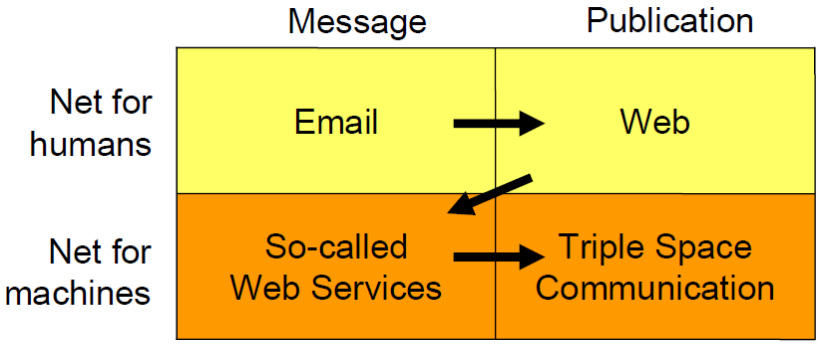


**Ilustración 3. La red vista desde el punto de vista de los seres humanos y de las máquinas.**

De esta forma, igual que los servidores web publican páginas web que leen los seres humanos, los servidores Triple Space publican datos interpretables por las máquinas. Los proveedores y consumidores podrían publicar y consumir tripletas mediante una infraestructura globalmente accesible.

Así, diversos servidores Triple Space podrían estar en diferentes máquinas y cada nodo en un proceso de comunicación podría elegir su espacio preferido de forma análoga a la Web actual.

Las ventajas inherentes de dicho mecanismo son que los proveedores de datos podrían publicar en cualquier momento (autonomía en el tiempo), independientemente de su mecanismo de almacenamiento interno (autonomía de localización), sin necesidad de conocer a sus potenciales lectores (autonomía de referencia) e independientemente del esquema de datos utilizado internamente por el repositorio (autonomía de esquema) [6].

Así, se desea pasar de un entorno en el que las máquinas se comunican directamente a través de mensajes que intercambian, al uso de un intermediario que facilite el intercambio de mensajes simplificando el esquema y ofreciendo una interfaz simple.

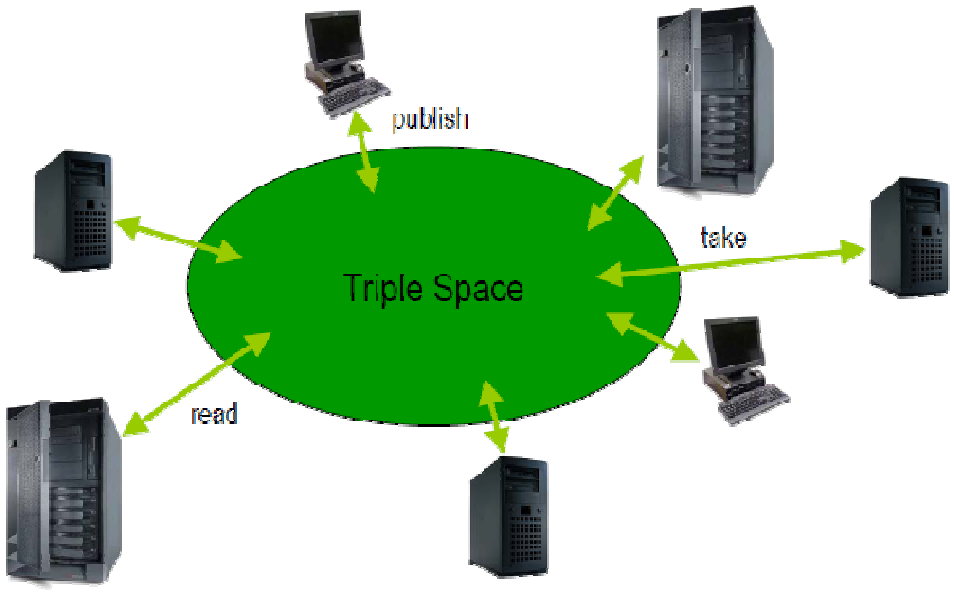


**Ilustración 4. Esquema del mecanismo de comunicación en triple space.**

De esta forma, se permite a las maquinas comunicarse de forma asíncrona, dotando a las aplicaciones que hagan uso de Triple Space de mayor autonomía.

Resumiendo, se podrían citar las siguientes ventajas que aporta TSC [6]:

- *Comportamiento web*. El paradigma Web es bien conocido y por tanto su uso es sencillo de entender. Permite configurar una comunicación más eficiente entre los procesos que siguen el paradigma.

- *Datos semánticos*. La infraestructura TSC hace que los datos de comunicación estén semánticamente definidos, por lo que dichos datos serán entendibles por las máquinas.

- *Comunicación asíncrona*. Permite que el productor y consumidor de datos mantengan autonomía entre si, leyendo o escribiendo sólo acorde a sus necesidades e independientemente el uno del otro.

- *Reducción de la incertidumbre, retraso y complejidad*.

- La incertidumbre en la comunicación se reduce dado que los datos se pueden escribir persistentemente (así ni siquiera hace falta que el TSC devuelva un mensaje de error, porque éste se podrá consultar más tarde en dicho espacio). No existen más preocupaciones por las interrupciones de la red.

- El retraso en la comunicación se elimina dado que el lector puede leer cuando le sea necesario y el productor puede escribir siempre que lo necesite.

- Mínimo impacto en el lector o productor de información. No existe impacto salvo por el acceso al Triple space mediante el protocolo de transferencia de Triple space, que es tan ligero como el propio protocolo HTTP. Además, no existe la necesidad de esperarse mutuamente para poder realizar la comunicación de forma síncrona.

#### 1.1.2.3 Proyectos relacionados

A continuación se detallan diversos proyectos actuales que tratan de desarrollar sistemas bajo el paradigma de Triple Space Computing.

##### 1.1.2.3.1 Triple Space Computing (TSC)

El objetivo del proyecto TSC es desarrollar una plataforma genérica que sirva de prototipo para entornos Triple Space Computing, posibilitando la comunicación y coordinación de la web semántica y de los servicios web semánticos.

Así, extiende el paradigma de “Tuple Space Computing” realizando un middleware mejorado con semántica, escalable y que presta especial atención a los servicios web semánticos.

TSC permite a aplicaciones Java crear y acceder a “Triple Spaces” para leer y escribir grafos RDF y realizar todas las operaciones definidas en el modelo conceptual de Triple Space Computing.

La aplicación se compone de tres bloques principales [7]:

- *Corso (Coordinated Shared Objects Space)*, usado para la capa de coordinación del Triple Space (TS) kernel, distribuye los espacios TSC entre múltiples TS kernels.

  CORSO es un middleware escrito en lenguaje C de memoria compartida para sistemas distribuidos heterogéneos, que permite solucionar los problemas de las aplicaciones distribuidas (compartir una memoria local de cada cliente con otras en un espacio): replicación, migración de datos, información de acceso, transacciones y tolerancia a fallos.

- *YARS (Yet Another RDF Repository)*, es el repositorio semántico que usa la capa de acceso a datos del TS kernel, responsable de buscar grafos en base a determinados patrones de búsqueda. YARS es un servicio web que debe desplegarse en un servidor web Tomcat, en cada ordenador que ejecute TS kernel (idealmente los clientes ligeros podrían usar un repositorio de clientes más pesados).

  YARS permite agregar información, recuperarla o eliminarla mediante simples consultas HTTP. Los resultados se devuelven en formato NTriples y las consultas se pueden realizar de forma más expresiva que usando simples patrones con la forma “<sujeto,predicado,objeto>”, mediante el lenguaje de consultas N3QL.

- *Librería Java TS kernel*. Implementa el modelo TSC y los acopla con Corso y YARS, implementando la capa de coordinación y acceso a datos.

  Así, para mantener la capa de coordinación y la de acceso a datos sincronizados, se implementan dos eventos: GraphListEvent (que se activa cada vez que un grafo cambia en un determinado espacio) y GraphTakeEvent (que se activa cuando cambia un objeto y se marca como eliminable).

  De esta forma, cada grafo escrito en un espacio que es conocido por el kernel local tiene que sincronizarse con el repositorio YARS local usando Corso cuando se den esos eventos.

Ente los principales inconvenientes del TSC se pueden enumerar los siguientes:

- La robustez del TS kernel no tolera una desconexión temporal de Corso. Si

esto ocurre, se debe crear una nueva instancia del TS Kernel.

- Cómo Corso se usa en modo persistente, almacenando espacios de forma confiable, actualmente la capa de acceso a datos sólo se usa para consultas mediante patrones de búsqueda. Para realizar este tipo de consulta puede ser suficiente una representación en memoria o incluso un índice de los grafos.

  De esa forma, la capa de acceso a datos podría funcionar en la misma máquina virtual de Java que la aplicación cliente, para lo que habría que portar YARS (lo cual no debería suponer un esfuerzo dado que YARS está implementado en Java.

  Idealmente también Corso, responsable de distribuir espacios en la red, podría ejecutarse en la misma máquina virtual de Java que la aplicación cliente. Desgraciadamente, portar Corso supondría un esfuerzo demasiado grande dado que está implementado en C.

- Latencia en el acceso a la capa de acceso a datos, dado que cada operación sobre el repositorio se traduce en una serie de consultas HTTP a YARS.

- El diseño de las estructuras de datos que usa Corso hace que a menos objetos Corso, se necesiten menos consultas a Corso, pero que se obtengan más datos desde Corso (que al final habrán de ser procesados por el cliente de la máquina virtual de Java). Por el contrario, más objetos acarrean más consultas a Corso, pero menor proceso a llevar a cabo en la máquina virtual Java del cliente. Haciendo que las estructuras de Corso fueran finamente granulares (fine-granular) se reducirían los conflictos transaccionales en caso de operaciones concurrentes sobre el mismo espacio.

- Cada componente puede proveer acceso a múltiples clientes a un solo TS kernel, manteniendo replicas de Triple Spaces. El TS kernel puede ser usado por un único cliente (cliente pesado), lo que quiere decir que cada cliente necesitará una instalación partícular de Corso y YARS, lo que puede suponer una sobrecarga innecesaria si múltiples clientes se sitúan en una misma máquina o área local.

- La licencia de Corso es una licencia privativa, que no permite por tanto acceder

y modificar el código.

Está última desventaja hizo que el proyecto de TSC como tal fuese reconcebido y modificado sustancialmente, dando paso al proyecto tsc++ que se explica en el siguiente epígrafe.

#### 1.1.2.3.2 tsc++

tsc++ [8] es una implementación del paradigma TSC, que hace uso de P2P y tecnologías de web semántica como RDF(tripletas) y SPARQL. Este projecto trata de desarrollar el paradigma Triple Space Computing como un framework middleware para la comunicación y coordinación de web semántica y servicios web semánticos. Usa Sesame u Owlim para almacenamiento persistente de las tripletas RDF, mientras para la comunicación de red y coordinación se realiza a traves del framework JXTA.

tsc++ está basado en el proyecto TSC explicado anteriormente y dados los problemas de licencias causados por el software propietario CORSO en la capa de coordinación se sustituyó éste por Jxta.

Pese a que en sus primeras versiones el proyecto tsc++ heredó el repositorio semántico usado por TSC YARS, éste fue sustituido paulatinamente por Sesame y Owlim, hasta que en la versión 1.1 (la última publicada hasta la fecha) fue suprimido por completo.

Tsc++ puede requerir gran cantidad de memoria y ancho de banda, por lo que la ejecución de multiples instancias del kernel en una sola máquina es dificultosa.

#### 1.1.2.3.3 Triple Space Communication (TripCom)

TripCom [9] es un proyecto que a día de hoy aún no se ha finalizado, por lo que no existe una versión final del prototipo que funcione según lo diseñado. Pese a ello, existen diversos aspectos del diseño que resultan interesantes de analizar.

El proyecto define un kernel que almacena la información semántica y ofrece un API sobre la que los clientes realizan sus operaciones. Desde ese punto de vista del kernel, pueden existir distintos kernels que alberguen distintos datos semánticos, y se comuniquen entre si de una forma completamente descentralizada, haciendo uso de

"P2P" [10].

Por otro lado, los clientes utilizan una aproximación centralizada en la que ellos consultan a un servidor (o kernel) mediante al que acceden al Triple Space.

Así se utilizan dos estrategias básicas para realizar las operaciones de la API. La primera está basada en **DNS**, dado que cada triplespace se identifica por una URL. La segunda se basa en un sistema de almacenamiento de índices distribuidos.

En el enfoque de TripCom los datos no se mueven, sino que se mueven índices a esos datos, que permiten que otros kernels puedan descubrir dónde se encuentran esos datos. Para llevar a cabo esta indexación (explicada en el epígrafe 1.1.2.4.3), se usa **P-Grid**, que permite una gran escalabilidad.

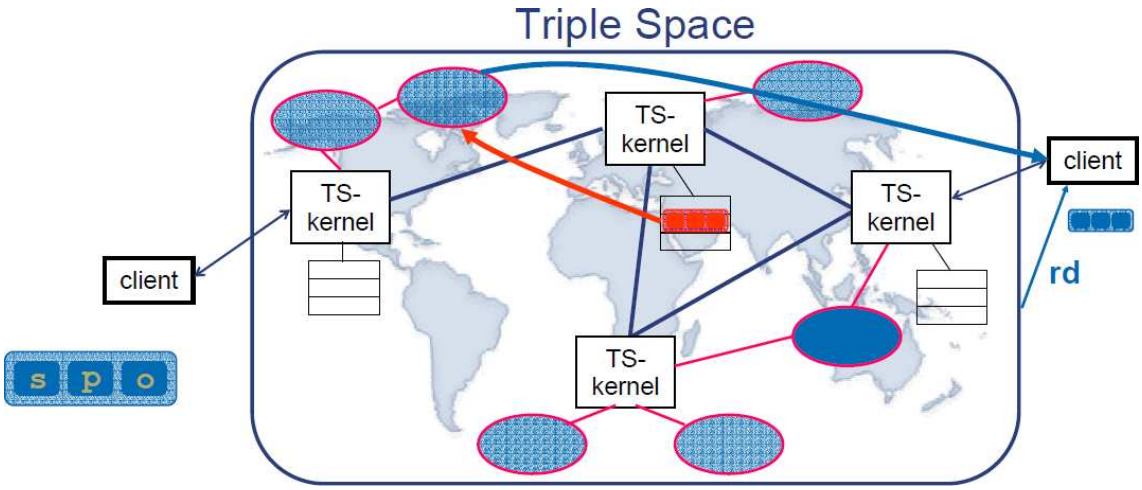


**Ilustración 5. Esquema de funcionamiento de las referencias en TripCom.**

En la Ilustración 5 se muestra un esquema en el que el cliente de la izquierda escribe una tripleta en el espacio de un TS-kernel al que identifica mediante su URL. Al insertar dicho conocimiento en el espacio, también se guardan unas referencias en otros TS-kernel (para saber más de dicho proceso, se puede consultar el epígrafe 1.1.2.4.3).

Por otro lado, cuándo el cliente de la derecha realiza una consulta a un TS-kernel cualquiera, este localiza a otro TS-kernel que conoce en la URL del espacio en que se encuentra dicha referencia (de nuevo, ese proceso se explica en el epígrafe 1.1.2.4.3). Así, finalmente se localiza y devuelve la tripleta que corresponde con el patrón consultado mirando en el espacio adecuado.

El kernel de TripCom está altamente desacoplado, y para comprenderlo mejor es necesario explicar cada uno de los distintos componentes que lo forman [11]:

- *Triple Store Adapter*: es el componente que recibe las consultas y lee los datos almacenados en el repositorio pertinente.

  En cuanto a la capa de acceso a datos se refiere, TripCom (en conjunto con el proyecto TAO[1]) se caracteriza por desarrollar su propio sistema de almacenamiento escalable, de alto rendimiento e independiente del lenguaje. Dicha plataforma se denomina ORDI (Ontology Representation and Data Integration) y será descrita en posteriores epígrafes.

- *Security Manager*: es el componente que asegura que las operaciones no violan la política de seguridad especificada.

- *Mediation Manager*: ejerce de intermediario entre las tripletas o grafos entrantes y salientes.

- *Metadata Manager*: ejerce de razonador, validando la instancia de la ontología con su esquema y razona para encontrar información de los datos requeridos por los usuarios (estadísticas de acceso para tripletas, grafos, etc.)

- *Query Processor*: descompone las consultas en partes que pueden ser satisfechas por el repositorio local y en partes que deben ser enviadas a otros kernels.

- *Transaction Manager*: gestiona las transacciones locales y ejerce de participante en las transacciones distribuidas, en las que puede ser el coordinador de la misma si el kernel al que pertenece inició dicha transacción.

- *Distribution Manager*: es el responsable de realizar un Triple Space distribuido, reenviando consultas y peticiones a kernels que probablemente sean capaces de satisfacerlas parcialmente y de reenviar las peticiones de escritura a los kernels apropiados.

- *Management API*: se usa por los administradores para inicializar las estructuras

[1] http://www.tao-project.eu

de datos y coordinar componentes del kernel. Es quien inicia todo el proceso del kernel.

- *Triple Space API*: ofrece una serie de operaciones a los clientes del TSC.
- *Web Service Invocation*: ofrece un API con operaciones que permiten transportar datos de un servicio web desde quien los pide a quien los provee.
- *Web Service Discovery*: es responsable de buscar descripciones de servicios web que concuerden con los objetivos de los requesters.
- *Web Service Registry*: transforma descripciones de servicios web, en grafos RDF.

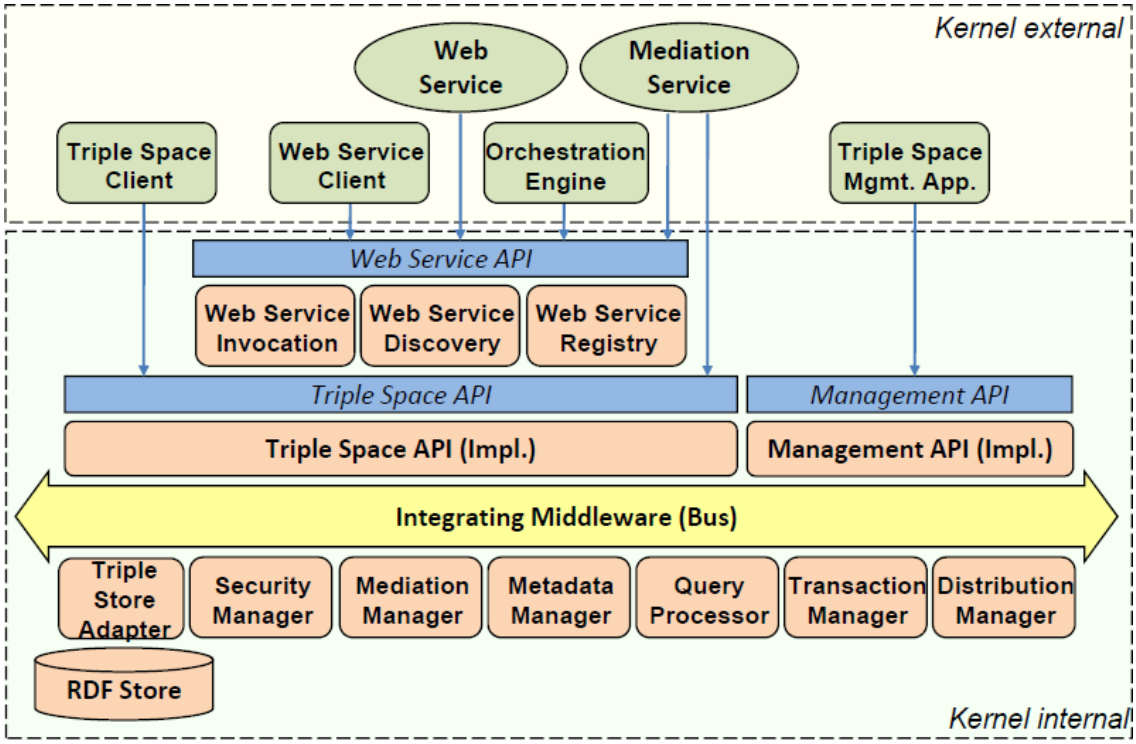


**Ilustración 6. Arquitectura de Triple Space Communication**

Además, existen una serie de entidades externas al kernel:

- *Triple Space Client*: para crear clientes que acceden al Triple Space usando directamente su API.
- *Service*: servicios webs que pueden ser usados por los clientes.

- *Mediation Service*: proveen funcionalidad para mapear entre conceptos y ontologías.

- *Orchestration Engine*: permite la composición de servicios.

Para que la integración de los componentes se realice de forma robusta, tolerante a fallos y con posibilidad de balanceo de carga, también se pensó en usar un middleware de espacio compartido. Pese a que en un principio se planteo usar un sistema de XVSM (eXtensible Virtual Shared Memory) para realizar dicho intercambio de datos entre los componentes del kernel, finalmente se adopto la solución de Javaspaces [10] (usando la implementación Blitz).

#### 1.1.2.3.4 Comparativa

Cómo se ha visto, existen dos plataformas principales, ambas todavía en proceso de desarrollo, que siguen el paradigma Triple Spaces: TSC y TripCom. Ambas constan de múltiples capas, de entre las que destacan por su importancia la capa de coordinación y la de acceso a datos. Como se resume en la siguiente tabla, ambas implementaciones hacen uso de distintas tecnologías en las distintas capas.

**Tabla 1: Comparativa de las características generales de los distintos proyectos que usan *Triple Space Computing*.**

| | TSC | Tsc++ | TripCom |
|---|---|---|---|
| **Capa de coordinación** | Corso | Jxta | DNS y P-Grid |
| **Capa de acceso a datos** | YARS | Sesame<br>Owlim | ORDI<br>• SGBD<br>• YARS<br>• OWLIM<br>• Sesame |
| **Licencia** | Licencia libre salvo en la capa de coordinación | GNU Lesser General Public License (LGPL) | GNU Lesser General Public License (LGPL) |

El proyecto TSC como tal no se finalizó, dado que como se ha explicado, se paso a

desarrollar el tsc++ que no es sino una versión mejorada del mismo. Pese ello, el TSC planteaba dos dificultades principales que se han podido ver en la tabla:

- La capa de coordinación estaba implementada con Corso, que no sólo estaba desarrollada en C y compilada para Windows, sino que tenía una licencia privativa que no permitía realizar modificaciones sobre el mismo con el fin de **portarla** a un hipotético dispositivo móvil que ejecutase programas en C.

- La capa de acceso a datos hacía uso de YARS, que es un servicio web que debe desplegarse en un **servidor web con contenedor de servlets** como **Tomcat**. La posibilidad de hacer que el dispositivo móvil pudiese ejecutar una versión reducida de Tomcat resulta utópica, pero para paliarlo se barajó la posibilidad de que clientes ligeros (por ejemplo dispositivos móviles) accediesen al servicio web YARS de clientes más pesados.

Analizando las características de Tsc++ y TripCom, podemos apreciar cómo ambos utilizan repositorios semánticos semejantes, y que su principal diferencia reside en la forma en la que se lleva a cabo la coordinación entre distintos Kernels.

Así mientras Tsc++ opta por usar Jxta para hacer que los Kernels se comuniquen entre si y se pidan los datos necesarios, TripCom utiliza DNS para localizar los kernels y P-Grid para llevar a cabo una indexación de los nombres de los kernels que tienen determinados datos en cada uno de los kernels.

Si se tiene en cuenta que DNS es una base de datos distribuida y jerárquica que almacena información asociada, y que se tarda un tiempo considerable en actualizar dicha información a lo largo de internet, no parece que sea un estructura lo suficientemente flexible como para adaptarse al entorno cambiante que forman los dispositivos empotrados y móviles que se consideran en ISMED. Además, no se ha encontrado referencia a ninguna versión equivalente de P-Grid para dispositivos móviles.

Por otro lado, aunque más sencilla, la estructura del Tsc++ no podría ser usada en entornos móviles, dado que hace uso de la versión para Java SE de Jxta. Además, como más tarde se verá, dado que no existen repositorios semánticos para dispositivos móviles, la estructura de tsc++ no será fácilmente expandible en entornos móviles.

En la siguiente tabla se observan otras características de ambos proyectos. Mientras tsc++ hace uso de complejas y pesadas librerías, que sólo funcionan en Java SE, en el caso de TripCom se le suma el inconveniente de que además tiene una gran complejidad interna ya que está compuesto por siete componentes que deben ser arrancados manualmente (pudiendo hacerse uso de un gestor, pero no parece en cualquier caso que sea sencillo de iniciar de un modo más automatizado).

**Tabla 2. Comparativa de las principales implementaciones del paradigma del *Triple Space Computing*.**

| | Tsc++ | TripCom2 |
|---|---|---|
| **Plataforma** | JDK 1.5 o superior | JDK 1.5 o superior |
| **Estándares soportados** | RDF<br>SPARQL (parcialmente mediante plantillas)<br>TURTLE<br>RDFXML<br>N3 (para carga de ficheros) | RDF<br>SPARQL (en el nivel extendido) |

[2] Dado que no existe una versión suficientemente madura de dicho proyecto, no se pueden obtener muchos datos necesarios para que la comparativa fuera más completa.

| | Tsc++ | TripCom2 |
|---|---|---|
| **Librerías necesarias** | Jxta (14MB)<br>jxta.jar<br>javax.servlet.jar<br>org.mortbay.jetty.jar<br>bcprov-jdk14.jar<br>log4j (349KB)<br>log4j.jar<br>Yet Another Swing Library (23KB)<br>slf4j-api-1.4.3.jar<br>slf4j-log4j12-1.4.3.jar<br>Sesame (1,35MB)<br>openrdf-sesame-2.1.3-onejar.jar<br>OWLIM (200KB)<br>trree-3.0.beta7.jar<br>owlim-3.0.beta7.jar | P-Grid<br>Blitz |
| **Tamaño de las librerias** | 16MB | ¿? |
| **Tamaño del núcleo** | 200KB | 105MB |
| **Componentes en los que se divide el núcleo** | 1 | 7 |

Todo ello nos hace pensar que pese a que TripCom es muy completo, probablemente se deba optar por usar un enfoque más semejante a tsc++. Pese a todo, tsc++ no parece ser suficientemente sencillo (necesita de mucha memoria y ancho de banda, como se reconoce en su propia guía de usuario [8]) para ejecutarlo en los tipos de dispositivos objeto dentro de ISMED.

*1.1.2.3.4.1 Primitivas*

Por lo general, las primitivas sobre triples spaces se podrían agrupar en las siguientes categorías:

- *Write*: almacena información en el repositorio RDF.
- *Read*: realiza una búsqueda en base a un patrón en todos los grafos devolviendo los grafos que contienen tripletas que cumplen dicho patrón.
- *Take*: es análogo a read, pero tras leer las tripletas, las elimina.
- *Query*: realiza una búsqueda en base a un patrón en todos los grafos devolviendo solamente las tripletas que lo cumplen.
- *Subscribe*: para solucionar el problema de la diseminación de información se suelen usar el paradigma publish/subscribe. Así los suscriptores pueden expresar su interés por determinados datos suscribiéndose.
- *Advertise*: permiten publicar cierta información cumpliendo con el paradigma previamente mencionado.

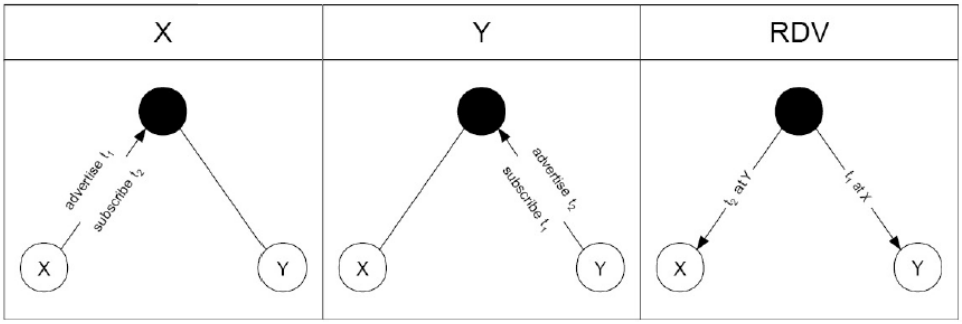


**Ilustración 7. Modelo de anuncio suscripción.**

- *Transacción*: permite realizar transacciones sobre el triple space.

El proyecto TripCom divide su API en distintos niveles dependiendo de la expresividad que dichas operaciones permiten. Pese a que a priori podría considerarse que más expresividad es recomendable, también hay que tener en cuenta que ésta limita la escalabilidad del sistema.

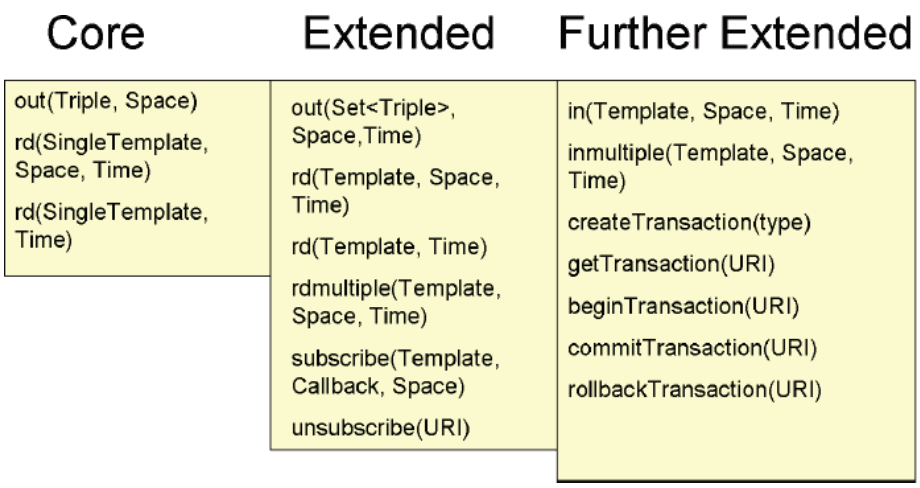


**Ilustración 8. División en distintos niveles de expresividad del API de TripCom.**

Las primitivas usadas por tsc++ [8] y TripCom [12] se pueden resumir en la siguiente tabla.

**Tabla 3. Primitivas usadas por las principales proyectos de *Triple Space Computing*.**

| | Tsc++ | TripCom |
|---|---|---|
| **Write** | URI write(URI space, Set<ITriple> triples)<br>Hace que las tripletas se escriban en el espacio localmente. En ese momento otros kernels podrán acceder a dichas tripletas remotamente. Al escribir, el conjunto de tripletas se considera un grafo que se identifica con una URI. La ventaja reside en que las mismas tripletas crearán el mismo identificador. | void out( Triple t, URL space)<br>void out( Set<Triple> t,URL space)<br>Escribe atómicamente una o varias tripletas en el espacio. |

| | Tsc++ | TripCom |
|---|---|---|
| **Read** | Set<ITriple> read(URI space, URI graph)<br>Set<ITriple> read(URI space, ITemplate template)<br>La diferencia con Query, es que sólo mira dentro de un grafo si se cumple el patrón especificado por template, y devuelve el grafo entero. | - |
| **Take** | Set<ITriple> take(URI space, URI graph)<br>Set<ITriple> take(URI space, ITemplate template)<br>Lee y elimina del repositorio. | Set<Triple> in( Template t, URL space, Time timeout)<br>Set<Set<Triple>> inmultiple( Template t, URL space, Time timeout)<br>Son operaciones paralelas a las explicadas en rd, pero se encarga de eliminar las tripletas después de leerlas. |

| | Tsc++ | TripCom |
|---|---|---|
| **Query** | Set<ITriple> query(URI space, ITemplate template)<br>Consulta tripletas en todos los grafos y devuelve las tripletas que lo cumplen.<br>Combina tripletas buscadas remotamente con las buscadas localmente.<br>Existe la posibilidad de indicarle que tan pronto como reciba los primeros resultados devuelva dichas tripletas (sin llegar a combinar nada). | Set<Triple> rd( SingleTemplate t, URL space, Time timeout)<br>Set<Triple> rd( SingleTemplate t, Time timeout)<br>Devuelve una tripleta que concuerda con el patrón facilitado (sin importar si existe más de una tripleta que cumpla el mismo). En la API extendida se pueden facilitar patrones de búsqueda más complejos (Template).<br>Cuándo no se facilita el espacio, el sistema selecciona cualquier tripleta que coincida con el patrón sin importar el espacio en el que esté (en el apartado 1.1.2.4.3 se explica dicho proceso más detalladamente), siempre que el cliente tenga permisos de lectura.<br>Set<Set<Triple>> rdmultiple( Template t, URL space, Time timeout)<br>Devuelve todas las tripletas que cumplen un patrón dado en un espacio. La diferencia con “rd” es que rdmultiple devuelve más de un resultado. |
| **Advertise** | URI advertise(URI spaceURI, ITemplate template)<br>void unadvertise(URI spaceURI, URI advertisement)<br>Hace uso de los “advertisements” que facilita Jxta. | |

| | Tsc++ | TripCom |
|---|---|---|
| **Suscribe** | URI subscribe(URI spaceURI, ITemplate template, INotificationListener listener)<br>void unsubscribe(URI spaceURI, URI subscription)<br>Dado un patron y una clase sobre la que se realizará la llamada de callback cuando dicho patrón haya cambiado, se suscribe al espacio definido por el parámetro spaceURI. | URI subscribe( Template t, Callback c, URL space)<br>void unsubscribe( URI subscription)<br>Se facilita un patrón y la clase a la que el sistema avisará en el momento en el que se encuentre una coincidencia para ese patrón. |
| **Transacciones** | - | URI createTransaction( String type)<br>boolean getTransaction( URI transactionID)<br>Se une a una transacción ya creada.<br>boolean beginTransaction(URI transactionID)<br>Todas las interacciones iniciadas por los agentes que compartan dicha transacción se realizarán “todas o ninguna”.<br>boolean commitTransaction( URI transactionID)<br>boolean rollbackTransaction( URI transactionID) |

Tanto en las operaciones “take” como “query” se puede apreciar que existe un último parámetro llamado **timeout**, que es el tiempo que el kernel deberá esperar a la resolución de la consulta antes de desbloquear al proceso que inició la consulta.

Cómo se ha podido ver en el anterior apartado, existen distintos patrones, que ofrecen distintos niveles de expresividad a la hora de realizar consultas. A continuación se

describen los mismos.

- Tsc++

    - Especifican las clausulas WHERE de las consultas SPARQL. Sustituyen el siguiente patrón por “?s, ?p, ?o”.

```
CONSTRUCT {
        ?s, ?p, ?o
} WHERE {
        GRAPH <ts://espacio>  {  ?s, ?p, ?o  }
}
```

- TripCom

    - SimpleTemplate
      Es una tripleta que puede contener variables (semejante a la usada por tsc++).

    - Template
      Es una consulta genérica. Se pueden realizar consultas más complejas, como por ejemplo, consultas SPARQL.

Así, dados los siguientes triples almacenados en el espacio “ts://www.ejemplo.org/espacio”:

```
@prefix rdf : <http:/ /www.w3 .org/1999/02/22rdfsyntaxns#> .
@prefix foaf : <http://xmlns.com/foaf/0.1/> .
<http:/ /members.sti2.at/membername> rdf:type foaf: Person .
<http://members.sti2.at/membername> foaf:name "Member Name" .
<http://members.sti2.at/membername> foaf:firstName "Member" .
<http://members.sti2.at/membername> foaf:phone <callto://membername> .
<callto://membername> rdf:type <http://skype.com> .
```

Ejemplos de consultas en tsc++:

- Query con patrón simple

```
<http://members.sti2.at/membername>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#rstName>
?o.
```

Tanto *tsc*++ cómo en *TripCom*[3] devolverían todas aquellas tripletas que cumplen el patrón:

```
<http://members.sti2.at/membername>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#rstName>
"Member".
```

Además, si en TripCom se usase la primitiva “rd” y existiese más de una tripleta que cumpliese dicho patrón, el cliente que realizase las consultas sólo obtendría una tripleta como respuesta a cada llamada “rd”, que no necesariamente tendría porque ser distinta cada vez [13].

- Query con patrón complejo

Los ejemplos de consultas complejas, se corresponderían con el funcionamiento normal de las consultas SPARQL.

Por ejemplo, en la siguiente consulta se devolverían tripletas de la forma “<nombre_persona de país>” de aquellos participantes de simposios que existen.

```
PREFIX ns: <http://example.com/any#>
CONSTRUCT {
?name ns:from ?country
```

[3] En el ejemplo, el resultado sería el mismo usando “rd” o “rdmultiple” porque sólo existe una tripleta que cumple dicho patrón.
En caso de usar la primitiva “rdmultiple” se devolverían todas las tripletas que lo cumpliesen. Si se optase por usar “rd”, se devolvería una sola tripleta por cada consulta realizada.

```
} WHERE {
?x rdf:type ns:Symposium .
?x ns:hasParticipant ?p .
?p ns:name ?name; ns:resides ?country. }
```

- Read

```
<http://members.sti2.at/membername>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#rstName>
?o.
```

En tsc++ se devuelve todo el grafo que fue escrito junto con la tripleta que cumple el patrón:

```
<http://members.sti2.at/membername>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://xmlns.com/foaf/0.1/Person>.
<http://members.sti2.at/membername>
<http://xmlns.com/foaf/0.1/name>
"Member Name".
<http://members.sti2.at/membername>
<http://xmlns.com/foaf/0.1/rstName>
"Member".
<http://members.sti2.at/membername>
<http://xmlns.com/foaf/0.1/phone>
<callto://membername>.

<callto://membername>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://skype.com>.
```

#### 1.1.2.4 Tecnologías base para implementar el paradigma Triple Space Computing

##### 1.1.2.4.1 Jena

Jena es una plataforma desarrollada por los laboratorios HP de Bristol. Este proyecto open-source provee soporte completo para inferencia en RDFS, soporte parcial para OWL y además permite que el usuario pueda crear motores de reglas personalizados.

##### 1.1.2.4.2 Jxta

Jxta [14] es una plataforma peer-to-peer open source creada por Sun Microsystems que define un conjunto de seis protocolos basados en XML que permiten que dispositivos conectados a la red (no necesariamente tiene porque ser la misma red local) intercambien mensajes entre si.

Los seis protocolos son asíncronos y están basados en un modelo de consulta/respuesta. A continuación se detallan dichos protocolos:

- *Peer Discovery Protocol (PDP)*. Se usa por peers para publicitar sus propios recursos (peers, grupos, pipes o servicios) y descubrir los de otros peers.

- *Peer Information Protocol (PIP)*. Se usa para obtener el estado de otros peers.

- *Peer Resolver Prototol (PRP)*. Permite a los peers enviar consultas genéricas y recibir respuestas. Este protocolo permite definir e intercambiar información arbitraria que se necesitará.

- *Pipe Binding Protocol (PBP)*. Establece un canal de comunicación virtual (un pipe), uniendo dos o más extremos de la comunicación.

- *Endpoint Routing Protocol (ERP)*. Se usa para buscar rutas y puertos de destino en otros peers. La información de encaminamiento tiene una secuencia ordenada de identificadores, que se puede usar para enviar mensajes al destino.

- *Rendezvous Protocol (RVP)*. Se usan para la resolución de recursos, propagar

mensajes, publicar recursos locales y organizarse con otros peers en caso de ser rendezvous (aquellos peers que ayudan a otros con la propagación de mensajes).

Existen diversas implementaciones de Jxta en distintos lenguajes de programación y plataformas (c, symbian, java me, etc.) que permiten su funcionamiento en un amplio rango de dispositivos (ordenadores, teléfonos móviles, PDAs), logrando que se comuniquen de forma descentralizada.

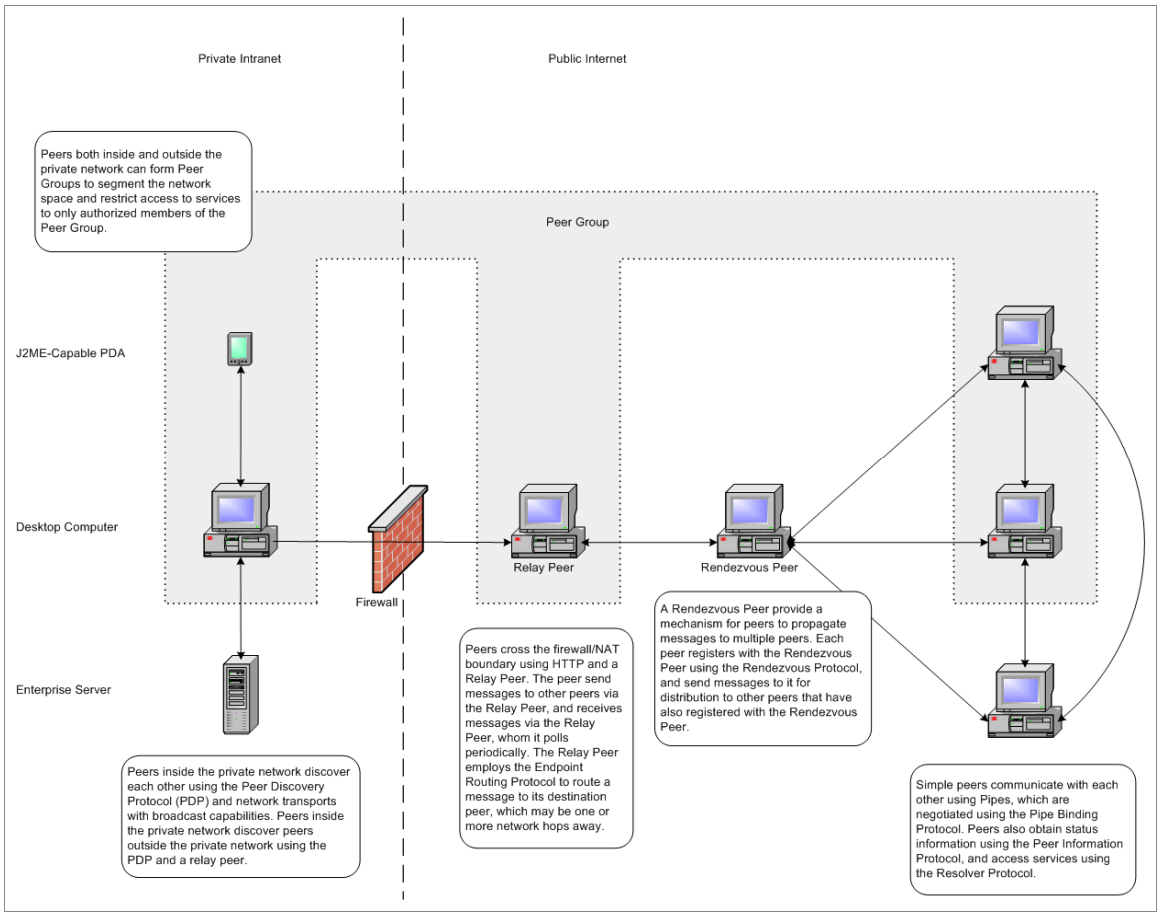


**Ilustración 9. Esquema del funcionamiento de los distintos tipos de elementos que pueden componer una red de Jxta**

#### 1.1.2.4.3 P-Grid

P-Grid [15] es una plataforma para la gestión de la información de forma distribuida que permite una compartición de ficheros sencilla. Esta plataforma provee de una estructura descentralizada de P2P que no requiere coordinación central ni conocimiento mutuo previo de las entidades que la forman.

Entre las características más destacables de P-Grid cabe destacar su completa descentralización, su capacidad de organización propia, el balanceo de carga descentralizado que permite, las funcionalidades para la gestión de datos que incluye, la gestión de IPs dinámicas e identidades que implementa y la búsqueda eficiente que posibilita.

P-Grid se usa en el proyecto TripCom para llevar a cabo una indexación de los datos almacenados en cada nodo contenedor de conocimiento. En concreto, sirve para implementar las tablas de hash distribuido (Distributed Hash Tables, DHT).

Las DHT son una clase de sistemas distribuidos descentralizados que reparten la propiedad de un conjunto de claves entre los nodos que participan en una red, y son capaces de encaminar de forma eficiente mensajes al dueño de una clave determinada. Cada nodo es análogo a una celda de una tabla hash.

Así, al almacenar una tripleta en el TripleSpace de TripCom identificado por una URL, se crean los índices que apuntan a la URL donde finalmente se guardará la tripleta en distintos *host P-Grid* indicados por la función hash aplicada a los distintos elementos de la tripleta.

Más tarde, en caso de realizar una consulta sin indicar la URL del espacio en el que se desea buscar, se calculará la función hash sobre los parámetros que se especifiquen en dicho patrón, para encontrar el número del host *P-Grid* en cuya tabla de referencias se encuentra una referencia a la URL que contiene los datos pertinentes para realizar la consulta final.

Para comprenderlo mejor, se usará el ejemplo de [13].

- Dada la operación “out(<a,p,j>,URL)”:
    - Se calcula el hash de a: h(a)=78
        - Se almacena la URL en la tabla de referencias del host *P-Grid* 78.
    - Se calcula el hash de a: h(p)=45
        - Se almacena la URL en la tabla de referencias del host *P-Grid*

45.

- Se calcula el hash de a: h(j)=67
    - Se almacena la URL en la tabla de referencias del host *P-Grid* 67.
- Se almacena la tripleta <a,p,j> en la URL

Así, tras dicha lectura, las tablas de referencias almacenadas en los distintos hosts *P-Grid* comentados, podría ser:

| **Host P-Grid 78** | | | |
|---|---|---|---|
| **s** | **p** | **o** | **TS URL** |
| a | p | j | tsa.com |
| b | q | m | tsc.com |

| **Host P-Grid 45** | | | |
|---|---|---|---|
| **s** | **p** | **o** | **TS URL** |
| a | p | j | tsa.com |
| c | r | n | tsb.edu |
| a | s | p | tsc.com |

| **Host P-Grid 67** | | | |
|---|---|---|---|
| **s** | **p** | **o** | **TS URL** |
| a | p | j | tsa.com |

- Si a continuación se realiza la operación rd(<a,?p,?o>).
    - Se realiza la función hash sobre a (el resto de elementos del patrón son variables), obteniéndose: h(a)=78
    - Luego, se obtiene del host P-grid 78, que para satisfacer la consulta se debe consultar en la URL del espacio “tsa.com”.
    - Realiza la consulta rd(<a,?p,?o>,URL)

#### 1.1.2.4.4 Javaspace

JavaSpace [16] es una implementación en Java del paradigma de “Tuple Spaces” que forma parte de la tecnología Jini, que permite almacenar objetos de forma distribuida (persistente o no). De esta forma, se pueden almacenar tanto el estado del sistema como las implementaciones de algoritmos.

La API de JavaSpace que permite la manipulación de objetos “Entry” (aquellos que se pueden almacenar en el repositorio) consta de tres primitivas básicas:

- *Write*, que ubica una nueva entrada en el espacio.

- *Read*, que devuelve una copia de un objeto que concuerde con una consulta expresada con una plantilla determinada.

- *Take* - elimina un objeto que concuerde con una determinada plantilla y lo devuelve al objeto que realiza la llamada.

Las ventajas inherentes del uso de JavaSpace son la escalabilidad que se puede lograr gracias al procesamiento paralelo que posibilita y el almacenamiento confiable que permite el repositorio distribuido.

#### 1.1.2.4.5 Blitz

El proyecto Blitz [17] es una implementación *open source* de JavaSpace, que trata de facilitar el desarrollo y despliegue de dicha tecnología de forma eficiente y compatible con Jini 2.x

Entre otras características destacables se encuentran las numerosas herramientas que ofrece para facilitar su gestión, la capacidad de personalizar la forma en la que se lleva a cabo la persistencia, indexación inteligente (usa almacenamiento en disco o en memoria logrando reducir el tiempo de búsqueda al mínimo), permite ver las entradas que hay en las distintas instancias del repositorio, posibilidad de uso de forma embebida para uso local en aplicaciones y facilidad de expansión.

#### 1.1.2.4.6 Repositorios RDF

Los repositorios semánticos son sistemas similares a los sistemas gestores de bases de datos, ya que permiten almacenamiento, consultas y gestión de datos estructurados [18].

La principal diferencia entre ambos reside en el uso de ontologías como esquema semántico de las que hacen uso los repositorios semánticos y que permiten razonamiento. Además, las soluciones de almacenamiento suelen estar basadas en tecnologías de web semántica cómo RDF y SPARQL.

*1.1.2.4.6.1 Sesame*

Sesame es una plataforma RDF *open source* que permite realizar consultas (en SPARQL y SeRQL) e inferencias. Se puede desplegar sobre múltiples sistemas de almacenamiento (bases de datos relacionales, en memoria, etc.).

En la Ilustración 10 se muestra las capas que componen Sesame entre las que cabe destacar:

- *Sail (Storage And Inference Layer)*, que es un API de bajo nivel que abstrae de detalles de almacenamiento e inferencia.

- *Repository API*, que es un API de mayor nivel que ofrece métodos para subir archivos de datos, consultas, extracción y manipulación de datos.

- *HTTP Server*, que es un servlet de Java que implementa un protocolo para el acceso a los repositorios de Sesame mediante HTTP (es posible funcionar sin dicho servidor, pero en caso de querer hacerlo funcionar es necesario tener Tomcat instalado u otro contenedor de servlets).

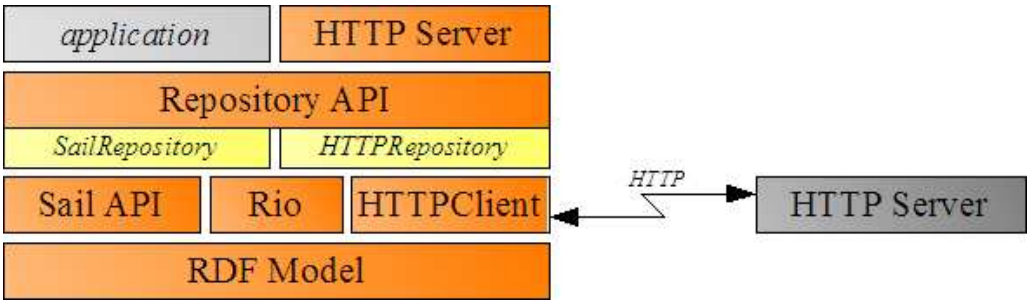


**Ilustración 10. Estructura de Sesame**

Sus características principales son:

- Su extensibilidad.

- Mínima sobrecarga de memoria (sólo la operación de escritura es directa)

- Consumo de memoria. Consultas y lecturas sobre información almacenada en disco duro.

- Crea una carpeta en la que almacena toda la información que gestiona.

Por otro lado, su mayor problema reside en el bajo rendimiento que logra al escribir grandes colecciones de triples.

*1.1.2.4.6.2 Ontology Representation and Data Integration*

ORDI [19] es un sistema de almacenamiento escalable, de alto rendimiento e independiente del lenguaje. Dicha plataforma, que es una extensión de Sesame, permite que se puedan acoplar distintos métodos que aseguren la persistencia de los datos y la capacidad de inferencia, mediante diversos adaptadores desarrollados a tal fin (por ejemplo, SesameAdapter, YARSAdapter o RDBMSAdapter) [20].

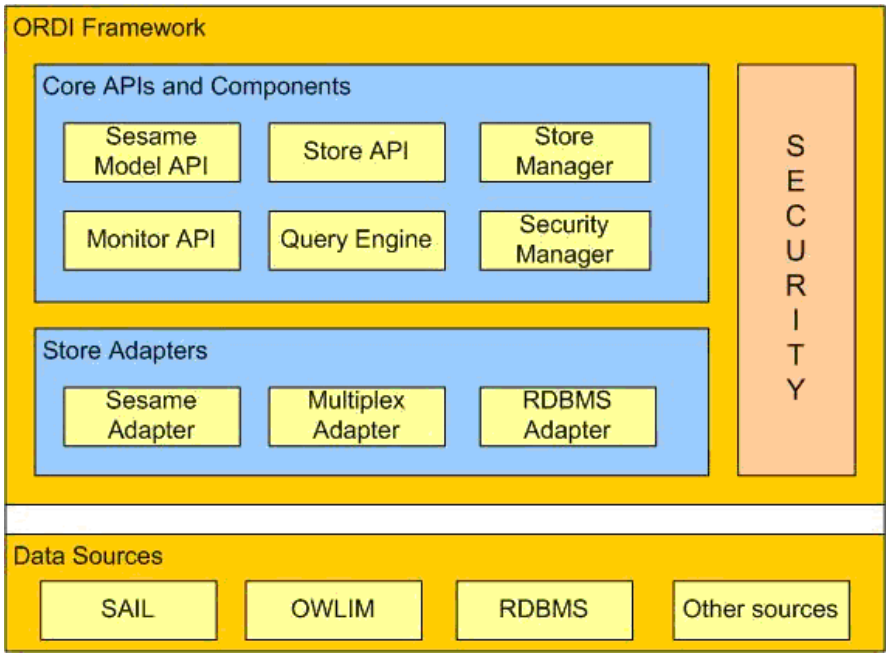


**Ilustración 11. Estructura de la Plataforma ORDI.**

ORDI facilita:

- Integración posible de diferentes estructuras de datos incluidos sistemas de gestión de base de datos relacionales.

- Retrocompatiblidad con especificaciones RDF existentes y lenguaje de consultas SPARQL.

- Operaciones transaccionales sobre un modelo (en desarrollo).

- Proceso eficiente y almacenamiento de meta-datos o información contextual.
- Sentencias de grupo para gestionar grupos con los siguientes propósitos:
    - Definición de permisos de acceso y firmas.
    - Gestión de grupos de sentencias que corresponden a construcciones simples en lenguajes de alto nivel.
    - Gestión y rastreo (tracking) de transacciones.
- Sencilla gestión de datos desde distintas fuentes dentro de un mismo repositorio (o entorno computacional). Aquellos casos en el que se tienen datos importados de diferentes archivos (por ejemplo, diversas ontologías).

*1.1.2.4.6.3 OWLIM*

OWLIM [21] es un repositorio semántico que implementa la interfaz SAIL de Sesame, permitiendo la gestión, integración y análisis de datos heterogéneos, combinando ligeras capacidades de razonamiento.

Se podría hacer la analogía de ver OWLIM como una base de datos RDF con alto rendimiento en el razonamiento. Existen dos tipos de soluciones (SwiftOWLIM y BigOWLIM) idénticas en API, sintaxis y lenguajes de consulta, pero que difieren en la su rendimiento.

Así, mientras SwiftOWLIM es bueno para experimentos y para cantidades de datos medias por su extrema rapidez en la carga de datos, BigOWLIM está diseñado para manejar grandes volúmenes de datos y consultas intensivas, de forma que posee una mayor escalabilidad con requisitos de memoria menores.

A continuación se ofrece una tabla comparativa con ambas soluciones:

**Tabla 4. Comparativa de las dos versiones de OWLIM.**

| | SwiftOWLIM | BigOWLIM |
|---|---|---|
| **Escala (Millones de sentencias explícitas)** | 10 MSt, usando 1,6GB RAM<br>100 MSt, usando 16GB RAM | 130 MSt, usando 1,6GB<br>1068 MSt, usando 8GB |
| **Velocidad de procesamiento (carga+inferencia+almacenamiento)** | 30 KSt/s en un portátil<br>200 KSt/s en un servidor | 5 KSt/s en un portátil<br>60 KSt/s en un servidor |
| **Optimización de consultas** | No | Sí |
| **Persistencia** | Back-up en N-Triples | Ficheros de datos binarios e índices |
| **Licencia y disponibilidad** | Open-source bajo LGPL; hace uso de SwiftTRREE que es gratuito pero no open-source. | Comercial.<br>Las copias de evaluación o para investigación se facilitan de forma gratuita |

OWLIM usa como lenguajes de consulta SPARQL, SeRQL, RQL y RDQL, y permite importar y exportar datos a XML, N3 y N-triples a través de Sesame.

La principal desventaja de N-Triples es su gran necesidad de memoria [8].

*1.1.2.4.6.4 MemCached*

Memcached [22] es un sistema de memoria caché distribuida de propósito general, originalmente desarrollado por Danga Interactive para el proyecto LiveJournal. Su utilización se centra en la reducción del uso de la Base de Datos de grandes proyectos, implementando un sistema que se encarga de cachear accesos previos en una memoria distribuida.

Memcached utiliza una tabla distribuida de par clave/valor, a lo largo de toda la memoria configurada para tal uso. En caso de necesitar mayor cantidad de memoria de la configurada, se utilizan algoritmos LRU para eliminar la información que más tiempo lleva en la tabla.

Memcached permite almacenar en la memoria distribuida todo tipo de objetos y acceder a ellos desde cualquiera de las múltiples API existentes y se distribuye bajo licencia libre de Danga Interactive, Inc.[4]

En la actualidad existen implementaciones de memcached en los siguientes lenguajes:

- Perl
- PHP
- Python
- Ruby
- Java
- C#
- C
- LUA

Aunque su uso se ha centrado en sitios web con gran cantidad de visitas y requisitos técnicos, su implementación de memoria distribuida puede usarse con propósito general.

Tal y como indican en la web existen numerosos proyectos de gran envergadura que utilizan dicha solución, entre los que cabe destacar: Youtube, LiveJournal, Slashdot, Wikipedia, SourceForge, Facebook, Digg, Twitter y NYTimes.com [5].

*1.1.2.4.6.5 YARS*

YARS (Yet Another RDF Repository) un repositorio semántico que se ejecuta a modo de servicio web en un servidor web Tomcat. YARS permite agregar información, recuperarla o eliminarla mediante simples consultas HTTP. Los resultados se devuelven en formato NTriples y las consultas se pueden realizar de forma más

[4] Memcached License. http://code.sixapart.com/svn/memcached/trunk/server/LICENSE

[5] Who's using Memcached. http://www.danga.com/memcached/users.bml

expresiva que usando simples patrones con la forma “<sujeto,predicado,objeto>”, mediante el lenguaje de consultas N3QL.

*1.1.2.4.6.6 Otros*

Pese a que no han sido utilizados en proyectos relacionados con el paradigma Triplespace, además de los analizados, existen otros repositorios semánticos que se listan a continuación:

- 3Store [23], es un almacen de tripletas RDF, escrito en C que usa MySQL y BerkeleyDB, diseñado para un manejo eficiente de bases de conocimiento RDF. Este repositorio permite el acceso a los datos RDF vía RDQL o SPARQL usando http, una interfaz de comandos o un API de C.

- RDFPeers [24], que es un repositorio de tripletas distribuido, donde los datos se guardan a través de un largo número de nodos, de forma que si uno de dichos nodos falla, la información aún es accesible dado que está replicada en otros nodos de la red. Pese a ello RDFPeers no es un repositorio semántico al uso, dado que los repositorios guardan fracciones de todos los datos, guardando índices del sujeto, predicado y objeto de las tripletas que almacenan.

- Kowari, que tiene soporte nativo de RDF y un lenguaje de consultas tipo SQL.

- TAP, que es un sistema que permite almacenamiento distribuido y búsqueda semántica.

1.1.2.4.7 Comparativa y adaptación a entornos móviles y empotrados

Sesame y Owlim [25] son librerías que pueden integrarse perfectamente en aplicaciones completamente hechas en java como tsc++. Pese a ello, y tal y como se puede ver a continuación, tienen requisitos computacionales excesivamente elevados:

**Tabla 5. Comparativa entre repositorios semánticos.**

| | OWLIM[6] | Sesame | ORDI |
|---|---|---|---|
| **Última versión** | 3.0 beta5 | 2.2.1 | 0.5 alpha2 |
| **Lenguaje en el que están programadas** | JDK 1.4.2 | JDK 1.5 | JDK 1.5 |
| **Consumo de memoria** | | | |
| **Escala (Millones de sentencias explícitas)** | 10 MSt, usando 1,6GB RAM<br><br>100 MSt, usando 16GB RAM | | |
| **Velocidad de procesamiento (carga+inferencia+almacenamiento)** | 30 KSt/s en un portátil<br>200 KSt/s en un servidor | | |
| **Librerías de las que hace uso** | trree-3.0.beta7.jar<br>owlim-3.0.beta7.jar | openrdf-sesame-2.2.1-onejar.jar | |
| **Espacio que ocupan las librerías** | 200KB | 1,55MB | |
| **Motor de razonamiento** | TRREE 2.9.1 | | TRREEAdapter |
| **Estándares** | RDF, OWL DLP, y OWL Horst. | RDF, RDF Schema, OWL | Especificaciones RDF |

[6] Para la comparación se ha utilizado SwiftOwlim dado que de las dos versiones es el que menores requisitos computacionales requiere y el que por tanto mejor se podría ajustar a los requisitos computacionales limitados de los dispositivos móviles.

| | OWLIM[6] | Sesame | ORDI |
|---|---|---|---|
| **Lenguajes de consulta** | SeRQL, RQL y RDQL (SPARQL desde la v3.0b7) | SeRQL y SPARQL | SPARQL |
| **Persistencia** | Back-up en N-Triples | - Basado en memoria<br>- Basado en disco (native store)<br>- Usando RDBMS<br>- Usando un repositorio HTTP | Múltiples opciones: tantos como tipos de adaptadores existan implementados. |
| **Licencia y disponibilidad** | Open-source bajo LGPL; hace uso de SwiftTRREE que es gratuito pero no open-source. | BSD | |

### 1.1.3 Dispositivos Móviles

El objetivo principal del módulo de modelado y coordinación semántica es adaptar y trasladar el paradigma TSC analizado en el epígrafe anterior para que funcione en dispositivos empotrados y móviles con capacidad de cómputo, almacenamiento y memoria reducida.

#### 1.1.3.1 Java ME

Java ME, anteriormente llamada J2ME, es una plataforma formada por un subconjunto

de la plataforma Java que tiene como propósito proveer un conjunto de APIs para el desarrollo de software en dispositivos con capacidades limitadas. Dichos dispositivos pueden ser móviles, PDAs o Set-Top Boxes (STB) entre otros.

Esta plataforma está dividida en dos configuraciones (CLDC y CDC). Cada una de estas configuraciones puede ser extendida mediante perfiles, los cuales definen los requisitos mínimos que los dispositivos deben cumplir.

Además de las configuraciones y perfiles, esta plataforma dispone de extensiones definidas mediante JSRs. Algunos ejemplos de estas extensiones son: Bluetooth (JSR-82), WMA 1.0 (JSR-120) y Web Services (JSR-172) y FileConnection (JSR-75).

En este punto se van a explicar las diferentes configuraciones y perfiles de Java ME. También se realizara una pequeña introducción a alguna de sus extensiones.

#### 1.1.3.1.1 CLDC

CLDC (Connected Limited Device Configuration) está formado únicamente por el subconjunto mínimo de la librería de clases de Java necesario para que una máquina virtual pueda ejecutarse. Por ello es la configuración más adecuada para dispositivos de capacidades muy limitadas como teléfonos móviles y PDAs. Esta configuración esta especificada en los JSR 30 (CLDC 1.0) y 139 (CLDC 1.1).

Como anteriormente se ha comentado, las configuraciones pueden ser ampliadas mediante perfiles. Los perfiles soportados por CLDC son MIDP e IMP.

##### *1.1.3.1.1.1 MIDP*

MIDP (Mobile Information Device Profile) es una especificación de perfil destinada a dispositivos embebidos tales como teléfonos móviles y PDAs. Este perfil está definido en los JSR 37 (MIDP 1.0) y 118 (MIDP 2.0). Actualmente se está definiendo el JSR 271 (MIDP 3.0).

La versión 1.0 de este perfil añade clases para el manejo de operaciones de entrada/salida, diseño de interfaces de usuario y almacenamiento persistente. Además añade la clase MIDLet.

La versión 2.0 extiende la librería de interfaces de usuario añadiendo soporte para el desarrollo de juegos e incluye nuevas funcionalidades que permiten crear aplicaciones multimedia y realizar conexiones seguras.

*1.1.3.1.1.2 IMP*

IMP (Information Module Profile) es una especificación de perfil destinada a dispositivos embebidos equipados con capacidades de visualización muy limitadas. Este perfil está definido mediante los JSR 195 (IMP 1.0) y 228 (IMP-NG).

La versión 1.0 de este perfil añade clases para el manejo de operaciones de entrada/salida y almacenamiento persistente. Además añade la clase MIDLet.

La versión NG incorpora el manejo de más tipos de redes y seguridad en éstas entre otras cosas.

1.1.3.1.2 CDC

CDC (Connected Device Configuration) está formado por casi toda la librería de clases exceptuando las relacionadas con la interfaz de usuario. Debido a esto, esta configuración requiere dispositivos con más capacidades que la configuración anteriormente explicada (CLDC), STB's por ejemplo. Esta configuración está especificada en los JSR 36 (CDC 1.0).

Los perfiles soportados por esta configuración son Foundation, Personal Basis y Personal.

*1.1.3.1.2.1 Foundation Profile*

Este perfil está diseñado para dispositivos que requieran implementación completa de la máquina virtual de Java así como una casi completa librería de clases de Java.

*1.1.3.1.2.2 Personal Basis Profile*

Personal Basis Profile es una extensión de Foundation Profile. Este perfil añade un subconjunto ligero de AWT.

*1.1.3.1.2.3 Personal Profile*

Personal Profile es una extensión de Personal Basis Profile. Este perfil soporta AWT completamente a diferencia de Personal Basis y además añade soporte para applets.

1.1.3.1.3 Extensiones

Como anteriormente se ha comentado, Java ME puede ampliarse mediante una serie de extensiones. Las extensiones más interesantes para este proyecto son las comentadas en los siguientes puntos.

*1.1.3.1.3.1 WMA*

WMA (Wireless Messaging API) es una extensión opcional de Java ME que permite acceder independientemente de la plataforma a servicios wireless tales como el envío de SMS. Esta extensión está especificada en los JSR 120 (WMA 1.0) y 205 (WMA 2.0).

Esta extensión puede ser utilizada tanto en CDC como en CLDC.

*1.1.3.1.3.2 WSA*

WSA (Web Services API) permite la creación de clientes de Servicios Web proveyendo de un modelo de programación que cumple los estándares de Servicios Web. Esta extensión está especificada en el JSR 172.

*1.1.3.1.3.3 FileConnection*

Esta extensión, especificada en el JSR 75, opcional permite acceder al sistema de ficheros de los dispositivos y a las tarjetas de memoria montadas en el mismo. Esta extensión puede ser utilizada tanto en CDC como en CLDC.

#### 1.1.3.2 Librerías de interés para implementar Triple Space Computing

##### 1.1.3.2.1 Motores de inferencia para Sistemas Embebidos

Existen numerosos motores de inferencia que funcionan en entornos donde los recursos computacionales son altos. Desgraciadamente, prácticamente no existen motores de inferencia para Sistemas Embebidos, destacando entre ellos el desarrollado por Iñaki Vázquez en su Tesis [26]. En la tesis se describen las características del denominado MiniOwlReasoner, el cual no es un razonador de ontologías completo, sino que es un razonador limitado, pero a la vez potente para entornos de Inteligencia Ambiental. Para realizar el razonamiento el motor de inferencia filtra las ontologías para seleccionar solamente las construcciones que es capaz de manipular, las cuales son las siguientes:

- rdfs:subClassOf
- owl:sameAs
- owl:TransitiveProperty
- owl:SymmetricProperty
- owl:inverseOf

Las construcciones mostradas en la lista anterior son los predicados más empleados para crear relaciones entre los conceptos de las ontologías. El razonador MiniOwlReasoner puede procesar cualquier tipo de característica transitiva o simétrica, que pese a ser poco, intrínsecamente aportan mucha inteligencia a cualquier ontología. El autor resalta que el razonador desarrollado implementa un pequeño conjunto de OWL Lite, pero que es más complejo y potente que RDF++ de Lassila [27] y equivalente a OWL Tiny [28].

##### 1.1.3.2.2 Microjena

MicroJena [29] es una versión reducida de la API Jena que fue diseñada para permitir el diseño y la implementación de aplicaciones de web semántica en dispositivos móviles.

Microjena cuenta entre sus ventajas, el requerir un menor espacio en memoria (384 KBs frente a los 16MBs de Jena), la mayor velocidad que se alcanza (que se hace más palpable cuanto mayor sea la complejidad de los modelos) y la

retrocompatibilidad con Jena que permite desarrollar de una forma similar a dicha API ahorrando de dicha manera tiempo de adaptación para los usuarios de Jena.

Entre sus problemas más destacables, se encuentran:

- Microjena excluye mecanismos de inferencia.
    - Permite realizar consultas sobre un modelo, realizando un filtrado y devolviendo las tripletas que cumplen dicho patrón, pero sin realizar ningún razonamiento.
- No tiene motores ARQ, de forma que no permite utilizar SPARQL, ni ningún otro lenguaje de consulta sobre el modelo semántico.
- No posee una forma de almacenar los modelos semánticos con los que trabaja, ni en base de datos, ni en ningún otro tipo de repositorios.
- El único formato en el que es capaz de exportar el modelo es en N-Triples.

Los creadores de Microjena proponen como solución a la limitación relacionada con la incapacidad de Microjena para inferir, el uso de un razonador DIG, aunque sin tener la certeza de la validez de dicha solución.

Las interfaces DIG proveen de un mecanismo neutral para el acceso a la funcionalidad que ofrece un razonador de lógica descriptiva. A alto nivel, el mecanismo consiste en mensajes XML enviados al razonador por conexiones HTTP y las respuestas que éste facilita.

Ante la incapacidad para almacenar el modelo en los dispositivos móviles, en posteriores epígrafes se analizarán sistemas para realizar esta tarea de la mejor forma posible.

#### 1.1.3.2.3 JXME

JXME [30] es una implementación del protocolo Jxta para dispositivos móviles que funcionen con Java Micro Edition.

*1.1.3.2.3.1 Con proxy*

Es la versión de JXME para dispositivos móviles con Java ME CLDC. Dada la menor capacidad de cómputo de los móviles de este tipo, la versión opera con un relay que ejerce además de proxy para comunicarse con el resto de nodos de la red.

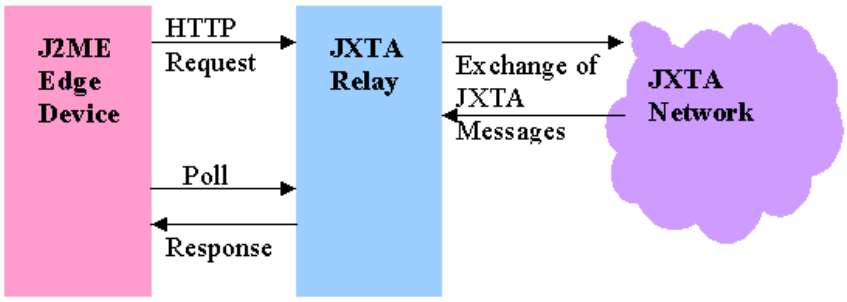


**Ilustración 12. Coordinación entre dispositivos J2ME y Relays JXTA**

Las restricciones concretas por las que es necesario usar un proxy/relay con los dispositivos móviles o empotrados son las siguientes:

- La memoria necesaria para poder parsear los mensajes XML que constituyen el protocolo Jxta.
- La memoria necesaria para cachear el estado de la red.
- Los peers Jxta escuchan a nivel de socket y datagrama para encontrar información sobre las redes, sin embargo los dispositivos móviles sólo pueden hacer uso del protocolo HTTP por defecto. Los sockets y conexiones datagrama de las que hace uso Jxta son opcionales.

Por lo tanto, dado Jxta excede los requisitos mínimos de MIDP, es necesaria la figura del relay que se encargue de realizar esas tareas que los dispositivos no son capaces de realizar.

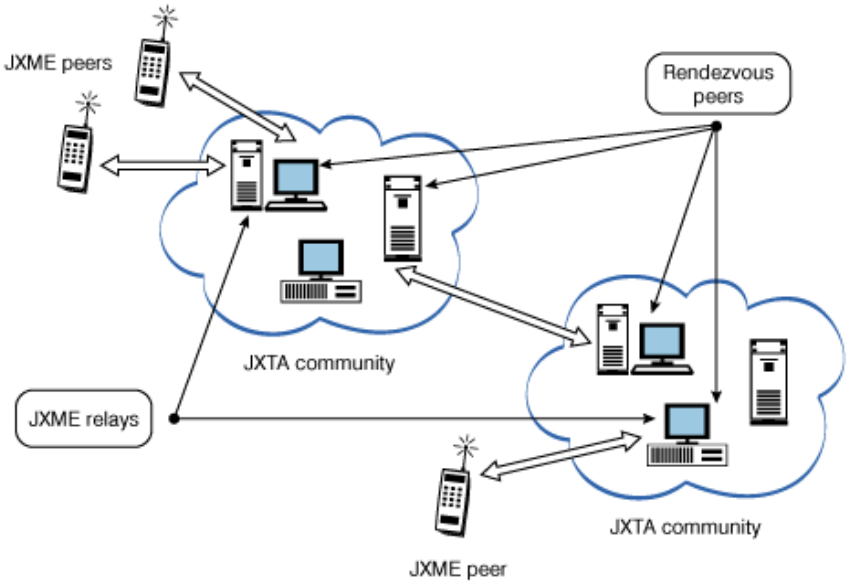


**Ilustración 13. Esquema del funcionamiento de peers JXME con el resto de peers de Jxta.**

*1.1.3.2.3.2 Sin Proxy*

Es la versión de JXME para dispositivos móviles Java ME CDC. Gracias a las mayores capacidades de los dispositivos móviles CDC, esta versión de Jxta no necesita hacer uso de proxys.

1.1.3.2.4 MobileSpaces

MobileSpaces [31] parece ser la única implementación de Javaspaces existente para dispositivos móviles. Esta librería trata de permitir a las aplicaciones para dispositivos móviles acceder a “tuple space”, de la forma más semejante posible a como se realiza desde otras plataformas.

El sistema creado se basaba en tres componentes principales:

- Javaspace como implementación del paradigma tuple space.
- La tecnología Jini, que agrupa servicios (como se ha explicado previamente JavaSpace es considerado un servicio de Jini) y facilita el acceso a los mismos.
- La arquitectura Surrogate de Jini, que permite a dispositivos limitados acceder

a los servicios de Jini.

Por su parte, esta librería está compuesta de dos elementos principales:

- Surrogate, que ejerce de proxy entre el dispositivo y JavaSpaces. Así, gestiona el acceso al espacio y envía y recibe objetos java, dado que Java ME es incapaz de ello debido a sus limitaciones.

- Manejar situaciones resultantes del funcionamiento común de los dispositivos móviles (como llamadas telefónicas, que pausarían la aplicación)

De esta forma, MobileSpaces permitía a los móviles convertirse en usuarios de JavaSpaces.

El aspecto negativo es la escasez de documentación que existe al respecto de MobileSpaces, y sobre todo, el hecho de que la implementación del sistema no parece haberse publicado en ningún lugar.

#### 1.1.3.2.5 Persistencia de datos y RDF

Cómo se ha explicado en el epígrafe 1.1.2.4.6, los repositorios semánticos no sólo sirven para almacenar modelos semánticos de forma persistente (como podrían hacer los sistemas de gestión de bases de datos), sino que además permiten realizar inferencia sobre dichos modelos.

Desgraciadamente, como se ha explicado anteriormente los motores de inferencia existentes para dispositivos embebidos son anecdóticos, por lo que es aún más complejo encontrar estos motores en conjunción con un mecanismo de persistencia. Es por ello, que no se ha tenido constancia de la existencia de ningún repositorio semántico.

Para suplir esta deficiencia, se podrían usar repositorios normales sin capacidades semánticas.

La solución más común bajo la plataforma Java Me CLDC es RecordStore, que es una base de datos de registro que se caracteriza por su gran consumo de tiempo y su capacidad para agotar la batería del dispositivo. Además, las búsquedas que se pueden realizar sobre él, son muy limitadas.

Existen numerosas abstracciones sobre RecordStore como son OpenBaseMovil, Perst Lite, Floggy, J2MEMicroDB o la capa de persistencia facilitada por la librería Java Polish. Además, existen distintos drivers que permiten acceder a servidores de bases de datos con funcionalidades no limitadas mediante una interfaz similar a JDBC, como puede ser Mimer.

Para CDC existen soluciones más sofisticadas como puede ser JavaDB (basado en apache Derby). JavaDB necesita de 256KBs de ROM y RAM, el perfil “Foundation Profile” (JSR-219) y el paquete opcional de JDBC para plataformas Java ME JSR-169.

### 1.1.4 Plataformas empotradas

A continuación se especificarán las capacidades, programabilidad, coste y capacidades de comunicación de las plataformas empotradas más relevantes susceptibles de ser usadas en el proyecto.

#### 1.1.4.1 Gumstix

Gumstix [32] es una empresa dedicada a la construcción de ordenadores miniaturizados completamente funcionales. La línea de productos de Gumstix está formada tanto por ordenadores empaquetados como por placas base sueltas.

Esta plataforma [33] está formada por una placa base, equipada con un procesador ARM, a la que se le puede añadir placas de expansión que ofrecen funcionalidades adicionales, tales como red Ethernet, wireless, lector de tarjetas, etc. Estos ordenadores funcionan con una distribución de GNU/Linux embebido llamada OpenEmbedded.

Actualmente existen las siguiente familias de placas base: Basix, Connex, Verdex, Verdex Pro y Overo Earth. Las diferencias entre todas estas placas base están especificadas en las siguientes tablas.

**Tabla 6. Comparativa de los distintos productos la línea Basix.**

| | Basix 200 | Basix 400 xm | Basix 400 xm-bt |
|---|---|---|---|
| **Procesador** | Intel XScale PXA255 a 200 MHz | Intel XScale PXA255 a 400 MHz | Intel XScale PXA255 a 400 MHz |
| **Mem. RAM** | 64 MB | 64 MB | 64 MB |
| **Mem. Flash** | 4 MB | 16 MB | 16 MB |
| **Tarjetas de memoria** | MMC | MMC | MMC |
| **Slots** | 1 x 60 pin | 1 x 60 pin | 1 x 60 pin |
| **Conexión** | | | Bluetooth |
| **Precio** | 99 $ | 129 $ | 169 $ |

**Tabla 7. Comparativa de los distintos productos la línea Connex.**

| | Connex 200 xm | Connex 400 xm | Connex 400 xm-bt |
|---|---|---|---|
| **Procesador** | Intel XScale PXA255 a 200 MHz | Intel XScale PXA255 a 400 MHz | Intel XScale PXA255 a 400 MHz |
| **Mem. RAM** | 64 MB | 64 MB | 64 MB |
| **Mem. Flash** | 16 MB | 16 MB | 16 MB |
| **Slots** | 1 x 60 pin, 1 x 92 pin | 1 x 60 pin, 1 x 92 pin | 1 x 60 pin, 1 x 92 pin |
| **Conexión** | | | Bluetooth |
| **Precio** | 109 $ | 129 $ | 169 $ |

**Tabla 8. Comparativa de los distintos productos la línea Verdex.**

| | Verdex xm4 | Verdex xm4-bt | Verdex xl6p |
|---|---|---|---|
| **Procesador** | Marvell PXA270 con XScale a 400 MHz | Marvell PXA270 con XScale a 400 MHz | Marvell PXA270 con XScale a 600 MHz |
| **Mem. RAM** | 64 MB | 64 MB | 128 MB |
| **Mem. Flash** | 16 MB | 16 MB | 32 MB |
| **Slots** | 1 x 60 pin, 1 x 120 pin, 1 x 24 pin flexible | 1 x 60 pin, 1 x 120 pin, 1 x 24 pin flexible | 1 x 60 pin, 1 x 120 pin, 1 x 24 pin flexible |
| **Conexión** | | Bluetooth | |
| **Otras características** | USB Host signals, CCD Camera signals | USB Host signals, CCD Camera signals | USB Host signals, CCD Camera signals |
| **Precio** | 129 $ | 159 $ | 169 $ |

**Tabla 9. Comparativa de los distintos productos la línea Verdex pro.**

| | Verdex pro xm4 | Verdex pro xm4-bt | Verdex pro xl6p |
|---|---|---|---|
| **Procesador** | Marvell PXA270 con XScale a 400 MHz | Marvell PXA270 con XScale a 400 MHz | Marvell PXA270 con XScale a 600 MHz |
| **Mem. RAM** | 64 MB | 64 MB | 128 MB |
| **Mem. Flash** | 16 MB | 16 MB | 32 MB |
| **Tarjetas de memoria** | Micro SD | Micro SD | Micro SD |
| **Slots** | 1 x 60 pin, 1 x 80 pin, 1 x 24 pin conector flexible y plano | 1 x 60 pin, 1 x 80 pin, 1 x 24 pin conector flexible y plano | 1 x 60 pin, 1 x 80 pin, 1 x 24 pin conector flexible y plano |
| **Conexión** | | Bluetooth | |

| | Verdex pro xm4 | Verdex pro xm4-bt | Verdex pro xl6p |
|---|---|---|---|
| **Otras características** | USB Host signals, CCD Camera signals | USB Host signals, CCD Camera signals | USB Host signals, CCD Camera signals |
| **Precio** | 129 $ | 159 $ | 169 $ |

**Tabla 10. Características de Overo Earth.**

| | Overo Earth |
|---|---|
| **Procesador** | Procesador de aplicaciones OMAP 3503 con CPU ARM Cortex-A8 a 600 MHz |
| **Mem. RAM** | 256 MB DDR de bajo consumo |
| **Mem. Flash** | 256 MB NAND |
| **Tarjetas de memoria** | Micro SD |
| **Slots** | 2 x 70 pin, 1 x 27 pin conector flexible y plano |
| **Otras características** | Soporte para placas base “OMAP 35x-based Overo”, bus I2C, 6 x líneas PWM, 6 x A/D, 1-wire, UART, entrada de cámara, entrada de micrófono, salida de auriculares, entrada batería de reserva, USB OTG signal, USB HS host |
| **Precio** | 149 $ |

1.1.4.1.1 Programabilidad

Al disponer de un sistema operativo GNU/Linux, se pueden desarrollar aplicaciones para dispositivos Gumstix en diferentes lenguajes y plataformas.

Los lenguajes más interesantes en los que se pueden desarrollar aplicaciones para esta plataforma son los siguientes:

- C/C++: Se pueden desarrollar todo tipo de aplicaciones, librerías, módulos del núcleo, etc.

- Java: Es posible instalar JamVM y GNU Classpath. Dependiendo de la versión

del sistema operativo, la versión de Java será 1.4 o 5.0

- Python: Es posible instalar Python 2.5
- Ruby: Es posible instalar Ruby 1.8

#### 1.1.4.2 SunSpot

Sun SPOT [34] es una mota de una red de sensores inalámbricos desarrollada por Sun Microsystems. Estas motas están desarrolladas para hacer uso del estándar IEEE 802.15.4, sobre el cual implementan la capa MAC, sobre la que se puede desplegar ZigBee.

A diferencia de otras motas de redes de sensores inalámbricas. Las Sun SPOT tienen soporte para aplicaciones Java debido a la máquina virtual que utilizan, la Squawk Virtual Machine.

Las características de las Sun SPOT son las siguientes:

**Tabla 11. Características de Sun SPOT.**

| | **Sun SPOT** |
|---|---|
| **Procesador** | ARM920T de 32 bit a 180 MHz |
| **Mem. RAM** | 512 KB |
| **Mem. Flash** | 4 MB |
| **Conectores** | USB |
| **Conexión** | IEEE 802.15.4 con antena integrada |
| **Sensores** | Acelerómetros en 3 ejes 2G/6G, temperatura, luminosidad, 8 LED tricolor |
| **Otras características** | 6 entradas analógicas, 2 botones, 5 pins de entrada/salida de propósito general, 4 pins de salida |
| **Precio** | 630 € por Kit de desarrollo |

#### 1.1.4.2.1 Programabilidad

Las Sun SPOT únicamente pueden ser programadas en Java, más concretamente en Java ME. Actualmente, la forma más sencilla es mediante el IDE Netbeans con la ayuda del plugin de desarrollo para esta plataforma.

#### 1.1.4.2.2 Squawk Virtual Machine

Squawk Virtual Machine [35] es una máquina virtual de Java (JVM) para sistemas embebidos. A diferencia de la mayoría de las JVMs, que están escritas en C/C++ y ensamblador, Squawk está desarrollada casi completamente en Java, lo cual aumenta su portabilidad.

Las características principales de esta JVM son: uso de memoria reducido, provee de un mecanismo de aislamiento por el cual una aplicación es representada como un objeto y es capaz de ejecutar más de una aplicación en una única instancia de la máquina virtual.

### 1.1.4.3 Millennial

Millennial [36] ofrece hardware y software para el desarrollo de WSN de alta calidad y bajo consumo. El rango de aplicaciones que pretende abarcar es muy amplio, como por ejemplo la automatización de la industria, sistemas de seguridad y aplicaciones de baja transferencia de datos, en general.

La plataforma MeshScape proporciona un gran rendimiento en base a una gran escalabilidad, fiabilidad, capacidad de respuesta y eficiencia energética. Utiliza IEEE 802.15.4 como estándar de comunicación.

Los módulos de los que se compone la plataforma MeshScape son:

- *MeshGate*: es el gateway de la red recibiendo información de ella, configurando sus parámetros y puede actuar como un monitor de red.

- *End nodes*: están integrados con sensores y actuadores para capturar la información del entorno.

- *Mesh nodes*: extiende la cobertura de la red encaminando los mensajes, proporcionando rutas de backups en caso de congestión de red o fallos. Los Mesh nodes se puede integrar directamente con los sensores y actuadores en configuraciones Mesh o de estrella.

**Ilustración 14. Módulo de Millenial.**

1.1.4.3.1 Programabilidad

Proporciona un kit de desarrollo que incluye todo lo necesario (software, hardware y accesorios) para crear una sencilla red de sensores wireless.

### 1.1.4.4 Crossbow Motes

Crossbow [37] nació en el año 1995 y ha sido, y sigue siendo, referencia en el desarrollo de sistemas de redes de sensores inalámbricas. Sus productos son conocidos como *Motes*, y en general son plataformas formadas por un microcontrolador y un transmisor. Utilizan un sistema operativo llamado *TinyOS*, de licencia libre, que permite la administración de tareas y eventos con un bajo consumo de energía.

Desde su creación, Crossbow ha desarrollado varios modelos de *Motes*: WeC, René, René2 o Mica. Actualmente, es posible adquirir varios modelos entre los que destacan:

- **Micaz**: plataforma wireless sobre IEEE 802.15.4/Zigbee de majo consumo, en la que todos los nodos tiene capacidad de enrutamiento. Trabaja sobre TinyOS.
- **Mica2**: plataforma wireless que utiliza un transceiver radio a 868/916 MHz, de bajo consumo y en la que todos los nodos tiene capacidad de enrutamiento. Trabaja sobre TinyOS.

- **iMote2**: esta plataforma trabaja sobre el Microframework de .NET, implementa comunicación IEEE 802.15.4/Zigbee y dispone de múltiples interfaces para comunicación con sensores.

**Ilustración 15. Nodos sensores de Crossbow.**

**Tabla 12. Comparativa de los distintos productos Crossbow.**

| | Micaz | Mica2 | iMote2 |
|---|---|---|---|
| **Procesador** | Atmel ATmega128L | Atmel ATmega128L | Intel PXA271 XScale® Processor at 13 – 416MHz |
| **Mem. RAM** | - | - | 256kB SRAM 32MB SDRAM |
| **Mem. Flash** | Programa: 128 KB Medidas: 512 KB | Programa: 128 KB Medidas: 512 KB | 32MB FLASH |
| **Conectores** | En él se pueden conectar una amplia variedad de sensores y placas que permiten la adquisición de datos. | Igual que en Micaz. | Igual que en Micaz. |
| **Conexión** | 802.15.4 / ZigBee | 802.15.4 / ZigBee | 802.15.4 / ZigBee |
| **Sensores** | Conector de expansión de 51 pines. | Conector de expansión de 51 pines. | |
| **Tamaño (mm3)** | 58x32x7 | 58x32x7 | 36x48x9 |
| **Precio** | 90 € | 108 € | 276 € |

#### 1.1.4.4.1 Programabilidad

Tanto las motas Mica2 como Micaz, se pueden programar en nesC, mientras que iMote2 usa C# micro framework.

Además, cabe destacar que Crossbow ofrece kits de desarrollo de diferentes características, en función del desarrollo a realizar:

- **WSN Starters Kit**: es un kit orientado al desarrollo de redes simples de sensores. Incluye pocas unidades de motes.
- **WSN Profesional Kit**: es un kit con 8 motes, y se puede orientar al estudio de redes de sensores más complejas.
- **WSN OEMS Design Kit**: permite el desarrollo de prototipos de forma rápida.
- **WSN Imote2 .Builder Kit**: es uno de los últimos productos de Crossbow, basado en la plataforma .NET de Microsoft. Se reduce la complejidad a la vez que incrementa la productividad.
- **WSN Classroom Kit**: es un kit orientado a entornos académicos donde con muy pocos componentes, y ayudado por software y tutoriales, resulta muy sencillo hacer una pequeña red de sensores.

Todos los kits incluyen todos los componentes, tanto hardware como software, necesarios para el montaje y desarrollo de una red de sensores.

### 1.1.4.5 Dust Networks

La familia de productos de Dust Networks [38] están orientadas a entornos industriales, aunque pueden ser utilizados dentro de otros campos. Proporcionan una solución wireless que permite extender la monitorización y control de forma sencilla, rápida y económica.

Dispone de 3 varias familias de productos: SmartMesh IA-500, SmartMesh-XD y SmartMesh-XT; y en todas ellas se trabaja en 2.4GHz sobre el estándar IEEE 802.15.4 (en la familia XT existen modelos a 900 MHz). Las capacidades de computación, sensorización o capacidades de energía difieren de unas a otras. Como ejemplo, se exponen a continuación las características del modelo M2510 de la familia SmartMesh AI-500.


**Ilustración 16. Modelo M2510 de la familia SmartMesh IA-500.**

El modelo M2510 es un módulo de 22 pines de conexión que puede funcionar con dos pilas de tipo AA, puede funcionar como router o como nodo de forma que la implementación de una topología de tipo Mesh sea muy sencillo. Integra todos los componentes del circuito de radio incluida una pequeña antena.

Dispone de un kit de evaluación con 12 nodos de evaluación y todo el contenido necesario para la realización de diferentes pruebas con el kit.

#### 1.1.4.6 SensiCast

Su producto Sensinet [39] es una completa red de sensores inalámbrica orientada a entornos industriales y comerciales. El sistema incluye todo el hardware y software necesario para controlar procesos y datos ambientales.


**Ilustración 17. Nodo de la red SensiNet**

Este sistema está orientado a su instalación rápida con el hardware y el software que se proporciona, y no la integración de nuevos nodos que no se proporcionen actualmente.

#### 1.1.4.7 SquidBee Mote

SquidBee [40] es una mota realizado con hardware libre, que gracias a sus sensores de humedad, temperatura o luminosidad es capaz de captar datos del entorno y

enviarlos de manera inalámbrica por la red utilizando el protocolo ZigBee. El objetivo del proyecto es la creación de un “open mote” para la creación de redes de sensores inalámbricas.

**Ilustración 18. Modelo de mota de SquidBee Mote**

Los desarrolladores de este sistema son los mismos que han desarrollado la plataforma Arduino, y en la página principal del proyecto se proporciona toda la información necesaria para su utilización.

**Tabla 13. Características de SquidBee Mote.**

| | SquidBee Mote |
|---|---|
| **Procesador** | Atmel ATmega8 o ATmega168 |
| **Mem. RAM** | 1 KB SRAM |
| **Mem. Flash** | 512 B EEPROM |
| **Conectores** | - |
| **Conexión** | IEEE 802.15.4 con antena integrada |
| **Tamaño** | 70x60x30 |
| **Precio** | - |

#### 1.1.4.8 Comparativa

A modo de resumen, se ha realizado una tabla comparativa con las características anteriormente descritas, escogiendo los modelos superiores de cada línea de productos para aquellas en las que había más de uno. Lógicamente en ISMED se adquirirán los kits con unas especificaciones de procesador o memoria superiores.

**Tabla 14. Comparativa del grado de soporte de SWRL por parte de los motores de razonamiento**

| | Procesador | Mem. RAM | Mem. Flash | Slots | Conexión | Sensores | Otras características | Precio |
|---|---|---|---|---|---|---|---|---|
| **Sun SPOT** | ARM920T de 32 bit a 180 MHz | 512 KB | 4 MB | 6 entradas analógicas, 2 botones, 5 pins de entrada/salida de propósito general, 4 pins de salida | IEEE 802.15.4 con antena integrada | Acelerómetros en 3 ejes 2G/6G, temperatura, luminosidad, 8 LED tricolor | Conector USB | 630 €, con kit de desarrollo |
| **Basix 400 xm-bt** | Intel XScale PXA255 a 400 MHz | 64 MB | 16 MB | 1 x 60 pin | Bluetooth | - | Permite conectar tarjetas de memoria MMC | 169 $ |
| **Connex 400 xm-bt** | Intel XScale PXA255 a 400 MHz | 64 MB | 16 MB | 1 x 60 pin, 1 x 92 pin | Bluetooth | - | | 169 $ |
| **Verdex xl6p** | Marvell PXA270 con XScale a 600 MHz | 128 MB | 32 MB | 1 x 60 pin, 1 x 120 pin, 1 x 24 pin flexible | Bluetooth | - | USB Host signals, CCD Camera signals | 169 $ |

| | Procesador | Mem. RAM | Mem. Flash | Slots | Conexión | Sensores | Otras características | Precio |
|---|---|---|---|---|---|---|---|---|
| **Verdex pro** | Marvell PXA270 con XScale a 600 MHz | 128 MB | 32 MB | 1 x 60 pin, 1 x 80 pin, 1 x 24 pin conector flexible y plano | - | - | Tarjeta Micro SD USB Host signals, CCD Camera signals | 169 $ |
| **Overo Earth** | Procesador de aplicaciones OMAP 3503 con CPU ARM Cortex-A8 a 600 MHz | 256 MB DDR de bajo consumo | 256 MB NAND | 2 x 70 pin, 1 x 27 pin conector flexible y plano | - | - | Tarjeta Micro SD; Soporte para placas base “OMAP 35x-based Overo”, bus I2C, 6 x líneas PWM, 6 x A/D, 1-wire, UART, entrada de cámara, entrada de micrófono, salida de auriculares, entrada batería de reserva, USB OTG signal, USB HS host | 149 $ |

| | Procesador | Mem. RAM | Mem. Flash | Slots | Conexión | Sensores | Otras características | Precio |
|---|---|---|---|---|---|---|---|---|
| **SquidBee Mote** | Atmel ATmega8 o ATmega168 | 1 KB SRAM | 512 B EEPROM | - | IEEE 802.15.4 con antena integrada | - | - | - |

## 1.2 Aprendizaje

### 1.2.1 Introducción

Los entornos inteligentes o entornos de inteligencia ambiental plantean un cambio de paradigma desde una visión techno-centred a otra human-centred donde la tecnología sería el que se adaptaría a las necesidades, preferencias, hábitos, costumbres,… de los usuarios. Así el entorno debe ser capaz de reconocer a las personas, aprender de su comportamiento e incluso anticiparse a sus necesidades. Estos entornos han sido definidos como “digital environments that proactively, but sensibly, supports people in their daily lives” [41].

Los entornos inteligentes, como paradigma, tienen potencial para hacer un gran impacto en la vida diaria de los usuarios, alterando positivamente las relaciones entre el usuario y las tecnologías.

Hay que destacar que hasta la fecha la AmI ha atraído a muchos investigadores y algunas aplicaciones han sido desarrolladas con diferente grado de éxito. Teniendo en cuenta la complejidad de dichos entornos, donde el hardware, software y sistemas de comunicación tienen que cooperar eficientemente, cada proyecto se ha centrado en diferentes aspectos de esa compleja estructura. En ese sentido, es comprensible e incluso lógico que los primeros desarrollos se hayan centrado en la parte de hardware y comunicaciones como forma de soporte para el resto de las aplicaciones. Esto ha supuesto que la inteligencia de dichas aplicaciones sea solo simples automatizaciones que actúan de forma reactiva en el entorno. Así, se ha identificado la necesidad de dotar de inteligencia a esos entornos como una de las prioridades para alcanzar el paradigma de entornos inteligentes. Aunque hasta la fecha han sido muchos los investigadores [42-45] que han destacado la importancia de este aspecto pocos trabajos han sido desarrollados en ese sentido, aunque con notables excepciones como se verá a continuación.

Una de las ideas subyacentes al paradigma de inteligencia ambiental es que a diferencia de los entornos actuales donde el usuario tiene que adaptarse al entorno y tiene que aprender cómo usar la tecnología, en entornos de AmI es el entorno mismo el que se adapta al usuario. Así, el usuario debería estar exento de alguna obligación de programar dispositivos [46]. El entorno automáticamente debería aprender cómo

reaccionar a las acciones de los usuarios, todo ello de una forma no intrusiva. Así, la capacidad de aprendizaje se convierte en un aspecto indispensable de los entornos inteligentes.

Aprendizaje, en entornos de AmI, significa que el entorno tiene que obtener conocimiento acerca de las preferencias, hábitos,… de los usuarios para poder acuerdo a ellos [47, 48]. El aprendizaje de patrones permite:

- Predecir las necesidades y acciones del usuario. Así, automatizando dichas acciones (por ejemplo, ajustando automáticamente la temperatura de acuerdo a las preferencias del usuario) se puede facilitar la vida diaria del usuario [49, 50]
- Hacer el sistema más eficiente en términos de eficiencia energética [51] (por ejemplo, apagando las luces si el usuario ha salido y no va a volver en 1 hora).
- Aumentar la seguridad [52-54] (por ejemplo emitiendo alarmas cuando se detectan situaciones fuera de los normal)

Como todo nuevo paradigma que impacta en la vida diaria de las personas, la AmI no ha quedado exenta de críticas. Hay suspicacias acerca de la perdida de la privacidad o el aumento de la individualidad en la sociedad. Estas críticas también fueron vertidas a la computación en general de modo que la AmI, como todo nuevo paradigma, tendrá que afrontar estos aspectos de forma adecuada, dándole una mayor importancia al aspecto de human-centric [55].

### 1.2.2 Características específicas de entornos AmI

Los entornos de inteligencia ambiental tienen ciertas características específicas, es decir, diferentes a otros entornos, que hay que tener en cuenta a la hora de llevar a cabo el aprendizaje. Estas características han sido agrupadas en cuatro grupos.

#### 1.2.2.1 Importancia de los usuarios

La primera característica diferenciadoras de los entornos AmI es la importancia que se le da al usuario, por ejemplo Leake et al. [55] comentan que además de ser el centro de todo el desarrollo, el usuario tiene que tener el poder sobre todos los dispositivos. El usuario es el centro de la AmI. Al usuario no le tiene que suponer ningún esfuerzo añadido que el entorno se convierta inteligente. El cambio que debe notar el usuario es que el entorno realiza tareas de forma adecuada que antes los realizaba él y le

suponían cierto esfuerzo. Si el entorno no realiza las tareas de forma adecuada el usuario deberá realizar correcciones, suponiendo un esfuerzo, que si es repetitivo puede ser un factor de no aceptación.

Así, Marzano [56] menciona que el hogar del futuro se asemejará más al hogar del ayer que al hogar actual ya que los dispositivos estarán integrados en los objetos cotidianos y no será necesario la inserción de nuevos objetos para dotar de inteligencia al entorno (Ver Figura 1). En este sentido menciona que la idea de equipar el hogar con dispositivos que aumenten la calidad de vida y reduzcan el trabajo no es novedosa, sino que la novedad que introduce la AmI es el de la transparencia y la posibilidad de interacción con el entorno.

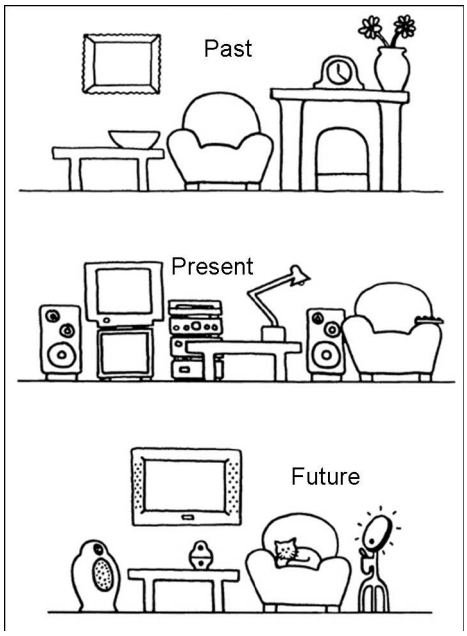


**Ilustración 19. Hogar del futuro.**

Otro aspecto a tener en cuenta es la interacción entre el usuario y el entorno. Dicha interacción debe ser a través de dispositivos comunes. El usuario está acostumbrado al uso de ciertos dispositivos. Así, la información necesaria para la generación de reglas como el feedback posterior tienen que ser obtenidos a partir de sensores estándares. Introducir nuevos dispositivos o nuevos interfaces para la interacción ha sido criticado por autores como Mozer [57] o Rutishauser [58], ya que el usuario está acostumbrado a usar dispositivos normales. Además habrá muchos usuarios que rechacen la idea de AmI si para ello hay que aprender a usar nuevos dispositivos o nuevos interfaces de usuario y ese aprendizaje no conlleva grandes beneficios.

Además los entornos AmI tienen que adaptarse a usuarios particulares y no a usuarios genéricos, aunque algunas reglas pueden ser relativas a usuarios genéricos o grupos de usuarios. Hay que destacar la complejidad de los comportamientos de los usuarios. Liao et al. [53] en su trabajo para reconocer el comportamiento destaca que 'Human behaviors involve many uncertain factors and are hard to model deterministically'. Así, Lialo considera que el 'probabilistic reasoning' parece un approach interesante. Otras características del comportamiento humano que Liao menciona en su trabajo son:

- Muchos factores que afectan al comportamiento (preferencias y capacidades) son difíciles de caracterizar.

- Existen grandes diferencias entre lo que miden los sensores (lo que observamos) y el comportamiento (lo que queremos saber)

- El comportamiento humano es muchas veces complejo. Kulkarni [59] menciona un ejemplo de este tipo de comportamiento. Imagina una habitación que tiene la regla de que si detecta que hay alguien en la habitación que está despierto encienda las luces. Esto en un principio puede parecer lógico, pero Kulkarni menciona que el ser humano tiene comportamientos complejos que no pueden ser reducidos a reglas tan simples. Por ejemplo hay personas que quieren ver una película en su habitación con las luces apagadas.

- Los usuarios cometen errores y reconocerlos es muy importante para no confundirlos con falsos deseos.

Debido a todas las posibles complicaciones expuestas anteriormente, además de los datos obtenidos de los sensores, conocimiento adquirido externamente será utilizado para llevar a cabo el aprendizaje. Este conocimiento externo puede ser:

- Información médica del paciente.

- Preferencias explicitas.

- Información acerca de actividades (por ejemplo, cuando se va de vacaciones).

#### 1.2.2.2 Naturaleza de los datos

Los datos recogidos de los sensores influenciará de forma definitiva el proceso de aprendizaje, ya que éste estará basado en dichos datos. Esta dependencia es incluso mayor si consideramos la problemática de recoger e interpretar dichos datos de forma correcta.

Los datos de los sensores llegarán de forma continua de diferentes fuentes, de modo que la integración de dicha información conllevará ciertos retos a tener en cuenta. Diferentes formatos, diferentes estados de diferentes dispositivos incrementan la complejidad de dicha integración.

Otro aspecto a tener en cuenta es como se ha indicado anteriormente los datos necesarios para el aprendizaje se recogerán desde los sensores, por lo que habrá diversas fuentes de información [58, 60], lo que puede exigir la sincronización y la coordinación de los mismos.

Por otro lado, considerando la naturaleza de la información recogida, teniendo en cuenta que es referente a comportamientos del usuario, hay que considerar acciones contradictorias entre sí. En este sentido, como se ha indicado en la anterior sección, conocimiento adquirido desde el exterior podría ayudar a esclarecer dichos casos.

Finalmente tal como indican Rivera-Illingworth et al. [52], hay que tener en cuenta también la diferencia entre entornos de laboratorio y entornos reales, ya que por ejemplo cámaras y micrófonos pueden dar un resultado óptimo en entornos de laboratorio mientras que en entornos reales pueden ser fuente de muchos problemas.

#### 1.2.2.3 Representación del conocimiento adquirido

Como el conocimiento adquirido, es decir, los patrones aprendidos serán utilizados dentro de un sistema más grande, habrá que considerar y analizar cómo representar el output del sistema de aprendizaje.

Si el conocimiento adquirido tiene que ser combinado con otro tipo de conocimiento sería preferible que dicho conocimiento sea comprensible. Además de combinarse con otro tipo de conocimiento, en ciertos entornos el entorno tendrá que dar “explicaciones” de por qué ha actuado de la forma en que lo ha hecho, por lo que en

estos casos también sería preferible que la salida sea comprensible, para que, de esa forma el entorno pueda explicar al usuario su lógica de actuación [55].

Otro aspecto a considerar es la representación en si de los patrones, es decir, cómo, de una forma sencilla, representar las preferencias, hábitos,… de los usuarios. En este sentido, la utilización de secuencias para la representación de las actividades comunes de los usuarios parece prometedora.

#### 1.2.2.4 Adaptación temporal de los patrones

Para adaptarse a las necesidades o preferencias del usuario, el entorno debe de tomar decisiones continuamente, y esas decisiones serán tomadas en base al conocimiento que tiene el sistema o cada agente. Pero este conocimiento se tiene que cambiar con el tiempo.

Se pueden definir diferentes etapas en el aprendizaje de los patrones:

- **Generación inicial de conocimiento:** La primera necesidad o reto que surge es la generación del conocimiento o “rulebase”. Para adaptarse a las necesidades o preferencias del usuario, el entorno debe de tomar decisiones continuamente, y esas decisiones serán tomadas en base a un conocimiento (reglas, casos,…) que tiene el entorno. Ese conocimiento puede ser generada de distintas maneras. Una opción sencilla parece ser que el mismo usuario o un experto construya o defina ese conocimiento, pero teniendo en cuenta ciertas características de entornos AmI (no intrusivo, transparente,…) se trata de generarla automáticamente utilizando algoritmos de aprendizaje automático o machine learning [58]. Además no es efectivo ni en tiempo ni en esfuerzo implementar (y menos mantener) el conocimiento manualmente. La generación de conocimiento y por lo tanto la adaptación del entorno al usuario puede ser pensado en distintas etapas, ya que la generación de conocimiento (extracción de perfiles de usuario) puede llevar un tiempo considerable. Así, la primera adaptación del entorno (adaptación a corto plazo) puede ser llevado a cabo sin la extracción de perfiles de usuario. La adaptación (adaptación a largo plazo), llevado a cabo mediante la predicción y anticipación de las necesidades del usuario, necesita de perfiles de usuario extraídos a partir de un conjunto de datos extensos que necesita tiempo en recopilarlos, por lo que también es necesario la adaptación a corto plazo mencionado anteriormente.

- **Adaptación del conocimiento:** Una vez que se hayan definido el conocimiento, el entorno va a tomar decisiones y actuar en consecuencia. Al referirse a estos conocimientos hay que darse cuenta de que se refieren a gustos, preferencias, hábitos,… de los usuarios, y que pueden cambiar a lo largo del tiempo. La necesidad de adaptación del conocimiento puede venir por algunas de las siguientes situaciones:

    - Posible modificación de comportamientos, necesidades, costumbres, gustos o preferencias del usuario.

    - Posible modificación del contexto en el que se sitúa el usuario con la eliminación, modificación o inclusión de nuevos dispositivos.

    - Existencia de situaciones no previstas.

    - Conocimiento no optimizado.

    Así, hay una clara necesidad de que este conocimiento sea adaptada, refiriéndose con adaptación a la posible modificación (ajuste) de alguno de los valores, a la generación de un nuevo conocimiento e incluso a la eliminación de un conocimiento no válido.

- **Actuación inteligente sin aprendizaje:** Para extraer patrones de comportamiento a partir de los datos recogidos “Generación inicial de conocimiento” la primera tarea es la misma recolección. Esta recolección implica la necesidad de estar observando las tareas o acciones que realiza el usuario en un periodo de tiempo. En ese periodo de tiempo el entorno no actuará, pudiendo ser ese periodo de 5 días, 1 semana, 2 semanas o más. Durante ese primer periodo lo ideal sería que el entorno también tratase de adaptarse al usuario. Esto podría ser realizado mediante técnicas de aprendizaje que no requieren entrenamiento, p.e. Case-Based Reasoning. Además esto podría proveer al sistema los primeros feedbacks del usuario.

Así, se puede concluir que primero habrá un periodo de actuación inteligente basado en técnicas que no necesita entrenamiento. A continuación otro periodo de aprendizaje realizado con datos recolectados durante un periodo de tiempo donde se extraerá el conocimiento, y finalmente habrá una adaptación continua de ese conocimiento tratando de adaptarse a los cambios de comportamiento que pueden existir.

### 1.2.3 Técnicas de Machine Learning

La utilización de técnicas de machine learning para el aprendizaje de patrones parece muy prometedor, debido a que históricamente han sido estas técnicas las usadas para llevar a cabo todo tipo de aprendizajes en todo tipo de entornos.

Lo que hay que destacar es que técnicas de machine learning que han sido útiles en otros entornos pueden no ser satisfactorios en entornos de inteligencia ambiental debido a las características especiales de dichos entornos. En este sentido, a continuación se muestra un análisis de diferentes técnicas de aprendizaje incidiendo en sus fortalezas (✓) y debilidades (×) para el aprendizaje de patrones en entornos AmI. No todas las técnicas de aprendizaje han sido consideradas sino que han sido seleccionadas las más prometedoras y las que han sido utilizadas por diferentes grupos para dicha tarea.

**Árboles de decisión**

- ✓ La salida puede ser traducida a un conjunto de reglas fácilmente interpretadas.
- ✓ Produce reglas que son no-ambiguas, de modo que no es relevante el orden en el que se ejecutan y no hay conflictos entre ellos.
- ✓ Funciona bien incluso cuando hay errores en los datos de entrenamiento o hay valores nulos.
- × El añadir o eliminar reglas puede exigir la re-estructuración de todo la estructura del árbol.
- × Las reglas que son extraídas directamente de los árboles de decisión pueden ser más complejas de lo necesario. En entornos complejos, esta característica puede ser una complicación innecesaria.
- × Dificultades para representar secuencia temporales. Aunque investigaciones recientes indican que se pueden generar árboles de decisión temporales.

**Aprendizaje de reglas**

- ✓ Las reglas son la representación más sencillas para la interpretación del conocimiento.
- ✓ Es fácil introducir excepciones en ellas
- ✓ Añadir, modificar o borrar una regla de la estructure existente es fácil.
- ✓ En el proceso de aprendizaje se puede especificar el límite para su aceptación.

- × Es posible que las reglas no cubran todas las opciones o casos. (✓) Pero esos casos se pueden utilizar para la adaptación de dicho conocimiento.
- × Hay posibilidad de conflicto entre las reglas, es decir, en la misma situación puede haber dos reglas con consecuencias incompatibles.
- × Como el conjunto de reglas es generado secuencialmente, un pequeño cambio puede hacer que todo el conjunto sea inconsistente.

**Descubrimiento de secuencias**

- ✓ El comportamiento humano es muchas veces mejor representado mediante secuencias que por medio de acciones independientes.
- ✓ El uso de secuencias permite usar tiempos relativos entre las distintas acciones.
- ✓ Añadir una nueva acción a una secuencia o una nueva secuencia al conjunto de secuencias es fácil.
- ✓ En el proceso de descubrimiento se puede establecer el límite para su aceptación.
- × El descubrimiento de secuencias ha estado orientado hasta la fecha a descubrir eventos que suceden dentro de un periodo de tiempo.

**Redes Neuronales Artificiales**

- ✓ Los datos de entrenamiento pueden contener errores.
- ✓ Robusto a los errores de entrenamiento.
- ✓ Aplicable a problemas complejos tales como aprendizaje de comportamientos o asociaciones entre datos.
- × La estructure no es comprensible. Sólo las entradas y las salidas pueden ser interpretadas.
- × El tiempo de entrenamiento puede ser largo. Además, debido a su estructura estática, necesita ser re-entrenada cuando se presentan nuevos datos.
- × Posibilidad de sobre-entrenamiento.

**Instance based learning**

- ✓ No necesita ningún periodo entrenamiento porque no estiman una solución para todo el espacio, sino que lo hace por cada nueva instancia.

- ✓ La adaptación es rápida. A diferencia de otras técnicas la adaptación a nuevos casos o conocimiento es rápida.
- × No hace explicito la estructura del conocimiento adquirido. Pero la captura de casos similares puede ser utilizado para explicar porqué el sistema ha actuado de esa forma.
- × Es lento para conjuntos de datos grandes. Este problema podría ser afrontado mediante estereotipos creados mediante la agrupación de casos similares.
- × Establecer pesos es difícil.

**Reinforcement Learning**

- ✓ Útil para entornos que aprenden su comportamiento basándose en la continua interacción con el usuario, donde las acciones de los usuarios son interpretados como premio o castigo de las acciones realizadas por el entorno.
- × Dificultad para interpretar el feedback del usuario. Además si el feedback es obtenido después de ciertas acciones es muy difícil estimarlo.

La Tabla 15 define las fortalezas y debilidades de cada técnica referente a los periodos de aprendizaje definidos anteriormente.

**Tabla 15. Técnicas de Machine Learning con sus fortalezas y debilidades.**

| | Fortalezas/ Debilidades | Árboles de Decisión | Aprendizaje de reglas | Descubrimiento de secuencias | Redes Neuronales | Instance Based Learning | Reinforcement Learning |
|---|---|---|---|---|---|---|---|
| **Aprendizaje en un periodo corto** | Fortalezas | Salida comprensible | Salida comprensible | Salida comprensible | ------- | No necesita entrenamiento. | ------- |
| | Debilidades | Necesidad de entrenamiento. | Necesidad de entrenamiento. | Necesidad de entrenamiento. | Necesidad de entrenamiento. Salida no comprensible | Sensible a instancias complicadas y erróneas | Necesidad de un modelo |
| **Aprendizaje de patrones** | Fortalezas | Salida comprensible | Salida comprensible. Posibilidad de excepciones. | Reglas que representan relaciones temporales | Capacidad de generalizar | Extracción de instancias generales. | -------- |
| | Debilidades | Dificultad para representar situaciones complejas (p.e. excepciones). | Posible conflictos entre reglas | Algoritmos no centrados en descubrir los lapsos de tiempo | Salida no comprensible | Lento con instancias complicadas | Necesidad de un modelo |
| **Adaptación de patrones** | Fortalezas | Facilidad para identificar el error | Facilidad para añadir, modificar o eliminar reglas. | Facilidad para añadir, modificar o eliminar reglas. | Posibilidad de introducir neuronas dinámicamente. | Posibilidad de clasificar instantes no conocidas. | Aprende interactuando con el usuario. |
| | Debilidades | Necesidad de reformular el árbol. | ------ | ------ | Necesidad de entrenar otra vez | Costes de computación y de tiempo. | Dificultades para interpretar los feedbacks. |

Actualmente es muy difícil establecer una aproximación global que pueda ser útil para todos los entornos. Teniendo en cuenta las necesidades, los objetivos, … de cada entorno, diferentes técnicas pueden ser utilizados. También debido a las características de diferentes técnicas, la combinación de diferentes técnicas puede ser una buena solución. Si por alguna razón se decide que la combinación de diferentes técnicas es interesante, los interfaces entre ellos deben ser definidos claramente. Un ejemplo de dicha combinación puede ser la utilización de Instance based learning para actuar de forma inteligente mientras se recogen los datos y luego utilizar alguna técnica que necesite entrenamiento (p.e. aprendizaje de secuencias), utilizando reinforcement learning para la adaptación de dichos patrones.

### 1.2.4 Utilización de técnicas y grupos de investigación

#### 1.2.4.1 Técnicas utilizadas

Aunque, como se ha indicado anteriormente, poco énfasis se ha hecho hasta la fecha en lo que se refiere al aprendizaje, ha habido excepciones que merecen ser mencionados. Teniendo en cuenta las técnicas definidas en la anterior sección, se van a analizar los trabajos realizados con dichas técnicas.

##### 1.2.4.1.1 Redes Neuronales artificiales

Chan et al. [61] y Mozer et al. [57, 62, 63] fueron de los primeros que trataron de inferir reglas sobre patrones de usuario para entornos inteligentes basándose en la vida diaria de los usuarios. Plantearon la utilización de redes neuronales artificiales para predecir el futuros estatus y controlar dispositivos tales como la calefacción, la televisión o las luces. A partir de ese momento muchos otros investigadores han utilizado las redes neuronales para el desarrollado de entornos inteligentes [52, 64-67]. Incluso se ha realizado una recopilación de dichos trabajos en un artículo [68].

##### 1.2.4.1.2 Técnicas de clasificación

Técnicas de clasificación, tales como árboles de decisión o aprendizaje de reglas también han sido utilizadas para el aprendizaje en entornos inteligentes. Un ejemplo de ello es el SmartOffice [69] donde el entorno trata de anticipar las intenciones de los

usuarios. Teniendo en cuenta las pruebas realizadas Brdiczka et al. [70] definen que los árboles de decisión dan los mejores resultados comparado con las técnicas de Find-S y Candidate Elimination [71].

#### 1.2.4.1.3 Aprendizaje de secuencias

Una técnica que está adquiriendo importancia para la predicción de actividades es el aprendizaje de secuencias que permite definir secuencias de acciones del usuario. Así, en [72] Jakkula y Cook desarrollan un framework general para representar y razonar sobre relaciones temporales entre acciones en entornos inteligentes. Representan las relaciones temporales utilizando las relaciones definidas por Allen [73, 74] y luego utilizan técnicas de data mining para el descubrimiento de asociaciones frecuentes (usando el algoritmo del A priori [75]).

#### 1.2.4.1.4 Instance Based Learning

Case-Based Reasoning, el cual se puede considerar como una técnica incluida dentro del Instance based learning, ha sido utilizado para el aprendizaje de preferencias del usuario. Por ejemplo, en el proyecto de MyCampus [76] utilizaban dicha técnica para filtrar los mensajes en base al feedback del usuario. Almacenaban información acerca en forma de casos y cuando una nueva situación se presentaba lo clasificaban comparando con los casos previos. En este caso, las relaciones mediante atributos, para obtener las métricas de similitud, fueron definidos mediante redes bayesianas. Nuevas aproximaciones basadas en Case-based reasoning, tales como Temporal CBR [47] han sido sugeridas también.

#### 1.2.4.1.5 Reinforcement learning

La técnica de reinforcement learning puede ser considerado como una técnica de adaptación. Por ejemplo, Chidambaram et al. [57] utilizaban dicha técnica conjuntamente con otras técnicas, para, teniendo en cuenta los feedback negativos que daba el usuario mejorar la exactitud de las predicciones.

#### 1.2.4.1.6 Otras técnicas

Aparte de las técnicas definidas anteriormente otras técnicas también han sido utilizadas para llevar a cabo el aprendizaje en entornos inteligentes.

En el proyecto MavHome [43, 50, 77] los autores intentan modelar el comportamiento de los usuarios mediante cadenas de Markov para maximizar el confort de los mismos. Considerando las dificultades que tienen técnicas tradicionales como los árboles de decisión de utilizar información histórica para la predicción secuencial [78], sugieren la utilización de cadenas de Markov. Algunos investigadores [52, 79] también han mencionado las dificultades que pueden tener los Hidden Markov Models debido a que son incapaces de afrontar comportamientos complejos y su algoritmo de aprendizaje tiene dificultades cuando se trata de información proveniente de numerosos sensores de bajo nivel. Así, algunos autores han preferido utilizar Hidden Semi-Markov Model dado que generaliza la duración de los estados.

Investigadores de la universidad de Essex [46, 80] han trabajo con la lógica fuzzy para el aprendizaje de patrones. Así, basándose en los datos recogidos por los sensores, tratan de inferir las funciones de pertenencia y las reglas del sistema de lógica fuzzy, el cual es capaz de adaptarse.

### 1.2.4.2 Grupos de investigación y/o entornos

Aunque la investigación del aprendizaje no haya sido hasta la fecha una de las prioridades entre los investigadores de AmI, hay notables excepciones. Así, hay grupos de investigación que viendo la necesidad de dotar de inteligencia el entorno, más allá de las necesidades de conectividad y comunicación, hayan empezado a trabajar en el aprendizaje.

A continuación se detallarán los grupos de investigación y entornos más activos.

#### 1.2.4.2.1 University of Texas / MavHome

El proyecto MavHome (Managing an adaptive versatile Home) trata de maximizar el confort del usuario, minimizando el consumo de recursos y manteniendo la seguridad y privacidad del usuario [77]. La idea de MavHome está siendo desarrollado e implementado en “MavLab” y en “MavPad” (un apartamento del campus habitado por

estudiantes) (Ver Figura 2). En dicho entornos están automatizados el control de luces y electrodomésticos, así como el aire acondicionado, ventiladores y persianas.

**Ilustración 20. MavHome y MavLab**

1.2.4.2.2 University of Ulster

La universidad de Ulster en su trabajo conjunto con el hospital del Ulster trabaja con un entorno de 30 apartamentos individuales que trata de proporcionar soluciones tecnológicas para una vida más independiente para las personas mayores [44].

1.2.4.2.3 University of Essex/iDorm

La universidad de Essex cuenta con un banco de pruebas donde hace test de sus mecanismos de aprendizaje y adaptación [81]. El iDorm (Ver Figura 3) es un espacio multiuso que contiene áreas para dormir, trabajar o entretenimiento. En estos momentos los creadores de iDorm están desarrollando el iDorm2, un apartamento de 2 habitaciones que trata de mejorar las prestaciones del iDorm [82].

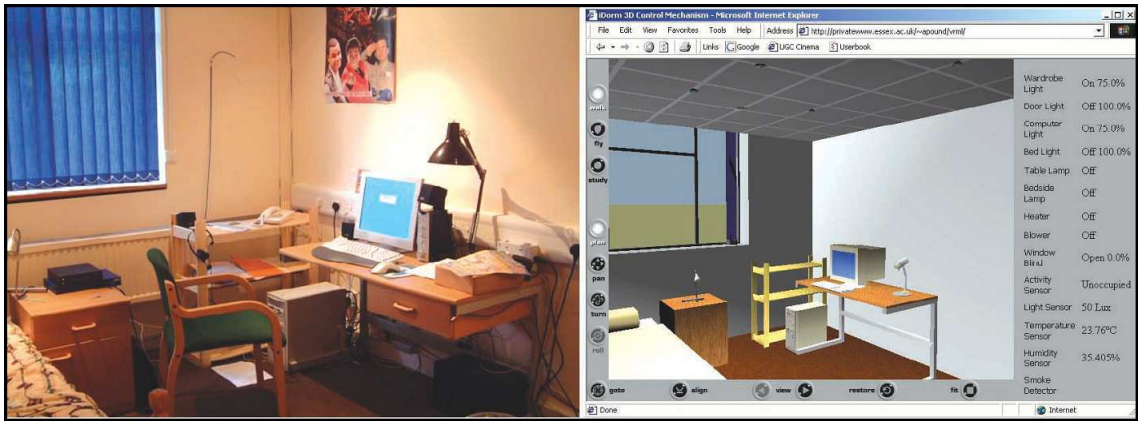


**Ilustración 21. iDorm**

#### 1.2.4.2.4 Georgia Institute of Technology/AwareHome

Esta casa cuenta dos espacios idénticos pero independientes entre sí. Cada espacio cuenta con dos habitaciones, dos cuartos de baño, una oficina, una cocina, sala de estar y cuarto para la limpieza. Han desarrolla algunas aplicaciones, por ejemplo una para encontrar objetos perdidos.

#### 1.2.4.2.5 Philips/HomeLab

Philips, con su HomeLab, ha sido una de las más activas entre las empresas del sector industrial. Sobre todo Philips se centra en la interacción y en cómo puede el hogar ayudar en la realización de tareas diarias. Además le da gran importancia al "social awareness" para que sea adecuado y aceptable para los usuarios. Algunos desarrollos concretos utilizados en HomeLab [83] son por ejemplo "AmI lightings Living light" donde se trata de crear el ambiente idóneo para ver la televisión o escuchar música mediante el control de las luces. Otra idea es "Sharing context and experiences" donde se trata de compartir espacios comunes aún cuando físicamente no se comparta ese espacio.

#### 1.2.4.2.6 Siemens

Siemens ha invertido en Smart Homes tratando de enriquecer el entretenimiento teniendo en cuenta la seguridad y la economía. Control de luces, ahorro de energía, pantallas táctiles, seguridad de dispositivos o monitorización de niños son algunas de los aspectos que trata Siemens.

#### 1.2.4.2.7 Intel/Proactive Health Group

Intel se centra en la investigación para aumentar la calidad de vida de los ancianos, dando importancia al “social network”.

#### 1.2.4.2.8 Otros entornos

Aunque los arriba mencionados sean los grupos y los entornos más significativos, hay otros grupos que utilizando entornos más sencillos y más reducidos investigan aspectos concretos.

Uno de los aspectos donde los investigadores más se han centrado es la ayuda a personas de la 3a edad y discapacitados, teniendo en cuenta la perspectiva de las personas cuidadas como de los cuidadores.

- Muchos han sido los autores que han mencionado los centros de ayuda a las personas de 3a edad y discapacitados. Residencias, hospitales, casas medicalizadas y apartamentos de ancianos son algunos de los entornos mencionados [44]. ‘Health Monitoring and Assistance’ es otro de los ámbitos donde la AmI puede ser aplicada [77]. Los discapacitados son otro grupo de posibles beneficiarios de AmI [84].

- Martin et al. [85] definen un entorno de teleayuda para aumentar el bienestar de los ancianos comprobando si cambian los patrones de comportamiento.

- Lühr et al. [86] también se centran en detectar desviaciones del comportamiento habitual, y son conscientes de que para ello necesitan tener un modelo de comportamiento para poder compararlo.

- Chan et al. [87] y Rivera-Illinworth et al. [52] tratan de desarrollar un sistema capaz de monitorizar a personas de 3a edad y discapacitados para permitir una vida independiente. Para ello han diseñado un sistema de monitorización, que ha sido instalado en 12 habitaciones, para que la gente tenga más seguridad sin causarles ningún problema adicional.

- Demiris et al. [88] ven la necesidad de un sistema AmI para ayudarles a las personas mayores a seguir siendo independientes en lo posible. Para ello,

además de a los ancianos, Demiris et al. ven la necesidad de involucrar a los cuidadores en el desarrollo de este sistema. En entorno donde están aplicando este sistema es un entorno real de una residencia de ancianos de 32 habitaciones.

- Pollack [89], al analizar la tendencia demográfica de USA y plantear la necesidad de sistemas que ayuden a la gente a vivir independientemente, menciona que con dichos sistemas los cuidadores no serán sustituidos, sino que la tecnología mejoraría la calidad de vida tanto de los pacientes como de los cuidadores.

- Kautz et al. [54] tratan con enfermos de Alzheimer ayudándoles con dos demostradores a orientarse y realizar múltiples tareas diarias.

### 1.2.5 Conclusiones

De lo expuesto en el documento se pueden extraer varias conclusiones:

- El aprendizaje automático de patrones de comportamiento de los usuarios es necesario si se quieren obtener entornos inteligentes de una manera no intrusiva.

- La adaptación a lo largo del tiempo de estos patrones también es necesario ya que estos patrones pueden cambiar.

- Hasta la fecha, la mayoría de las investigaciones llevadas a cabo en AmI no ha hecho mucho énfasis en la necesidad de aprendizaje y adaptación, ya que se han centrado más en temas relativos a sensores y redes, aunque con notables excepciones

- El aprendizaje tiene que llevarse a cabo sin que ello le suponga un esfuerzo añadido al usuario, o en su defecto, tratando que el usuario interactúe mediante interfaces estándares o interfaces multimodales de fácil uso. De la misma forma, el feedback del usuario, necesario para la adaptación del conocimiento, tiene que recogerse preferentemente a través de dispositivos estándares o interfaces multimodales de fácil uso.

La utilización de algoritmos de Machine Learning para el aprendizaje de tales patrones parece una opción lógica, pero la(s) técnica(s) de Machine Learning utilizada(s) tiene(n) que tener en cuenta las características específicas de entornos AmI, por ejemplo:

- Tipología de datos: Ruido en los datos, Diversas fuentes de información, etc.
- Objetivos que se persiguen: Confort del usuario, Ahorro de energía, etc.
- Capacidad de los dispositivos: Limitaciones de memoria y procesamiento, etc.
- Capacidad de razonar la decisión tomada, por lo que puede ser importante la interpretabilidad de lo aprendido.

Así, la elección de qué técnica de Machine Learning utilizar puede depender de las características arriba mencionadas, del entorno particular que tenemos, preprocesado que se ha hecho de los datos obtenidos, etc. Hasta la fecha, diferentes autores han utilizado diferentes técnicas dando cada uno da importancia a diferentes aspectos, por lo que no se ha podido establecer una técnica como la más usada o como la que da mejores resultados. Así como apunta Muller “In many research projects, great results were achieved... but the overall dilemma remains: there does not seem to be a system that learns quickly, is highly accurate, is nearly domain independent, does this from few examples with literally no bias, and delivers a user model that is understandable and contains breaking news about the user’s characteristics. Each single problem favours a certain learning approach”.

### 1.3 Composición de servicios con técnicas de planificación y workflow

La composición de servicios consiste en conectar servicios entre sí para crear procesos de negocio más complejos o de alto nivel, facilitando de esta manera el desarrollo de nuevas aplicaciones reutilizando componentes ya existentes [90]. De esta manera dado un conjunto de servicios y con métodos eficaces de composición automática de servicios la visión de la programación en la que se especifica qué es lo que tiene que hacer el programa y no el cómo lo tiene que hacer puede llegar a ser una realidad [91].

En el área de los WS el enfoque de los SWS [92] es un paso importante hacia el

descubrimiento y la composición dinámica [93], donde sistemas inteligentes tratan de componer servicios autónomamente, basándose en requisitos de usuario abstractos. Todo ello mediante SWS construidos basándose en técnicas de representación del conocimiento, con ontologías que describen el dominio de manera formal y técnicas de planificación de IA para hacer los sistemas de composición más autónomos [94, 95].

Fujii y otros [96] manifiestan que la composición dinámica de servicios es una de las tecnologías clave que facilitará el desarrollo de aplicaciones para la computación ubicua ya que su aplicabilidad es muy útil en entornos donde el número de componentes, servicios y usuarios disponibles es variable a lo largo del tiempo.

### 1.3.1 Módulo de composición de servicios

Los enfoques de composición pueden ser agrupados en función del grado de participación que tiene el usuario en el mismo, pero además de este factor la composición puede ser diferenciada en dos aspectos importantes, por una parte la síntesis y por otra la orquestación (término que no consideramos adecuado, consideramos más adecuado el término ejecución) [91]:

- La síntesis se refiere a la generación del plan que tiene como objetivo realizar el comportamiento deseado combinando múltiples servicios, es decir, se refiere al empleo de técnicas para crear la composición que posteriormente será ejecutada.

- La ejecución se refiere a la coordinación del control y los flujos de datos a lo largo de los diversos componentes cuando se está ejecutando el plan previamente definido, es decir, se refiere a las técnicas o enfoques empleados para ejecutar el plan.

En los últimos años se han desarrollado muchos proyectos y enfoques que tienen como objetivo la composición de servicios tanto semánticos como no semánticos. Consideramos que en vez de enumerar todos los enfoques es mejor describir cuáles son las características o tipos de enfoques que se han llevado a cabo describiéndolos en base a tres aspectos de composición (participación del usuario, síntesis y ejecución). A continuación se describirá en qué consiste cada uno de los aspectos.

#### 1.3.1.1 Enfoques de composición en base a la participación del usuario

La composición de servicios viene derivada de modelos empresariales [97] y puede ser categorizada en dos tipos: la composición estática (proactiva) de servicios y la composición dinámica (reactiva) de servicios [98, 99]:

- La estática es un enfoque en el que los diseñadores de aplicaciones implementan una nueva aplicación manualmente diseñándola mediante herramientas de workflow o diagramas de estado, describiendo los patrones de interacción entre los componentes del mismo. Estas aplicaciones deben de ser manualmente diseñadas antes de desplegarlas, es decir, la composición de servicios se hace en tiempo de diseño. Este tipo de composición es adecuado para aplicaciones de tipo B2B, en donde la interacción entre los componentes es compleja pero estática [96]. Este tipo de composición es la empleada por la industria.

- En cambio en la composición dinámica de servicios el modelo de proceso se genera automáticamente cuando un usuario lo precise. La composición dinámica es más aconsejable para entornos ubicuos o móviles, donde el número de componentes varía a lo largo del tiempo [96]. Al contrario que en el caso de la composición estática, la composición de servicios se realiza en tiempo de ejecución. Este tipo de composición ha sido planteado por grupos relacionados con la Web Semántica.

- Existe otro enfoque intermedio denominado composición semi-automática o interactiva que proponen métodos y herramientas para la interacción del usuario que capturen las intenciones de los usuarios de manera interactiva y continua durante el proceso de composición del plan a ejecutar [100].

#### 1.3.1.2 Enfoques para la síntesis de la composición

En los últimos años se han publicado diversos artículos que tienen como objetivo realizar una clasificación de las diferentes técnicas de composición, centradas en la síntesis del plan a ejecutar. A continuación se van a describir cuales son las diferentes agrupaciones que realizan los autores.

Rao y Su [101] realizan una revisión bibliográfica centrándose en las técnicas

empleadas para crear la composición identificando dos técnicas principales:

- Técnicas de workflow: Esta técnica tiene como objetivo crear un workflow para su posterior ejecución. Esta técnica puede ser dividida en dos, la estática en donde el peticionario además del modelo abstracto de proceso define también las tareas y las dependencias de datos entre las mismas, en este caso solamente la selección y el binding es realizado automáticamente por el programa; o bien la dinámica en donde se crea dinámicamente el modelo de proceso abstracto y se seleccionan dinámicamente los servicios, pero definiendo como entradas una serie de restricciones además de las preferencias y de las dependencias de los servicios atómicos.

- Técnicas de planificación de IA: La composición de servicios basada en planificación consiste en emplear un algoritmo que permita resolver un problema de planificación. De esta manera partiendo de un estado inicial se determinan los pasos a realizar para llegar a un estado final o meta, pero para ello es necesario realizar una representación de los estados posibles del entorno, incluyendo el estado inicial y el final, así como las operaciones permitidas en cada uno de ellos para realizar el cambio de estado. Los autores del artículo además dividen este tipo de técnicas en 5 subgrupos, como son las de cálculo situacional, planificación basada en reglas, PDDL (Planning Domain Definition Language), prueba de teoremas y otras (HTN, etc.). Algunas de estas técnicas serán explicadas en el punto 1.3.1.3.

Peer [102] en cambio solamente se centra en las técnicas de planning de IA empleadas para la composición de Servicios Web, tras el análisis de los diferentes enfoques realiza una comparativa de los planificadores más comunes evaluándolos en base a unos criterios definidos en el artículo. Del mismo modo, May Chan y otros [103] realizan un análisis centrándose solamente en las técnicas de planning de inteligencia artificial y para ello se basan en la categorización de técnicas realizada por Ghallab y otros [104] sin profundizar en las técnicas de neo-planning y en sus extensiones, ya que no son adecuadas para entornos de Servicios Web. Del informe se extrae como conclusión que la planificación basada en HTN puede ser considerada como la más completa y adecuada para la composición automática de Servicios Web.

Existe otro análisis de Küster y otros [91] en el que la clasificación no es realizada en base a la técnica empleada para la composición, sino en los diferentes tipos de

aplicaciones de composición de servicios, es decir, realizan una clasificación en base al problema que se tiene entre manos a la hora de la composición, para ello identifican 3 clases de problemas que a su vez están divididos en varios grupos: Cumpliendo las precondiciones: El servicio que ofrece el efecto deseado existe, sin embargo, no todas sus precondiciones son cumplidas. Generando múltiples efectos: La petición de servicio dispone de múltiples efectos que están relacionados, pero no pueden ser realizados por un único servicio. Superando la carencia de conocimiento: Es necesaria disponer de conocimiento adicional para satisfacer correctamente una petición.

Alamri y otros [105] presentan otro análisis que agrupa los enfoques de composición en 6 tipos diferentes: reconfiguración en tiempo de ejecución empleando wrappers, adaptación de servicios en tiempo de ejecución, lenguajes de composición, técnicas de composición basadas en workflows, técnicas de composición basadas en ontologías y técnicas de composición declarativa. Tras describir cada uno de los tipos se detalla una tabla en donde se describen las ventajas y las limitaciones de cada uno de los grupos.

Por último Ter Beek [106-108] y otros realizan una clasificación basada en los lenguajes existentes para composición. Por una parte describen los lenguajes para composición estática como pueden ser BPEL para orquestación y WS-CDL para coreografía y por otra parte los lenguajes dinámicos (Servicios Web semánticos) destacando OWL-S y WSMO.

#### 1.3.1.2.1 Modelos de composición

Basándose en la expresividad del lenguaje subyacente para la especificación de tareas de usuario y los servicios del entorno es posible describir otro enfoque de agrupación, que se cree que es más adecuado de cara al presente proyecto de investigación y que son descritos a continuación en seis modelos de composición:

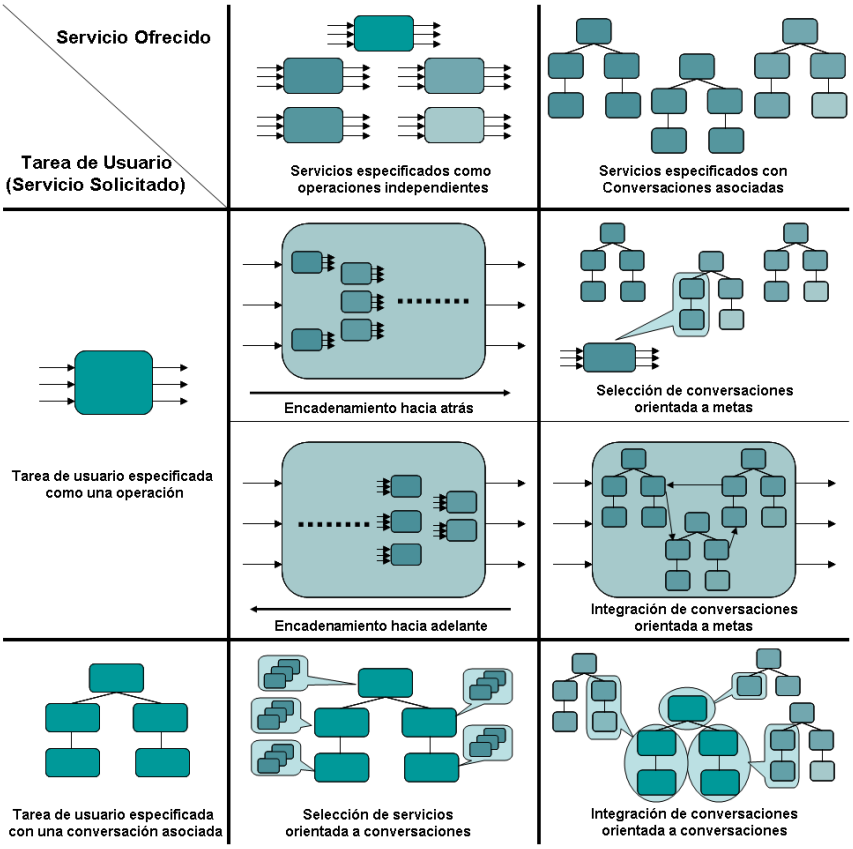


**Figura 1.1: Modelos de Composición**

Los primeros dos modelos de composición denominados encadenamiento hacia atrás y encadenamiento hacia adelante son empleados cuando los servicios de la red y la tarea del usuario están descritas mediante operaciones individuales, sin tener asociadas conversaciones. En estos modelos de composición las operaciones de los servicios del entorno son combinadas basándose en el emparejamiento entre signaturas. El objetivo de esta combinación es obtener un servicio compuesto que sea capaz de realizar la tarea del usuario en términos de especificación de signaturas. El encadenamiento hacia adelante consiste en ir seleccionando los servicios empezando por las entradas del servicio requerido (y precondiciones) y continuar así hasta que la salida (y efectos) del servicio requerido son generados [109]. Por el contrario el método de encadenamiento hacia atrás comenzaría por las salidas (y efectos) e iría generando el flujo hasta llegar a las entradas (y precondiciones) [110]. Aunque estos enfoques permitan combinar servicios sin conocimiento previo en el modelo de combinación (por ejemplo en qué orden ser van a encadenar los servicios), su complejidad es muy alta, ya que todas las posibilidades de encadenamiento tiene que ser investigadas, por lo que el coste computacional puede ser muy elevado. Además, como el proceso es ciego (las operaciones son encadenadas solo en base a la

compatibilidad de las signaturas), existe cierta incertidumbre en lo relativo a como es manipulada la información del usuario. Sin embargo, existen algunos enfoques que asumen un conocimiento previo descrito en forma de reglas de descomposición, con el objetivo de orientar el proceso de encadenamiento [111].

Los tercer y cuarto modelos de composición denominados selección de conversaciones orientada a metas e integración de conversaciones orientada a metas, investigados respectivamente en [112] y en [113] que está basado en [114], asumen que los servicios del entorno están descritos mediante conversaciones y las tareas de usuarios están descritas mediante operaciones individuales. En el primero de los modelos el denominado Process Query Language es empleado para encontrar aquellas conversaciones que contienen fragmentos que satisfacen la tarea del usuario requerida. Por lo tanto, se asume implícitamente que la tarea del usuario puede ser realizada por un único servicio al contrario que el segundo modelo de composición, que tiene como objetivo integrar un conjunto de conversaciones de servicios para realizar la tarea del usuario (que está descrita como una operación simple). En el segundo modelo de composición las conversaciones de un conjunto de servicios seleccionados son integradas para dar como resultado una composición que verifica ciertas propiedades como puede ser la ausencia de bloqueos además de conformar la tarea del usuario consumiendo todas sus entradas y generando todas las salidas requeridas. En los dos modelos de composición sigue existiendo un nivel de incertidumbre debido a la manera de combinar los servicios de la red. De hecho, verificando que la composición resultante está libre de bloqueos no quiere decir que los datos del usuario no hayan sido transformados de manera inadecuada empleando operaciones no apropiadas (por ejemplo empleando operaciones que un usuario no hubiera empleado para cumplir el objetivo) para cumplir con la especificación de la tarea del usuario.

El quinto modelo de composición, denominado selección de servicios orientada a conversaciones, donde la tarea a realizar está descrita como una conversación y los servicios del entorno son descritos mediante conjuntos de operaciones independientes. Este modelo ha sido muy empleado en entornos internet [115, 116] y sobre todo en la composición dinámica de servicios en entornos de computación ubicua [117, 118]. En este modelo, las operaciones de los servicios ofrecidos son emparejadas contra las operaciones requeridas en la tarea del usuario. Los diferentes enfoques que emplean este modelo se diferencian en la expresividad del lenguaje

empleado para la descripción de servicios además del algoritmo de emparejamiento. Como este enfoque es capaz de satisfacer la conversación descrita en la tarea del usuario consigue que la composición resultante sea más ajustada a las necesidades del usuario todo ello siendo el flujo de información controlado, ya que es el descrito por el usuario.

El último modelo de composición, denominado integración de conversaciones orientada a conversaciones, donde tanto los servicios del entorno como la tarea del usuario están descritos mediante conversiones o comportamientos complejos. En este modelo, investigado en [119], las conversaciones de los servicios de red son integradas con el objetivo de satisfacer y cumplir la conversación de la tarea requerida por el usuario. Este modelo naturalmente incluye al modelo anteriormente descrito, ya que en caso de que existan servicios de red descritos de manera simple (mediante operaciones) el algoritmo los usará en combinación con las conversaciones de otros servicios para construir la tarea del usuario. Además del trabajo de Berardi en este mismo enfoque se encuentra también el trabajo que está realizando Ben Mokhtar en INRIA Roquencourt [120, 121].

#### 1.3.1.3 Técnicas de planificación de IA

En este punto se van a explicar algunas de las técnicas de planificación de IA comentadas anteriormente.

##### 1.3.1.3.1 Planificación HTN

La planificación HTN (Hierarchical Task Network) [122] permite descomponer una tarea compleja en un conjunto de tareas más simples. Esta planificación define el concepto método. Un método define las condiciones que se deben cumplir por una tarea para poderla descomponer en un conjunto de sub-tareas. Las tareas que pueden ser descompuestas se llaman tareas compuestas. En cambio, las tareas que no pueden ser descompuestas se llaman tareas primitivas. Para poder ejecutar la planificación HTN se debe crear una teoría de dominio. Esta teoría consiste en un conjunto de métodos y operadores que habilitan la creación de planes. Un método establece la relación entre una tarea compleja, sus precondiciones y las sub-tareas en las que puede descomponerse. Cuando el algoritmo tiene que descomponer una tarea, las precondiciones son comprobadas para probar que pueden ser aplicadas al

estado actual del entorno.

Esta planificación es utilizada en [123] junto con un razonamiento LCW [124]. El uso de LCW permite evitar la redundancia en el acceso a la información durante la composición de un servicio. La base de conocimiento almacena en conocimiento que el necesita para empezar la planificación y la información que es producida en el proceso. Cuando un agente, que utiliza planificación HTN, quiere descomponer una tarea primero debe comprobar sus precondiciones. El uso de razonamiento LCW redice el número de peticiones a la base de conocimiento gracias al uso de la suposición de que la información está completa.

El planificador tiene un conjunto de acciones especiales que permiten obtener información durante el proceso de planificación. Esta información no está incluida en el plan pero es obtenida por el planificador usando un mediador semántico. La adicción de este mediador semántico reduce el tiempo de planificación debido a que el número de acciones que el planificador tiene que realizar para obtener la solución es reducido. Esta solución utiliza OWL-S para realizar la descripción de las acciones del entorno.

Una solución similar puede ser encontrada en [125]. Este trabajo propone el uso de un planificador HTN llamado SHOP2 (Simple Hierarchical Order Planner) que ha sido seleccionado porque planifica las acciones de una tarea en el mismo orden que la ejecución. Esto permite conocer el estado del entorno en cualquier momento del proceso de la planificación y ofrece la posibilidad de incluir razonamiento e inferencia durante el proceso de evaluación de precondiciones.

#### *1.3.1.3.1.1 Planificación utilizando un lenguaje de programación lógica*

Otras propuestas para la composición de servicios usan lenguajes de programación lógica. La composición es realizada mediante un conjunto de procedimientos reutilizables descritos este tipo de lenguajes.

Un ejemplo de aproximación a esta es mostrado en [126]. donde los autores proponen la utilización de GOLOG [127]. GOLOG es un lenguaje para agentes inteligentes cuyo objetivo es el razonamiento de las acciones y los cambios del entorno. Este lenguaje fue desarrollado sin tener en cuenta las tareas relacionadas con la recolección de información. Aun así, los autores del trabajo consideran que estas operaciones son necesarias para la composición de servicios, es más, han propuesto una extensión de

GOLOG que las incluye.

El fin de esta solución es conseguir crear procedimientos reusables que puedan ser compartidos y utilizados por todos los usuarios del entorno. Los autores proponen compartir estos procedimientos dentro de la descripción de servicios en OWL-S. Cuando un agente recupera uno de estos procedimientos genéricos, este lo debe amoldar a las necesidades de los usuarios teniendo en cuenta el estado y de los servicios que están disponibles en el entorno. Cuando el procedimiento esta creado y personalizado este podrá ser ejecutado. En este punto existe una necesidad de obtener la información del entorno mediante la evaluación de precondiciones y de los efectos de la composición del servicio.

Por otro lado, en el momento de ejecución del servicio es donde aparecen la mayoría de los problemas. Los autores de esta proposición utilizan una solución que mezcla un intérprete online y otro offline. Los intérpretes online son los que realizan la adquisición de la información durante el proceso de razonamiento. En cambio, los intérpretes offline únicamente utilizan la información de la que disponen en su base de conocimiento. El intérprete propuesto obtiene la información del entorno cuando es necesitada; aunque, aquellas acciones que afectan al entorno son simuladas y únicamente son ejecutadas en el plan final. Esta solución reduce el espacio de búsqueda cuando se razona y mantiene la posibilidad de obtener información mientras se ejecuta y razona el plan.

La principal desventaja de esta solución es que el plan está pensado para ser inalterable, por lo que, los cambios que se produzcan en el entorno pueden afectar a la validez de este y pueden convertirlo en inutilizable.

*1.3.1.3.1.2 Planificación utilizando un Motor de Reglas*

Otra solución para la composición de servicios puede ser encontrada en [128]. En este caso, se utiliza una herramienta para la construcción de reglas basada en sistemas expertos. La herramienta utilizada es JESS [129] la cual utiliza el algoritmo RETE [130] para realizar la unión entre reglas y hecho de la base de conocimiento. Este algoritmo es mucho más rápido que otros basados en sentencias if-else en un bucle.

Los autores identifican los pasos que son necesarios para realizar la composición de servicios usando un motor de inferencia de este tipo. El primer paso es la realización

de la traducción entre descripciones semánticas y las tripletas verbo-sujeto-objeto utilizadas por JESS. Las tripletas generadas en este paso son introducidas en la base de conocimiento como nuevos hechos. El siguiente paso es la construcción de operadores que correspondan con servicios atómicos. Estos operadores son esenciales para el uso del algoritmo RETE.

La composición de servicios es un proceso iterativo formado por dos pasos: el primero, búsqueda de aquellos servicios que satisfagan el objetivo existente y añadirlos a la composición; el segundo, conversión de las entradas y las precondiciones de todos los servicios añadidos a la composición. Este proceso será repetido mientras sigan existiendo más servicios. La composición será considerada exitosa cuando únicamente los objetivos que corresponden a entradas y precondiciones externas no han sido resueltos. Las entradas y precondiciones externas son aquellas que pueden ser resueltas por el usuario o agente utilizando la información disponible. En cambio, si cuando la composición termina existen objetivos que dependen de precondiciones y/o entradas consideradas internas, aquellas que solo pueden ser resueltas por otros servicios, se considerará que la composición está incompleta y no podrá ser ejecutada.

La solución para la composición de servicios da por hecho que todas las descripciones de servicios han sido obtenidas y almacenadas en la base de conocimiento. Sin embargo, puede ser necesario el descubrimiento de nuevos servicios durante el proceso de planificación [131]. Una posible solución consiste en la división del proceso de composición en diferentes partes. Inicialmente se deberían descubrir los servicios teniendo en cuenta sus entradas y salidas. Los dispositivos semánticos encontrados son unidos utilizado encaminamiento hacia delante (salidas unidas con entradas) o mediante encadenamiento hacia atrás (entradas unidas con salidas). Los grafos que son producidos como resultados del el encadenamiento hacia adelante empiezan desde las entradas provistas por el usuario, mientras que, en los grafos formados mediante el encadenamiento hacia atrás la composición será iniciada desde las salidas que desea el usuario. Además, como resultado del encadenamiento hacia adelante un conjunto de grafos parciales serán construidos, que serán utilizados para crear el grafo final utilizando técnicas de planificación de IA.

#### 1.3.1.4 Enfoques para la ejecución de la composición

Actualmente existen dos enfoques principales para la ejecución de composiciones de servicios: Por una parte está la orquestación y por otra la coreografía. Estos conceptos definen en cierta medida cómo se establecen las relaciones entre los diferentes servicios para obtener un comportamiento coordinado de los mismos. Las diferencias se pueden resumir del siguiente modo:

#### 1.3.1.5 Orquestación

Un proceso se puede considerar una orquestación de servicios cuando es controlado totalmente por una única entidad. Este proceso define completamente las interacciones con los servicios componentes y la lógica requerida para conducir correctamente esas interacciones (ver Ilustración 22). Este tipo de proceso puede entenderse como privado y ejecutable ya que sólo la entidad que está orquestando el proceso conoce el flujo de control e información que sigue el proceso que se está orquestando. De esta manera se crea un proceso que utiliza diferentes servicios manipulando la información que fluye entre ellos, convirtiendo, por ejemplo, los datos de salida de algunos servicios en datos de entrada de otro. Aquí, cada entidad que participa implementa y controla su propio proceso [132, 133].


**Ilustración 22. Orquestación de Servicios Web [132]**

La descripción anterior concuerda con una orquestación centralizada en la que una entidad es la que controla al resto y ordena cuando deben ejecutarse, pasando todo el flujo de información por él. Pero, en los últimos años se ha propuesto otro tipo de orquestación, la denominada orquestación distribuida o descentralizada [134, 135] que en vez de disponer de un nodo central o director que orquesta al resto de servicios se pretende dividir el workflow en diferentes partes para así ejecutarlo de manera distribuida. Algunas de las ventajas de la orquestación descentralizada son las siguientes:

- No hay coordinación central que puede llegar a ser el cuello de botella del sistema

- Distribuyendo los datos se reduce el tráfico de red y se mejora el tiempo de transferencia.

- Distribuyendo el control se mejora la concurrencia.

#### 1.3.1.5.1 Coreografía

Un proceso es una coreografía de servicios cuando define las colaboraciones entre cualquier tipo de aplicaciones componentes, independientemente del lenguaje o plataforma en el que estén definidas las mismas (ver Ilustración 23). Un proceso de coreografía no es controlado por uno solo de los participantes. A diferencia de la orquestación, la coreografía puede verse como un proceso público y no ejecutable. Público porque define un comportamiento común que todas las entidades participantes deben conocer y no ejecutable porque está pensado para verse más bien como un protocolo de negocio que dicta las reglas para que dichas entidades puedan interactuar entre sí [132, 133].

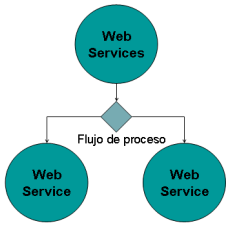


**Ilustración 23. Coreografía de Servicios Web [132]**

### 1.3.2 Análisis de los principales enfoques de composición de servicios en entornos de computación ubicua

En las siguientes páginas se describen los principales enfoques de composición en computación ubicua, pero previamente se establecen las características a analizar en cada uno de ellos.

#### 1.3.2.1 Criterios para la evaluación de enfoque de composición para entornos de computación ubicua

A continuación son descritos los cuatro grupos de características que se han definido para la clasificación de los enfoques existentes, así como los resultados de las comparativas de los mismos:

##### 1.3.2.1.1 Expresividad del Lenguaje

En este grupo se encuentran aquellas características relacionadas con la expresividad del lenguaje empleado para describir los servicios así como las tareas definidas por el usuario (ver Tabla 16):

**Lenguaje de modelado de servicios (mod):** Lenguaje empleado para describir las características funcionales como no-funcionales de los servicios.

**Semántica (sem):** Esta característica indica si el lenguaje empleado para describir los servicios ofrece soporte semántico para describir tanto los atributos funcionales como no-funcionales de los servicios para su posterior empleo en las fases de descubrimiento y composición.

**Mecanismo para representar la tarea del usuario (task):** Este criterio representa el mecanismo empleado para describir el servicio o tarea requerido por el usuario. Por ejemplo, este puede ser descrito como una operación, una conversación, como la meta a conseguir, etc.

**Mecanismo para representar los servicios publicados (pub):** Este criterio representa el mecanismo empleado para describir el servicio publicado. Por ejemplo, operaciones, conversaciones, etc.

**Tabla 16. Resultados de la comparativa del grupo de expresividad del Lenguaje.**

| Approach | mod | sem | task | pub |
|---|---|---|---|---|
| [100, 136] | WSDL, UPnP | Syntactic | Workflow | Simple Services |
| [137] | OWL-S | Semantic | Composite FSM | FSM adaptive behaviour |
| [138] | WSDL | Syntactic | Workflow | Simple Services |

| Approach | mod | sem | task | pub |
|---|---|---|---|---|
| [139] | WSDL | Syntactic | Context information + Goal | Simple Services |
| [140] | Key-Value pairs | Syntactic | Workflow | Simple Services |
| [141] | OWL-SC | Semantic | Simple Service | Simple Services |
| [142] | Key-Value pairs | Syntactic | Workflow | Simple Services |
| [143] | OWL-S | Semantic | Workflow | Simple Services |
| [144] | OWL-S | Semantic | Workflow | Simple Services |
| [123] | OWL-S | Semantic | Workflow | Simple Services |
| [145] | UPnP | Syntactic | Workflow | Simple Services |
| [146] | OWL-S | Semantic | Workflow (3rd party service providers) | Simple Services |
| [147] | Not specified | Semantic | Simple Service | Simple Services |
| [148] | WSDL | Syntactic | Workflow (State Chart Diagrams) | Simple Services |
| [149] | WSDL | Syntactic | Message Sequence Charts | Simple Services |
| [150] | WSDL | Syntactic | Simple Services | Simple Services |
| [151] | OWL-S | Semantic | Workflow | Simple Services |
| [121, 152] | COCOA-L (OWL-S) | Semantic | Workflow | Workflow |
| [98, 153] | DAML-S | Semantic | Workflow (Description-level Service Flow) | Simple Services |
| [117, 154] | OWL-S | Semantic | Workflow | Simple Services |
| [155, 156] | OWL-S | Semantic | Goal (desired state) | Simple Services |
| [157] | DAML+OIL | Syntactic | Goal | Simple Services |
| [118] | WSDL | Syntactic | Goal (converted to a Workflow) | Simple Services |
| [94, 158] | OWL-S | Semantic | Goal (converted to a abstract plan) | Simple Services |

#### 1.3.2.1.2 Modelo de Composición

En este grupo se encuentran aquellas características relativas al método empleado para la creación de la composición o plan (síntesis) para su posterior ejecución (ver Tabla 17):

**Técnica de creación de la composición (tech):** Técnica empleada para conseguir el objetivo/meta definido por el usuario. Por ejemplo, técnicas de planificación de IA, de Workflow, de Data-Mining, etc.

**Lenguaje de composición (lang):** Lenguaje empleado para representar y ejecutar la composición que previamente ha sido creada mediante la técnica de composición. Por ejemplo, scripts, bpel4ws, etc.

**Sensibilidad al contexto (cont):** Indica si la técnica de composición tiene en cuenta la información contextual (por ejemplo: localización del usuario, perfil del usuario, localización de los dispositivos, etc.) del entorno durante el proceso de creación del plan.

**Sensibilidad a la calidad de servicio (QoS):** Indica si la técnica de composición tiene en cuenta la calidad de servicio (por ejemplo: latencia, memoria utilizada, etc.) durante la creación del plan.

**Intervención del usuario (user):** Indica si el usuario puede participar durante el proceso de creación de la composición, por ejemplo seleccionando los servicios en caso de conflicto, etc.

**Tabla 17. Resultados de la comparativa del grupo de Modelo de Composición.**

| Approach | tech | lang | cont | QoS | user |
|---|---|---|---|---|---|
| [100, 136] | AI Planning - Depth First Search 1 | Scripts | No | No | Yes |
| [137] | AI Planning - Finite State Machines | OWL-S Proccess | No | No | No |
| [138] | ECF+Mobile Agents | Executable Choreography Lang. (ECL) | Yes | No | No |
| [139] | AI Planning - HTN - SHOP2 | BPEL4WS | Yes | No | No |
| [140] | Workflow (Service Selection) | Not specified | Yes | No | No |
| [141] | AI Planning - HTN | Directed Acyclic Graph (DAG) | Yes | Yes | No |
| [142] | Workflow (Service Selection) | XML | Yes | Yes | No |

| Approach | tech | lang | cont | QoS | user |
|---|---|---|---|---|---|
| [143] | Workflow (Service Selection) | Not specified | Yes | No | Yes |
| [144] | Workflow (Service Selection) | XML | No | Yes | No |
| [123] | AI Planning - HTN + LCW | Not specified | Yes | No | No |
| [145] | Workflow | Functional Task Description (FTD) | Yes | Yes | Yes |
| [146] | Workflow (Service Selection) | OWL-S Proccess | Yes | No | No |
| [147] | Data-Mining | Not specified | Yes | No | No |
| [148] | Agent Conversation based | State Chart Diagrams | Yes | Yes | No |
| [149] | Workflow (Service Selection) | BPEL4WS | Yes | Yes | No |
| [150] | Ubiquitous Coordination Model | Not specified | Yes | Yes | No |
| [151] | AI Planning - Backward chainning - STRIPS | OWL-S Proccess | Yes | No | No |
| [121, 152] | Workflow - Finite State Automatas | COCOA-L (OWL-S) | Yes | Yes | Yes |
| [98, 153] | Workflow (Service Selection) | Description-level Ser. Flow (DAML-S) | No | No | No |
| [117, 154] | Workflow (Service Selection) | Not specified | Yes | No | Yes |
| [155, 156] | AI Planning - STRIPS | Not specified | Yes | No | Yes |
| [157] | AI Planning - Blackbox planner - STRIPS | Lisp | Yes | No | Yes |

| Approach | tech | lang | cont | QoS | user |
|---|---|---|---|---|---|
| [118] | Workflow (Service Selection) | BPEL4WS | Yes | No | Yes |
| [94, 158] | Worfklow + AI Planning - CSP (for validation) | OWL-S Proccess | Yes | No | No |

#### 1.3.2.1.3 Modelo de Ejecución

En este grupo son descritas aquellas características relacionadas con el mecanismo empleado para la ejecución del plan creado en el grupo anterior (ver Tabla 18):

**Orquestación/Coreografía (comp):** Este criterio especifica si la composición es ejecutada de manera centralizada (orquestación) o bien de manera distribuida (coreografía)

**Replanificación en la ejecución (repla):** Este criterio especifica si el mecanismo de ejecución soporta la replanificación (por ejemplo debido a un cambio contextual) de la composición durante la ejecución de la misma.

**Binding dinámico o estático (bind):** Este criterio especifica si los servicios empleados en la composición están definidos de manera estática o bien dinámica, es decir, si los servicios de la composición están definidos previamente o pueden variar.

**Tabla 18. Resultados de la comparativa del grupo de Modelo de Ejecución.**

| Approach | comp | repla | bind |
|---|---|---|---|
| [100, 136] | Orchestration | No | Dynamic |
| [137] | Orchestration | Yes | Dynamic |
| [138] | Choreography | No | Dynamic |
| [139] | Orquestration | No | Dynamic |
| [140] | Orchestration | No | Dynamic |
| [141] | Orchestration | No | Dynamic |
| [142] | Orchestration | Yes | Dynamic |
| [143] | Orchestration | No | Dynamic |
| [144] | Orchestration | No | Dynamic |
| [123] | Orchestration | No | Dynamic |
| [145] | Orchestration | No | Dynamic |
| [146] | Orchestration | No | Dynamic |
| [147] | Orchestration | No | Dynamic |

| Approach | comp | repla | bind |
|---|---|---|---|
| [148] | Choreography (Agent based) | No | Dynamic |
| [149] | Orchestration | Yes | Dynamic |
| [150] | Orchestration | No | Dynamic |
| [151] | Orchestration | No | Dynamic |
| [121, 152] | Orchestration | No | Dynamic |
| [98, 153] | Distributed Orchestration | Yes | Dynamic |
| [117, 154] | Orchestration | No | Dynamic |
| [155, 156] | Orchestration | No | Dynamic |
| [157] | Orchestration | Yes | Dynamic |
| [118] | Orchestration | No | Dynamic |
| [94, 158] | Orchestration | No | Dynamic |

#### 1.3.2.1.4 Entorno de Ejecución

En este grupo son descritas aquellas características relacionadas con el entorno de ejecución en el que operan los servicios, se realiza la composición y es ejecutada la misma (ver Tabla 19):

**Arquitectura (arch):** Este criterio especifica la arquitectura empleada en el enfoque de composición. Ésta puede estar basada en una arquitectura existente o puede ser una nueva.

**Mecanismo de descubrimiento de servicios (disc):** Tecnología o protocolo empleado para el descubrimiento de los servicios publicados y para la fase de creación de la síntesis de proceso.

**Dispositivos que soportan la ejecución de la composición (device):** Este parámetro representa que tipo de dispositivos son los que soportan la ejecución de la composición.

**Tabla 19. Resultados de la comparativa del grupo Entorno de Ejecución**

| Approach | arch | disc | device |
|---|---|---|---|
| [100, 136] | End-User Programming Framework (EUP) | UPnP, WS, Jini | Mobile Phones |
| [137] | Not specified | Not specified | Not specified |

| Approach | arch | disc | device |
|---|---|---|---|
| [138] | Executable Choreography Framework (SOAP) | Not specified | Embedded Systems |
| [139] | SOAP | UDDI | Not specified |
| [140] | IPOJO over OSGI | OSGI | Gateway (Home) |
| [141] | SOAP | Not specified | Not specified |
| [142] | Smart Space Middleware over OSGI | OSGI | Gateway |
| [143] | Jini | Reggie (Jini lookup service) | Not specified |
| [144] | Home Appliances Integration Unit (HAIU) | Not specified | Home entertainment syst. |
| [123] | OntoPlan | Not specified | Not specified |
| [145] | UPnP | UPnP | Resource Constrained devi. |
| [146] | OSGI (UPnP & Jini) | OSGI | Gateway (Home) |
| [147] | Service Provisioning Middleware (COSEP) | Not specified | Not specified |
| [148] | SOAP | UDDI | Not specified |
| [149] | SOAP | Not specified | Not specified |
| [150] | CB-Sec Framework | LW-UDDI | Embedded PCs |
| [151] | Multi-agent architecture and SOAP | Not specified | Not specified |
| [121, 152] | SOAP | Not specified | Not specified |
| [98, 153] | Architecture developed for the approach | Group Based Service Disc. | Mobile Devices |
| [117, 154] | Task Computing Environment (SOAP) | UPnP | PDA |
| [155, 156] | SOAP | WS-Discovery, UPnP | Mobile Devices |
| [157] | GAIA | Discovery module of GAIA | Not specified |
| [118] | SOAP | UDDI, Multicast DNS | iPaq |
| [94, 158] | SOAP | Not specified | Not specified |

#### 1.3.2.2 Análisis de los enfoques de composición

Tomando como base los resultados de la comparativa a continuación se realiza un breve resumen destacando las características más importantes de los enfoques planteados.

Propuesto en [117, 154], Task Computing orienta la composición de servicios hacia lo que quiere hacer el usuario, en lugar de hacer que el usuario se adecue a los servicios del entorno. De esta manera se ofrece un entorno completo orientado al usuario final basado en descripciones DAML-S. También Ni [155, 156] pretende abstraer al usuario de la complejidad en la composición por medio de OSOA defendiendo, como muchos otros autores [157], técnicas de AI planning para resolver el problema de la composición de servicios. En la propuesta presentada en [100, 136] por Nokia Research Center Cambridge, se propone una arquitectura en la cual se deja en manos de la intuición del usuario final la elección de los dispositivos que se quieren utilizar en la composición y se utiliza el algoritmo de composición DSF1 para buscar las posibles combinaciones entre los dispositivos seleccionados. Se trata de un sistema de composición semi-automático puesto que cuenta con la intervención del usuario durante la composición.

En [145] el objetivo se define como end-to-end, ej."link (video-source) with (display)" y el sistema busca los servicios apropiados para unir los dos extremos con un algoritmo de búsqueda simple. Entre las posibles soluciones se elige la más adecuada basándose en policies y en la disponibilidad de los recursos. Vukovic y Robinson [139], plantean la composición como un problema de planificación usando HTN con SHOP2. La descripción de la composición se define usando BPEL4WS. Esta descripción se traduce a objetivos SHOP2 y el planificador resuelve un plan para estos objetivos. Finalmente, este plan es traducido de nuevo a BPEL4WS. Cambios en el contexto pueden provocar la replanificación de la composición durante su ejecución, haciendo que la aplicación se adapte a estos cambios contextuales, el mismo objetivo perseguido en [150].

Qiu y otros proponen en [141] utilizar OWL-SC, una extensión de OWL-S para incluir información contextual en la descripción del servicio, de manera similar a la propuesta en [159]. También proponen HTN como método de planificación, pero introducen la idea de "librería de planes" para poder almacenar planes y no tener que recalcular más de una vez un plan determinado. Ontoplan [123] también emplea HTN, aunque lo combina con Local Closed World reasoning (LCW) para evitar redundancias en la composición.

El uso de workflows como herramienta para componer servicios también es defendido por muchos autores como Ranganathan y McFaddin [118], Bellur y Narendra [149], etc. Por ejemplo Ben Mokhtar y otros [120, 152] proponen el sistema COCOA donde

las tareas de los usuarios están descritas mediante procesos abstractos OWL-S en donde la composición final se consigue mediante un algoritmo de emparejamiento que construye la composición en base a fragmentos de las conversaciones ofrecidas por los servicios utilizando Finite State Machines (FSM) para modelizarlas. Vallee y otros [94, 158] y Maamar y otros [148] presentan un enfoque que combina técnicas de sistemas multi-agente con Servicios Web Semánticos para proporcionar composición dinámica de servicios sensible al contexto. En [147], se emplean técnicas de data-mining considerando información de contexto y el historial de uso de los servicios, para definir nuevas composiciones de servicios.

La representación de los servicios por medio de ontologías es defendida también en [143, 144, 151]. Otros como Lee y otros en [160], emplean OSGi como especificación de los servicios de forma que una composición expresada en forma de grafo y representada en formato XML, se construye como un nuevo servicio OSGi, utilizando servicios existentes. También en [140] y en [146] se emplea OSGi, proponiendo como novedad en este último dejar en manos de terceros más expertos, en lugar del usuario, la composición de los servicios.

A diferencia de las propuestas estudiadas hasta ahora, en la que la responsabilidad de la composición era centralizada, en cambio, Chakraborty y otros [98, 153] proponen protocolos descentralizados para componer servicios expresados en DAML-S. De forma similar, la gran mayora de los trabajos analizados utilizan orquestación centralizada para la ejecución de la composición en la que un coordinador es el encargado de dirigir el control del flujo.

Existen varios trabajos que describen algoritmos para la composición que utilizan descripciones semánticas de los servicios destacan los siguientes: En [137] Carey y otros proponen extender la planificación IA, criticada por necesitar replanificación para adaptarse a los cambios en el contexto, por medio de máquinas de estados finitos (FSM) que modelizan el comportamiento adaptativo (no funcional) que debe tener la composición resultante, de forma que estas FSM se encargan de adaptarse a los cambios en el contexto sin necesidad de replanificación. sEl ECL (Executable Choreography Language) descrito en [138] permite inyectar modificaciones en el flujo de control de un servicio sobre el ECF (Executable Choreograhy Framework) de forma no invasiva, esto es, sin alterar manualmente el flujo original del servicio.

### 1.3.3 Conclusiones

Del análisis de los sistemas para entornos ubicuos anteriormente mencionados y de los diferentes métodos para composición expuestos se pueden extraer las siguientes conclusiones:

- Los sistemas o métodos de workflow son mayoritariamente empleados en los casos en donde el peticionario ya tiene definido un modelo de proceso, pero en donde es necesario que un programa encuentre automáticamente los servicios atómicos para rellenar dicho proceso. Y en cambio las técnicas de planificación IA son empleadas cuando el peticionario no tiene un modelo de proceso pero se tiene un conjunto de preferencias y restricciones, de esta manera el planificador basándose en dichas preferencias y restricciones realiza la planificación. De estos dos métodos es difícil decir cual es más adecuado a un entorno inteligente, ya que puede variar en función del tipo de sistema que se desee desarrollar y en función de los casos de uso a emplear.

- Casi en la práctica totalidad de enfoques estudiados en el presente artículo la ejecución de la composición es centralizada es decir, la composición la realiza un dispositivo y no es una composición distribuida realizada de manera colaborativa. Una composición distribuida y colaborativa, en donde el número de dispositivos y servicios es dinámico se asemeja más a la visión de la computación ubicua planteada por Weiser [161].

- Además de ser centralizados, varios de los sistemas emplean dispositivos con capacidades avanzadas de computación para la composición, por ejemplo, empleando PDAs y PCs. Pero en entornos ubicuos la mayoría de dispositivos que se encuentran en el mismo no tiene gran capacidad de procesamiento, ya que deben ser dispositivos pequeños y por lo tanto con una capacidad de procesamiento adecuada al mismo. Ello trae consigo un problema, ya que si se pretende emplear una composición distribuida con dispositivos con recursos limitados es posible que las tecnologías y estándares actuales no puedan ser empleados, por lo que tal vez tengan que ser adaptados para ser aplicables a dicha casuística.

- En muchos de los sistemas la interacción con el usuario es muy alta y necesaria, pero creemos que esto no debe de ser as, el entorno debe de

conocer el perfil y el comportamiento que el usuario desea que adopte dicho entorno y adaptarse a l, siendo prácticamente inexistente la interacción con el usuario. Para ello es necesario que el usuario previamente defina que tareas o que comportamientos quiere realizar, definiendo las tareas mediante un lenguaje especialmente definido para ello. De esta manera es posible, por ejemplo, crear un dispositivo inyecte en el entorno las tareas que desea realizar el usuario cuando este entre en una habitación para hacer conscientes a los dispositivos de cuáles son las necesidades del usuario y para que los dispositivos cooperen para componer tareas de alto nivel para satisfacer las necesidades del usuario.

- En las propuestas planteadas los lenguajes empleados mayoritariamente son BPEL4WS y OWL-S, en cambio, no se han realizado trabajos relativos a composición para computación ubicua con propuestas como WSMO, SWSF, WSDL-S y SAWSDL, esto puede ser debido a que estas últimas propuestas son de mas reciente creación y OWL-S y BPEL4WS son tecnologías más maduras y desarrolladas con mas herramientas de desarrollo y APIs. Por lo que hay una va abierta en el posible empleo de estas nuevas propuestas de Servicios Web Semánticos en los entornos de computación ubicua o su posible adaptación para con recursos limitados (sistemas embebidos).

#### 1.3.3.1 Áreas que requieren profundización técnica

Los dispositivos, aplicaciones y sistemas actuales carecen de los niveles de reactividad y sensibilidad al contexto descritos en los escenarios propuestos por Lassila [162], Weiser [161], Aarts [163] o el ISTAG [164], ya que son sistemas con dispositivos simples que tienen pautas de comportamiento básicos, como pueden ser abrir unas puertas cuando se detecte la presencia cercana de una persona, encender una luz cuando se detecta la proximidad de una persona, redireccionar una llamada al teléfono más cercano que se encuentra una persona, etc., o bien son sistemas centralizados en donde un ordenador central es quien tiene toda la inteligencia y los dispositivos que se encuentran alrededor solamente se encargan de recibir órdenes y actuar, sin tener iniciativa propia [165].

Para conseguir comportamientos más inteligentes, avanzados, espontáneos y con capacidad de razonamiento y aprendizaje autónoma es necesario que los dispositivos

de los entornos inteligentes tengan cierta inteligencia, pero también es indispensable que sean ligeros, es decir, pequeños y que consuman poca energía, para así embeberlos en los objetos cotidianos. Nosotros pensamos que los entornos inteligentes sólo pueden ser construidos sobre la base de sistemas embebidos y no sobre sistemas en donde son necesarios grandes dispositivos como PCs, que crean escenarios irreales y artificiales alejados de la visión de la computación ubicua.

El problema de estas dos variables es que son inversamente proporcionales, ya que si se pretende que los dispositivos sean ligeros y pequeños, nos encontramos con que tenemos que emplear sistemas con muy poca capacidad de procesamiento, lo que hace que sea muy difícil desarrollar dispositivos inteligentes, y por el contrario si queremos que el dispositivo sea inteligente necesitamos de elementos con mayor capacidad de procesamiento y consumo de energía que hacen que tengamos un dispositivo grande, perdiendo por tanto la ligereza. Es por ello que nos encontramos en el punto en el que hay que encontrar el equilibrio adecuado entre ambas variables, o bien decantarse por alguna de las dos sacrificando a la otra.

Creemos que es posible un término medio, en donde el sistema sea ligero, es decir, sea pequeño y consuma poca energía, pero a la vez tenga inteligencia, es decir, realice razonamiento sobre el contexto del usuario, los inputs percibidos y los objetivos deseados y sea capaz de comunicarse y coordinarse con otros dispositivos que se encuentren en el entorno para componer tareas más complejas. Y para ello creemos necesario establecer una cadena de valor que proporcione inteligencia al entorno, para que actúe con reactividad y sensibilidad al contexto basándose en las preferencias de los usuarios. La cadena de valor resultante es la siguiente: Sensorización y perfiles de usuario->Percepción de contexto (basada en ontologías)->Razonamiento (basado en reglas)->Composición (basada en estrategias)->Operación->Dispositivos y aparatos. Paralelo a esta cadena de valor estará el aprendizaje inicial que determinara las reglas iníciales y la adaptación posterior, es decir, el aprendizaje permitirá crear el contenido de las reglas, los perfiles de usuario, etc.

La posibilidad de instalar las capacidades de la Web Semántica en plataformas embebidas supone un desafío importante. Ya que pese a que se hayan realizado intentos para aplicar estas tecnologías para obtener entornos más reactivos y conscientes del usuario, los resultados obtenidos han sido desiguales, por lo que la adaptación de las nuevas tecnologías de la web semántica para su uso en plataformas embebidas para entornos inteligentes supone un reto.

Consideramos que los conceptos claves empleados en las tecnologías de Servicios Web Semánticos para la interoperabilidad y composición automática de procesos empresariales pueden ser extrapolados para obtener el mismo efecto en procesos ambientales, es decir, colaboración automática de servicios ambientales para obtener un comportamiento combinado del entorno por el bien del usuario.

Por ejemplo, si un usuario se encuentra viendo la TV con la luz ambiente apagada y suena el teléfono, se podrá efectuar un comportamiento coordinado del entorno y sus dispositivos que resulten en que el número llamante aparece superpuesto en la pantalla del TV, se afecta una disminución del volumen del altavoz del mismo y un encendido de las luces para facilitar la atención de la llamada y la subsiguiente conversación. En este caso existen varios servicios en los dispositivos del entorno (superposición de texto, ajuste de volumen, encendido y apagado de la luz) que deben coordinarse y colaborar como respuesta al evento de llamada entrante en el teléfono.

Una infraestructura de Servicios Web Semánticos Ambientales permitirá diseñar y desplegar este tipo de escenarios y configurarlos a la medida del usuario proporcionándole un modelo más avanzado y asistivo de interacción con el entorno. Por ello consideramos el estudio y experimentación en este campo como un desafío tecnológico a abordar y puede requerir:

- Adaptación de estándares existentes de composición (orquestación / coreografía) para la creación y ejecución de procesos empresariales distribuidos en entornos de computación ubicua (servicios ambientales) donde los dispositivos tienen restricciones respecto a su capacidad de procesamiento y comunicación, ya que los dispositivos están distribuidos en el entorno.

- Desarrollo de especificaciones nuevas para la composición distribuida de servicios ambientales y así cumplir ciertos requisitos particulares que se dan en este tipo de escenarios.

## 1.4 Descubrimiento de servicios/dispositivos semánticos

### 1.4.1 Introducción

La presente sección tiene como objetivo realizar un análisis crítico del estado del arte

relativo al descubrimiento y emparejamiento de servicios en entornos de computación ubicua. Para ello inicialmente se describen cuáles tienen que ser las características principales y los componentes de un mecanismo de este tipo, para después describir los diferentes problemas que hay que abordar en el descubrimiento de servicios. Tras ello se realiza un análisis de los diferentes tipos de emparejamiento semántico y posteriormente se hace un repaso de los dos principales enfoques para descubrimiento de servicios eficiente en entornos de computación ubicua.

### 1.4.2 Componentes de un mecanismo de descubrimiento de servicios

En general, cualquier mecanismo de descubrimiento de servicios tiene que disponer los siguientes componentes [166]:

- Descripción de servicio: Lenguaje empleado para describir tanto la semántica funcional como la no-funcional de un servicio.

- Selección de servicios: Mecanismo de razonamiento para el emparejamiento (matching, retrieval, matchmaking, etc.) de servicios que consiste en la comparación de pares de descripciones de servicios en términos de la semántica funcional y no-funcional, así como la clasificación de los resultados en base a criterios de emparejamiento y preferencias.

- Arquitectura de descubrimiento: Arquitectura empleada como base para el descubrimiento, haciendo hincapié en la topología de red (centralizada, descentralizada), almacenamiento de la información de servicios (distribución de los servicios, ontologías, registros) y servicios de localización así como la funcionalidad de los agentes implicados (proveedor de servicios, consumidor de servicios, middle agent).

### 1.4.3 Arquitectura de descubrimiento

El enfoque ideal de descubrimiento para entornos pervasivos tiene que abarcar un amplio rango de factores relativos a interoperabilidad, debido a la heterogeneidad de estos entornos y debido al hecho de que los servicios pervasivos y los clientes potenciales (clientes que realizan las peticiones de servicios) son diseñados, desarrollados y desplegados independientemente.

Los protocolos de descubrimiento de servicios permiten que los servicios puedan ser descubiertos en la red, permitiendo de esta manera oportunidades para la colaboración, para así componerse con otros servicios para crear servicios más complejos, cumpliendo las necesidades de las aplicaciones. Muchos protocolos de descubrimiento de servicios han sido propuestos tanto por la industria como por la comunidad científica, como pueden ser UDDI o el Trading Service de CORBA para Internet o SLP y Jini para redes de área local o redes ad-hoc.

En el artículo [167] se realiza una clasificación de los diferentes protocolos de descubrimiento de servicios que realiza una distinción entre los protocolos pull-based y los protocolos push-based. En los protocolos pull-based, el cliente envía la petición al proveedor de servicios (distributed pull-based mode) o a un repositorio de un tercero (centralized pull-based mode) con el objetivo de conocer la lista de servicios compatibles con la petición realizada. En los protocolos push-based, los proveedores de servicio proveen las descripciones de los servicios a los clientes que localmente mantienen la lista de los servicios disponibles en la red.

Los principales protocolos de descubrimiento de servicios, como Jini y SSDP, emplean el enfoque pull-based para el descubrimiento, soportando algunas veces tanto el modo centralizado como modelos de interacción distribuidos (SLP, WS-Discovery). En los protocolos de descubrimiento centralizado tipo pull-based uno o varios repositorios almacenan las descripciones de los servicios disponibles en la red y la localización de los repositorios puede ser conocida (UDDI) o dinámicamente descubierta (Jini). Los repositorios se mantienen actualizados gracias a que los proveedores se dan de baja explícitamente o bien gracias a que dan de baja a los servicios periódicamente debido a que ha expirado la vigencia del mismo. Si existen múltiples repositorios, estos cooperan entre ellos para distribuirse los registros de los nuevos servicios o también para encaminar las peticiones de servicios al repositorio relevante de acuerdo a relaciones preestablecidas.

Pese a que en la actualidad existen muchas soluciones relativas a arquitecturas de descubrimiento de servicios todavía hay retos y problemas a solucionar en los entornos de computación pervasiva, como se verá a continuación:

#### 1.4.3.1 Descubrimiento de servicios multiprotocolo

La heterogeneidad de middlewares hace que surjan problemas de interoperabilidad

entre los diferentes protocolos de descubrimiento de servicios (por ejemplo: SLP, SSDP, UDDI). Los protocolos de descubrimiento de servicios actuales no son capaces de interactuar con otros protocolos, ya que emplean formatos y protocolos incompatibles para la descripción de servicios o para las peticiones de descubrimiento, e incluso tipos de datos o modelos de comunicación incompatibles. En cualquier caso, las características de los entornos inteligentes y los estándares de-facto de algunos de los protocolos existentes hace imposible que surja un nuevo y único protocolo de descubrimiento de servicios.

Diversos proyectos han tratado de buscar soluciones de interoperabilidad [168-170], ya que requerir a los clientes y proveedores de servicios que sean capaces de soportar múltiples protocolos de descubrimiento de servicios no es real. Los protocolos de descubrimiento de servicios interoperables típicamente han empleado un modelo común para la representación de los elementos de descubrimiento de servicios (por ejemplo: descripción del servicio, petición de descubrimiento) [171] en vez de realizar mapeos directos entre los protocolos [170], no pudiendo ser este último enfoque muy escalable con un número elevado de protocolos. Más allá, la capa de interoperabilidad debe de estar alojada cerca de la capa de red [171] y traducir eficientemente y transparentemente los mensajes de red entre los diferentes protocolos, o bien proveer de un interface explicito [172] a los clientes o servicios para así ampliar los protocolos existentes introduciendo características avanzadas tales como la gestión del contexto.

#### 1.4.3.2 Descubrimiento de servicios multi-red

La heterogeneidad de la red hace que muchas redes independientes (que se pueden interconectar libremente con los dispositivos multi-radio móviles actuales) estén disponibles para los usuarios en una localización, por lo tanto soluciones innovadoras son necesarias para la diseminación, filtrado y selección de peticiones de descubrimiento servicios y el anuncio de los mismos [172]. Diversos proyectos han investigado el empleo del enrutamiento entre pares (gateways) o a nivel de aplicación (P2P) combinado con filtros inteligentes que proveen de descubrimiento de servicios eficiente y escalable en entornos multi-red.

El protocolo mSLP [173] mejora la eficiencia y la escalabilidad del protocolo de descubrimiento SLP introduciendo mejoras en la malla así como filtros de preferencia para el emparejamiento de los registros. INS/Twine [174] propone una arquitectura

P2P escalable donde los servicios de directorio colaboran como pares para distribuir la información de los recursos y para resolver las consultas. GloServ [175] es una arquitectura global de descubrimiento de servicios basada en ontologías que opera tanto en redes de área extensas como en redes de área local empleando una arquitectura híbrida (jerárquica y peer to peer). MUSDAC [172] permite unir dinámicamente redes que se encuentran cerca mediante componentes específicos que proveen de enrutamiento a nivel de aplicación en dispositivos multi-radio, lo que permite la diseminación de peticiones de descubrimiento en el entorno. Un factor clave en el descubrimiento de servicios multi-red es divulgar cambios dinámicos en la red sin que ello suponga compromiso alguno en el procesamiento ni en los recursos de la red debido a la considerable cantidad de información que es procesada e intercambiada.

#### 1.4.3.3 Movilidad en el descubrimiento

La movilidad tanto del cliente como del servicio ha sido raramente investigada en el contexto de descubrimiento de servicios. El problema de la movilidad es principalmente visto como un problema relativo al acceso al servicio (por ejemplo: mantenimiento de la conectividad). Sin embargo, el empleo de patrones de movilidad ha sido empleado como medio para favorecer la interacción entre los nodos que continuarán estando al alcance en la red durante la duración completa de la sesión [176].

#### 1.4.3.4 Seguridad y privacidad en el descubrimiento

El intercambio de descripciones y las peticiones de descubrimiento debe ser seguro, ya que es crucial en entornos inteligentes, especialmente cuando la información está relacionada con información contextual relativa al usuario o al servicio que puede ayudar a conocer información privada del usuario [177]. Muchos mecanismos para asegurar el acceso a la información del servicio han sido propuestos centrándose bien en la encriptación y los derechos de acceso [178] o en la necesidad de equilibrar el acceso y la privacidad [177].

### 1.4.4 Descripción de servicio

Además de la arquitectura que permite descubrir los dispositivos del entorno, es necesario definir el lenguaje en el que se van a representar las características

funcionales como no-funcionales de los servicios del entorno. Los enfoques de representación basados en semántica ofrecen mayor flexibilidad y expresividad para representar los servicios de entornos ubicuos, por lo tanto en las siguiente líneas se describirán y analizaran las principales iniciativas para la representación de servicio semánticos [179].

#### 1.4.4.1 OWL-S

OWL-S [180] anteriormente denominada DAML-S [170, 181], es una ontología para WS realizada en OWL [182, 183], un lenguaje recomendado por el W3C para representar ontologías y que sean fácilmente procesables por máquinas. El lenguaje OWL proporciona un conjunto de construcciones para representar las relaciones presentes en una ontología. Estas construcciones permiten representar conceptos, herencia, relaciones entre los conceptos, y restricciones sobre los valores que pueden tomar las propiedades de un concepto. OWL proporciona mayores facilidades a la hora de representar el significado y la semántica que otros lenguajes como pueden ser XML y RDF. OWL-S fue la primera propuesta enviada al W3C relativa a SWS, y se remonta al 2 de noviembre del 2004.

OWL-S es, por tanto, una ontología creada a partir de OWL que define los conceptos y relaciones necesarias para describir semánticamente un WS. Las descripciones de servicios se realizan mediante axiomas OWL-DL (uno de los tres sublenguajes de OWL), pero se ha visto que no aporta la expresividad necesaria para la descripción de los servicios, por lo que posteriormente OWL-S ha sido mejorado para que pueda permitir expresiones lógicas de otros lenguajes como pueden ser: DRS [184], KIF [185], SWRL [186] y PDDL [187]. El entorno de ejecución de OWL-S se denomina OWL-S VM [188].

OWL-S constituye una ontología de alto nivel con la cual se construye la descripción semántica de cada servicio. Esta ontología proporciona una clase principal definida como *Service* desde la que se puede acceder al resto de la descripción del servicio. A partir de esta clase se puede obtener el perfil del servicio (*ServiceProfile*), el modelo del servicio (*ServiceModel*) y el soporte del servicio (*ServiceGrounding*):

- **ServiceProfile**: La clase *Service* presenta un perfil de servicio (*Profile*). Este perfil describe qué es lo que el servicio hace. Proporciona información sobre el

proveedor, un conjunto de atributos que permiten describir características del servicio y su descripción funcional. Los datos sobre el proveedor del servicio hacen referencia a nombre, administrador, información de contacto, etc. En cuanto a los atributos que permiten describir el servicio, éstos incluyen conceptos como la categoría a la que pertenece el servicio dentro del UNSPSC, su fiabilidad (buena, muy buena, aceptable, etc.) o la puntación del servicio en algún sistema de valoración de WS. La descripción funcional incluye la especificación de que funcionalidad proporciona el servicio y que precondiciones deben satisfacerse para obtener el resultado deseado. OWL-S presenta dos aspectos de la funcionalidad del servicio: la transformación de la información, representada por las entradas y salidas, y por otro lado el cambio de estado producido por la ejecución del servicio, es decir, que precondiciones son necesarias para su ejecución y que efectos se producen tras ella. OWL-S proporciona las clases necesarias para describir los parámetros, los efectos, las precondiciones y todos aquellos conceptos necesarios para modelar la descripción funcional del servicio. Para realizar estas descripciones funcionales se utiliza la jerarquía de clases definida para el modelado del comportamiento del servicio y que se corresponden con el ServiceModel explicado a continuación.

- **ServiceModel**: Después de haber seleccionado un servicio las tareas de invocación, composición e interoperación se realizan mediante el modelo del servicio. Existen tres tipos de procesos, que a su vez son subclases del la clase *Process*. El proceso define las maneras en las que el cliente puede interactuar con el servicio. OWL-S hace una diferenciación entre tres tipos de procesos: los procesos simples *SimpleProcess*, proceso no invocable utilizado como elemento de abstracción), los procesos atómicos (*AtomicProcess*) y los procesos compuestos (*CompositeProcess*). Un proceso atómico es una descripción de un servicio que espera un mensaje de entrada y produce un mensaje de salida. Un proceso compuesto es aquel que mantiene un estado interno, de tal forma que cada mensaje que envía el cliente produce un cambio en ese estado. Un proceso compuesto necesita el envío de varios mensajes para producir el resultado deseado. Para cada proceso es posible especificar las precondiciones y los efectos. Un proceso puede tener una o más precondiciones que deben ser cumplidas para que el servicio produzca el efecto correcto. La ontología proporciona clases para representar parámetros,

entradas y salidas, y las precondiciones y efectos del servicio. Los procesos compuestos se pueden descomponer en otros procesos que pueden ser simples o compuestos a su vez. En OWL-S su descomposición puede ser especificada usando estructuras como las secuencias (*Sequence*) o estructuras condicionales de tipo (*If-Then-Else*). Estas estructuras no representan cual es el comportamiento interno real del servicio, sino que explican el comportamiento que el servicio tiene tal como lo ve el usuario desde fuera. Además OWL-S incluye la posibilidad de describir el flujo de datos, pudiendo definir como la entrada de un subproceso esta relacionada con la entrada de otro, todo ello mediante las clases *InputBinding* y *OutputBinding*.

- **ServiceGrounding**: A través de esta rama de la ontología principal se puede conocer en detalle cómo acceder a un determinado servicio. El *ServiceProfile* y el *ServiceModel* son abstracciones de alto nivel de un servicio, solo el *ServiceGrounding* conoce las especificaciones reales del mismo a bajo nivel. Su función es determinar cómo las entradas y salidas que han sido especificadas en un nivel más alto son convertidas en mensajes concretos que transportan esas entradas y salidas en un formato específico. Los desarrolladores de OWL-S han elaborado esta correspondencia para WSDL y para SOAP. Cada mensaje WSDL es derivado de una especificación realizada en OWL-S a través del uso de transformaciones XSLT.

OWL-S presenta dos aspectos de la funcionalidad del servicio: por un lado la transformación de la información, representada por las entradas y salidas, y por otro lado el cambio de estado producido por la ejecución del servicio, es decir, que precondiciones son necesarias para su ejecución y que efectos se producen tras ella. OWL-S proporciona las clases necesarias para describir los parámetros, los efectos, las precondiciones y todos aquellos conceptos necesarios para modelar la descripción funcional del servicio.

A pesar de que OWL-S no define un lenguaje específico para representar las reglas de los efectos y precondiciones, recomienda y proporciona algunas facilidades para trabajar con el lenguaje Semantic Web Rule Language (SWRL) y ofrece mecanismos para representar dichas fórmulas en otros lenguajes.

Los Inputs, Outputs, Preconditionss y Effects se conocen como IOPEs. En la última versión de OWL-S, los efectos son definidos como parte de un Result. Utilizamos el

término Result para referirnos al conjunto formado por un Output y un Effect. El esquema para describir los IOPEs está definido en el Process.

Los Inputs y Outputs especifican la transformación de datos producida por el proceso bajo unas condiciones. Describen la información que se requiere para la ejecución del proceso y la información que produce el servicio. En el caso de procesos atómicos la información se recibirá del cliente y en el caso de los compuestos, algunos Inputs se recibirán del cliente y otros serán los Outputs recibidos de procesos anteriores. El número de Inputs y Outputs en un proceso es indefinido, pudiendo ser incluso cero.

En OWL-S, las precondiciones y los efectos se representan por fórmulas lógicas. El número de precondiciones y efectos también es indefinido. Las Precondiciones describen las condiciones del estado del mundo que deben ser ciertas para poder ejecutar el servicio satisfactoriamente. Se tratan de subclases de *Expression*. En OWL-S las expresiones especifican el lenguaje en el que la expresión está descrita (SWRL, KIF, DRS) y la propia expresión está codificada como literal (string o XML) dependiendo del lenguaje para expresiones. Los Efectos describen las condiciones del estado del mundo que son verdaderas después de la ejecución del servicio. Están definidos como parte de un Result.

El resultado de la invocación de un servicio se puede ver como una salida y un efecto. El resultado de un servicio se puede condicionar mediante las propiedades inCondition, hasResultVar, withOutput, hasEffect. La propiedad inCondition especifica la condición bajo la que se produce ese resultado y no otro. Las propiedades withOutput y hasEffect establecen qué sucede cuando la condición se satisface. La propiedad hasResultVar declara variables usadas en inCondition. Estas variables, llamadas ResultVars, son análogas a las variables Locals, y sirven para un propósito similar, permitiendo ligar las variables a las precondiciones y usarlas en las declaraciones de salida y efectos.

#### 1.4.4.2 WSMO

WSMO [189-191] es otra ontología para descripción de Servicios Web semánticos, realizada tomando como base el *framework* para WS llamado WSMF [192]. Al contrario que OWL-S, el proyecto WSMO no sólo pretende crear la especificación, sino que, también crear la arquitectura y un conjunto de herramientas para soportar la especificación [193]. El entorno de ejecución de WSMO se denomina WSMX [194] y

es la implementación de referencia de WSMO para la integración de aplicaciones empresariales. WSMO fue enviado al W3C como propuesta el 4 de abril del 2005.

WSMO define cuatro conceptos principales: ontologías, WS, metas (*goals*) y mediadores (*mediators*). Además define su propio lenguaje para definir ontologías denominado WSML [195] que está basado en F-Logic [196, 197]. Hay cinco variantes de WSML en base a la expresividad y el alcance del lenguaje: *Core, DL, Flight, Rule, Full*. WSMO está definido utilizando un lenguaje llamado *MOF* [198]. MOF define una arquitectura de meta-datos consistente en cuatro capas: la capa de información que contiene la información que se quiere describir, la capa de modelo que contiene la meta-información que describe la información de la capa anterior, la capa del meta-modelo que contiene la información que describe la estructura y semántica de la meta-información, y por último, la capa de meta-meta-modelo que describe la información y estructura de la capa del meta-modelo. La ontología de modelado WSMO está definida en la capa de meta-meta-modelado.

En WSMO cada elemento está identificado mediante una URI o un identificador anónimo. Los identificadores anónimos se utilizan para denotar elementos que existen pero no necesitan un identificador específico.

Las ontologías se definen como clases de MOF que contienen una serie de propiedades. Estas propiedades pueden ser descriptivas como el autor, la descripción de la ontología, la fecha, su formato, su identificador, el lenguaje, etc. O atributos funcionales: ontologías importadas a las que hace referencia esta ontología, mediadores utilizados para que las ontologías importadas puedan ser utilizadas entre sí, relaciones, funciones, instancias y axiomas.

Un concepto es el elemento básico de una ontología, las propiedades de un concepto son sus atributos, que definen campos que pueden ser rellenados con valores cuando se realice una instancia del concepto. Además, todo concepto puede tener un super-concepto que actúa como padre en una relación de jerarquía. Las propiedades del concepto padre son heredadas por el concepto hijo. Todo concepto puede poseer una expresión lógica que sirve para definir la semántica del concepto formalmente.

Las relaciones se utilizan para modelar las interdependencias entre conceptos, concretamente entre las instancias de los mismos. Toda relación puede tener una super-relación, lo que significa que heredará las restricciones definidas en la misma.

Una relación tiene un número de parámetros que intervienen en ella. Las restricciones de una relación son especificadas mediante expresiones lógicas. Las funciones de una ontología son un tipo especial de relación. Pueden ser evaluadas especificando valores para los parámetros de la misma.

En WSMO un servicio queda definido, entre otras propiedades, por su capacidad, que define el servicio de acuerdo a su funcionalidad. La capacidad de un servicio queda definida por sus precondiciones, postcondiciones y efectos. Además un servicio puede tener interfaces. Una interfaz define como la funcionalidad del servicio puede ser conseguida. Los objetivos, por su parte, son descripciones de los problemas que deben ser resueltos por los servicios. Un objetivo queda descrito por las capacidades de los servicios que el usuario querría utilizar y por la interfaz que el usuario querría que tuviera el servicio con el que interactuar. WSMO también proporciona una serie de propiedades para definir a los mediadores.

El mecanismo para describir coreografías y orquestación en WSMO está basado en la metodología de *ASM* [199, 200]. El modelo implementado sigue estos principios: está basado en estados, representa los estados mediante álgebra y modeliza los cambios de estado mediante reglas de transición almacenadas que cambian los valores de las funciones y las relaciones definidas en el álgebra.

En WSMO un Servicio Web (WS) se define, entre otros parámetros, mediante las propiedades Interface y Capability. La propiedad Interface especifica detalles operacionales del WS mientras que Capability se refiere a sus capacidades funcionales.

En WSMO se especifica que cada Web Service sólo puede tener una Capability (multiplicity = single-valued). Una Capability puede tener Preconditions, Assumptions, Postconditions y Effects, que son objetos de tipo Axiom, expresiones lógicas compuestas de reglas escritas en WSML. Desde un punto de vista funcional, si se cumplen los axiomas definidos en Preconditions y Assumptions, se puede asumir que el servicio dispone de la Capability, y por tanto, se pueden ejecutar los axiomas que procedan de entre los definidos en Postconditions y Effects.

Las Preconditions describen los estados válidos del espacio de información previos a la ejecución del WS. El espacio de información se refiere a los Input de las operaciones para la ejecución del servicio en cuestión. Del mismo modo, las

Postconditions describen los estados válidos de información que se garantiza tras la ejecución. Es decir, describen los Outputs obtenidos tras la ejecución del WS.

Por contra, los Effects describen los efectos que la ejecución del servicio provoca en su entorno; o dicho de otro modo, los cambios que ocurren en el ambiente. En cambio, las Assumptions describen los estados válidos y necesarios del entorno o ambiente para la ejecución del WS.

#### 1.4.4.3 SWSO

SWSF [201] es una propuesta de la iniciativa SWSI para el desarrollo de SWS. Esta iniciativa agrupa a investigadores del programa DAML y de proyectos europeos relativos a la Web Semántica. El trabajo del SWSI está estrechamente relacionado con OWL-S y WSMO, que son tomadas como referencia por los desarrolladores de SWSF. SWSF fue enviado al W3C como propuesta el 9 de mayo del 2005.

El modelo SWSF incluye 2 componentes principales:

- **SWSL** [202] es un lenguaje basado en lógica para especificar caracterizaciones formales de conceptos y descripciones de Servicios Web. Su principal objetivo es actuar como base para la ontología SWSO, el modelo conceptual de SWSI para los SWS. SWSL tiene dos variantes. La primera de ellas es SWSL-FOL, que tiene como objetivo la definición de la ontología del proceso y la otra es SWSL-Rules, un sublenguaje basado en reglas para soportar razonamiento sobre ontologías de lógica de orden uno. SWSL-FOL y SWSL-Rules son subconjuntos diferentes y separados de SWSL, siendo empleados para diferentes tareas, aunque el conocimiento expresado en los dos lenguajes puede ser combinado.

- **SWSO** [203] define el modelo conceptual para la definición de SWS en SWSL. La ontología SWSO, expresada en SWSL-FOL, también denominada FLOWS [204], tiene como objetivo la declaración de SWS y el razonamiento en base a éstos. Para conseguir este objetivo, incluye mecanismos de la familia de lenguajes PSL [205-207] para la descripción de procesos de negocio. SWSO puede ser visto como una extensión o refinamiento de OWL-S, pero aunque tenga muchas similitudes con las ontologías de OWL-S una gran diferencia es la expresividad del lenguaje subyacente.

La estructura de FLOWS es muy parecida a la propuesta por OWL-S, la cual, consiste en tres componentes principales:

- **Service Descriptors** provee información básica acerca del web service (nombre, autor, información de contacto, etc.) para que sea utilizado en el descubrimiento y emparejamiento de servicios en base a las preferencias del cliente.

- **Process Model** adapta la ontología genérica de los procesos PSL para proporcionar un *framework* para la descripción de WS. El Process Model es creado como un conjunto de seis modulos de ontologia del PSL-OuterCore. Este fundamento modular del modelo de proceso de SWSO hace que los desarrolladores de WS puedan definir y utilizar sus propias extensiones a FLOWS-Core. De hecho un objetivo principal de SWSO es facilitar la integración con otros lenguajes para el modelado de *workflows*, como puede ser BPEL4WS. SWSL permite traducir parcialmente SWSL-FOL a SWSL-Rules. Consiguiendo de esta manera expresar FLOWS mediante reglas. Éste es el objetivo de ROWS, una traducción de FLOWS a SWSL-Rules, que permite describir el *Process Model* mediante programación lógica.

- **Grounding** permite declarar detalles de bajo-nivel de las definiciones de los Servicios Web, como pueden ser: formato de los mensajes, protocolos de transporte, direccionamiento de red, etc. y tiene como objetivo el mapeo de descripciones de alto nivel de SWSO con WSDL.

FLOWS-Core incluye las primitivas de modelado de servicios básicas: servicio, proceso atómico e IOPEs, composición de actividades, mensaje y canal. Un bloque fundamental del modelo de proceso para Web Services de FLOWS es el concepto de proceso atómico. Un proceso atómico es una actividad PSL que generalmente es una subactividad de la actividad asociada con un servicio. Un proceso atómico es directamente invocable, no tiene subprocesos y se ejecuta en un sólo paso. Dicha actividad puede ser empleada para describir un Web Service simple o bien puede ser parte de una composición más compleja junto con otros procesos atómicos o compuestos.

Asociados con cada proceso atómico hay múltiples parámetros de entrada, salida, precondiciones y efectos (condicionales). Las entradas y las salidas son las entradas y

salidas del programa que realiza el proceso atómico. Las precondiciones son aquellas condiciones que tienen que ser ciertas en el mundo para que el proceso atómico pueda ser ejecutado. En la especificación se dice que en muchos casos no habrá parámetros de precondición ya que la mayoría del software no tiene precondiciones físicas de ejecución, al menos al nivel en el que se pretende modelizar SWSO. Los efectos del proceso atómico representan las condiciones del mundo que son ciertas tras la ejecución del proceso atómico. Las precondiciones y efectos de un proceso atómico son expresados mediante una fórmula lógica de primer orden.

Además de las IOPEs de los procesos atómicos existen, las IOPEs asociadas a los servicios que están asociadas al comportamiento complejo del servicio. Pero éstas no son formalmente definidas ya que las entradas, salidas, precondiciones y efectos solamente pueden ser especificadas en los procesos atómicos. Sin embargo, para ciertas tareas automáticas, como el descubrimiento de servicios y la composición es necesario considerar las entradas, salidas, precondiciones y efectos del comportamiento complejo que describe el modelo de proceso completo del servicio. Y para ello, en algunos casos, estas propiedades pueden ser inferidas de los IOPEs de los procesos atómicos que constituyen el modelo de proceso completo.

#### 1.4.4.4 WSDL-S

WSDL-S [208] es un pequeño conjunto de extensiones propuestas para WSDL, que tiene como objetivo asociar anotaciones semánticas a elementos WSDL como interfaces y tipos de esquemas de operaciones: precondiciones, efectos, salidas y entradas. Para ello aumentan la expresividad de las descripciones WSDL asociando ciertos elementos clave con conceptos expresados en ontologías. Estos Servicios Web Semánticos pueden ser además publicados en un registro UDDI y para ser descubiertos dinámicamente utilizando conceptos expresados en ontologías. Una de las grandes ventajas de WSDL-S respecto al resto de propuestas es que está basada estándares de Servicios Web ya existentes, cosa que el resto no lo hace.

En la especificación de la propuesta se resalta que se ha aumentado la expresividad de WSDL con semántica utilizando conceptos análogos a los que utiliza OWL-S, siendo agnósticos al lenguaje de representación semántico empleado en el mismo. El modo en el que WSDL-S permite especificar la correspondencia entre elementos WSDL y conceptos semánticos es muy similar a como lo hace el *ServiceGrounding*.

Lo que pretende esta propuesta es externalizar los modelos de dominio semánticos, es por ello que WSDL-S no está atado a ningún lenguaje de representación de ontologías, facilitando de esta manera la reutilización de modelos de dominio existentes, pudiendo elegir el lenguaje que más nos interese y permitiendo la anotación utilizando múltiples ontologías (del mismo o diferente dominio). En la propuesta ser resalta que es posible emplear WSDL-S enriquecido con anotaciones semánticas de OWL y WSMO.

Kopecký y otros [209] plantean una propuesta para extender WSDL-S para que cubra todas las características de WSMO, pero sin comprometer el objetivo principal de WSDL-S: la simplicidad y la independencia y resaltan que no tienen constancia de que se esté realizando trabajo alguno en lo referente a la integración entre WSDL-S y OWL-S [209].

WSDL-S está asociada con un entorno de ejecución y con un conjunto de herramientas denominado METEOR-S [210] llevado a cabo por el grupo LSDIS de la Universidad de Georgia. WSDL-S fue enviado al W3C como propuesta el 1 de octubre del 2005.

El proyecto METEOR-S define un amplio *framework* semántico que cubre todas las fases del ciclo de vida de los procesos Web [211]. Y muestra de ello son los resultados obtenidos por esta propuesta como pueden ser en la agregación de QoS [212], anotación semántica de Servicios Web [213], descubrimiento de Servicios Web [214], emparejamiento semántico [193, 215] y composición [115, 216, 217]. WSDL-S ha sido tomado como entrada por el grupo de trabajo del W3C para la nueva recomendación SAWSDL que pretende añadir semántica a las descripciones WSDL.

La especificación de WSDL-S enviada al W3C define que una precondición es el conjunto de statements (o expresiones representadas empleando conceptos de un modelo semántico) que se tienen que cumplir para que la operación sea satisfactoriamente invocada. Un efecto es el conjunto de statements (o expresiones representadas empleando conceptos de un modelo semántico) que son necesarios que se cumplan después de que una operación haya finalizado su ejecución tras ser invocada. Los efectos pueden ser diferentes en caso de que la operación se haya completado satisfactoriamente o no.

El elemento PreCondition se define en base al atributo Name en conjunto con

ModelReference o Expression (el atributo ModelReference y el atributo Expression son mutuamente exclusivos). El atributo Name específica un identificador único para representar a una precondición dentro del conjunto de precondiciones en el documento WSDL. El atributo ModelReference específica la URI de la parte del modelo semántico que describe la precondición. El atributo Expression representa a la expresión que define la precondición. El formato de la expresión viene definido por el lenguaje semántico de representación empleado para expresar el modelo semántico. El elemento Effect se define de la misma manera que el PreCondition, pero además, un Effect puede estar asociado a un Fault, para ello basta con añadir al elemento Fault de la operación el elemento Effect correspondiente.

Las precondiciones y efectos son especificados como elementos hijo de la operación Element. Pese a que los esquemas especifican que solamente se puede tener una precondición y tantos efectos posibles como se quieran, en la especificación de dice que cada operación puede tener como máximo una precondición y un efecto, para así mantener la especificación simple. La especificación detalla que las precondiciones y los efectos complejos y condicionales deberían ser expresados en el modelo dominio/semántico. Por ejemplo, un conjunto de precondiciones (efectos) debería ser definido mediante expresiones lógicas que faciliten operaciones del tipo 'and', 'or', 'xor', etc. al evaluar las expresiones. En la especificación, se presupone que el lenguaje empleado para la representación semántica soporta la representación de múltiples precondiciones en una única precondición de alto nivel. Ésta puede ser referenciada en el elemento Operation dentro de la especificación WSDL.

Respecto a lenguajes de representación de efectos y precondiciones, en la especificación no se establece cuál hay que emplear sino que hacen referencia a varias propuestas, tales como, SWRL (comunicad de la Web Semántica) y OCL (comunidad de modelado).

#### 1.4.4.5 SAWSDL

SAWSDL [218] tiene como objetivo añadir semántica a las descripciones de los Servicios Web. Ya que la especificación de WSDL 2.0 [219] no incluye semántica en la descripción, por ejemplo dos servicios pueden tener una descripción similar pero tener significados completamente diferentes. Para cubrir dicha carencia SAWSDL define cómo añadir anotaciones semánticas a varias partes de documentos WSDL, como

pueden ser estructuras de mensajes de entrada y salida, interfaces y operaciones. Las extensiones definidas en la especificación encajan en el marco de extensibilidad de WSDL 2.0. SAWSDL tiene dos tipos básicos de anotación denominados *ModelReference* y *SchemaMapping*:

- La anotación ModelReference, al igual que el ModelReference de WSDL-S, es utilizado para asociar tipos de interfaces/puertos, operaciones, salidas, entradas y elementos y atributos de XML schema con conceptos semánticos.

- La anotación SchemaMapping es empleada para transformar la representación de un dato en otro, como también lo hace WSDL-S.

En la recomendación del W3C no se detalla en ningún lugar que puedan representarse efectos y precondiciones en los servicios SAWSDL, pero en el Working Group Note del 28 de Agosto del 2007 [220] se describe cómo se pueden representar restricciones relativas al comportamiento de los servicios, es decir algo similar a los efectos y las precondiciones.

Los efectos y precondiciones son descritos dentro de cada operación, y más concretamente, referenciando las reglas que representan las precondiciones en el Input y las que representan los efectos en el Ouput. En los ejemplos que ofrecen las reglas están descritas en SWRL y WSML.

#### 1.4.4.6 AmIGO-S

AmIGO-S [221] es una propuesta que extiende OWL-S integrando características relativas a la heterogeneidad y riqueza de los entornos ubicuos, añadiendo nuevas clases para definir propiedades funcionales (especifican cómo es posible el descubrimiento del servicio descubrimiento, comunicación y protocolos de red) y no funcionales (permitiendo modelizar el contexto del entorno ubicuo y también los atributos de calidad de servicio - QoS) para los servicios Amigo y permitiendo la definición de varios Groundings para un servicio.

Un servicio AmIGO se describe utilizando la clase Profile de OWL-S, que se utiliza para definir las propiedades globales del servicio. Una funcionalidad específica que ofrece un servicio Amigo es la Capability. El servicio se define utilizando tantas clases Capability como sean necesarias. Las propiedades funcionales de una Capability de

un servicio están descritas por sus precondiciones, efectos y sus entradas y salidas. Las propiedades no funcionales de un servicio están descritas a nivel global o para cada Capability del servicio mediante el contexto del servicio y los parámetros QoS del servicio.

En AmIGO-S, un servicio es descrito por uno o más profiles (clase Profile) de OWL-S. Define propiedades (algunas nuevas y otras de OWL-S) globales del servicio, comunes para todas las capabilities proporcionadas del servicio. Se reutilizan todas las propiedades definidas en el Profile de OWL-S, excepto la descripción funcional, que será fijada a nivel de Capability. Las propiedades funcionales de un servicio presentadas en el Profile se especifican con las capabilities del servicio, las conversaciones del servicio, y el Middleware subyacente.

En lugar de especificar la funcionalidad proporcionada por el servicio a nivel de Profile, como en OWL-S, en AmIGO-S se asocia un Profile con una o más Capabilities. Las ProvidedCapabilities permiten describir las Capabilities que son ofrecidas por el servicio y las RequiredCapabilities son las Capabilities que tendrían que ser ofrecidas por servicios externos. Si la RequiredCapability no está disponible, el servicio no pude ofrecer las ProvidedCapabilities. Por lo tanto, se pueden describir varias funcionalidades ofrecidas y requeridas por un dispositivo Amigo. Las propiedades comunes de todas las funcionalidades se especificarán en el Profile, mientras que las especificaciones de cada funcionalidad se describirán en la definición de Capability. La descripción funcional del Profile de OWL-S es inútil ya que las funcionalidades se especifican en la Capability.

Una conversación (descrita utilizando Atomic o Composite Process) ofrece la forma de interactuar con el servicio en términos de una secuencia de intercambio de mensajes. En OWL-S, cada Profile puede tener una descripción de la conversación descrita en el Model asociado con el Profile mediante la propiedad hasProcess. Un servicio Amigo proporciona (o requiere) varias capabilities que cada una describe una funcionalidad diferente. Por lo tanto, se introduce la propiedad hasConversation para definir conversaciones asociadas a cada Capability. La propiedad hasConversation es una propiedad funcional que declara que cada Capability tiene como mucho una conversación. La conversación global soportada por el servicio Amigo será la unión de todas las conversaciones de todas las capabilities proporcionadas. Los procesos pueden ser reutilizados en varias conversaciones que implementan interacciones similares.

La funcionalidad del servicio es ofrecida por Capability, utilizando la clase Capability del lenguaje AmIGO-S, relacionada con la clase Profile de OWL-S mediante la propiedad hasCapability. Al igual que la descripción funcional de la clase Profile de OWL-S, la descripción de cada Capability está dada por los Inputs y Outputs del servicio, las Preconditions que deben cumplirse para la ejecución del servicio y los Effects (Results) producidos en el mundo por la ejecución del servicio. Los inputs y Outputs corresponden a los mensajes que serán enviados y recibidos desde y hacia el servicio y se expresan en cualquier tipo de sistema con XML literales. Los Effects y las Preconditions se dan en una fórmula lógica, según lo estipulado por la clase Expresión de OWL-S como las expresiones DPR, KIF o SWRL. Por lo tanto la estructura, propiedades y forma de definir es la misma.

#### 1.4.4.7 Análisis de los enfoques

En las siguientes líneas son descritas las conclusiones que se han obtenido tras el análisis de los principales modelos para representación de servicios semánticos (ver Tabla 20):

- Pese a que cada propuesta emplea diferente terminología a la hora de nombrar las entradas, salidas, efectos y precondiciones (IOPEs), todos tienen la misma base. Aun así, ninguno de los modelos define un modelo ontológico para la representación de efectos y precondiciones, y simplemente mencionan los lenguajes que pueden ser empleados para representarlos: SWRL, WSML y SWSL.

- Todos los lenguajes excepto WSDL-S permiten representar múltiples precondiciones por cada unidad invocable. Sin embargo, la única precondición de WSDL-S puede ser el resultado de un conjunto complejo de expresiones booleanas. A pesar de que todos los lenguajes permiten expresar múltiples efectos. WSDL-S recomienda definir un único Effect por Operation.

- WSDL-S y SAWSDL permiten añadir información semántica a descripciones WSDL, manteniendo así la estructura y simplicidad de WSDL. Estos dos enfoques son agnósticos del lenguaje empleado para representar las ontologías, siendo posible emplear OWL, WSML u otro modelo para anotar los conceptos del servicio.

- OWL-S, WSMO, SWSO y AmIGO-S son lenguajes más complejos y expresivos. Prueba de ello es que permiten definir procesos complejos, compuestos por operaciones simples, cosa que no es posible con WSDL-S y SAWSDL.

- En OWL-S y SWSO las IOPEs son definidas a nivel de Atomic Process. Algunas tareas automáticas como el descubrimiento y la composición requieren considerar las IOPEs del comportamiento complejo que describe el proceso de modelo compuesto del servicio. Como dicho comportamiento no puede ser definido de manera formal, son inferidos desde los Atomic Processes. Pese a que AmiGO-S emplea como base OWL-S, difieren en que las IOPEs son definidas a nivel de Capability.

- En OWL-S y SWSO un servicio solo puede tener una única unidad invocable (Atomic Process o Composite Process), por lo tanto las IOPEs de esta unidad pasan a ser las IOPEs del servicio. El resultado de esto es un conjunto de IOPEs por servicio. Lo mismo ocurre en WSMO, pero en este caso las IOPEs son definidas para la única Capability que puede tener el servicio. Por el contrario, en WSDL-S y SAWSDL las IOPEs son definidas a nivel de Operation y como un servicio puede tener múltiples Operation, no existen las IOPEs de servicio. Sin embargo, en AmIGO-S un servicio puede tener varias Capability y a su vez también varios Process, se puede decir que AmIGO-S trata de integrar la expresividad de OWL-S con la simplicidad de WSDL en lo referente a las Capabilities.

Definir cuál de los lenguajes es mejor resultad difícil, ya que todo depende del entorno, problemática, tipos de servicios y arquitectura que se pretende emplear. Pero a rasgos generales, se pueden destacar los siguientes aspectos de cada uno de ellos:

- OWL-S tiene la madurez y el amplio trabajo que se ha hecho en el hasta la actualidad.

- La ventaja de SWSO es su gran expresividad, pero por el contrario la comunidad científica que está trabajando en este enfoque es muy escasa.

- WSMO también ofrece un marco completo para describir servicios, pero está más orientado a otros enfoques más empresariales que relativos a la

computación ubicua.

- AmiGO-S, siendo basado en OWL-S, tiene toda su madurez y además ha sido desarrollada con la clara intención de que sea principalmente empleada en dispositivos de entornos de computación ubicua. Pero por el contrario, la comunidad que está trabajando en ella es escasa.

- WSDL-S y SAWSDL destacan por su simplicidad, ya que no soportan la definición de procesos. Sin embargo, son enfoques que están más cerca de los servicios web sintácticos, pero añaden semántica a estos de manera muy simple. Estos enfoques, y en especial SAWSDL, principalmente debido a que ha sido propuesto por el W3C como enfoque para la descripción de Servicios Web Semánticos, hace que sea un candidato ideal como enfoque como enfoque de Servicios Web Semánticos sin proceso asociado.

Del análisis se concluye que los lenguajes para describir servicios semánticos no detallan ni describen cómo tienen que ser modelados los efectos y las precondiciones de los servicios y deja en manos de los diseñadores/desarrolladores cómo representar los efectos y las precondiciones y la relación que tienen estos con el entorno. Por lo tanto, consideramos necesaria la creación de un modelo ontológico para representar los efectos y las precondiciones de servicios que será utilizado dentro de ISMED.

**Tabla 20. Lenguajes de representación semántica de servicios vs Efectos y precondiciones.**

| | OWL-S | WSMO | SWSO | WSDL-S | SAWSDL | AMIGO-S |
|---|---|---|---|---|---|---|
| **Input** | Input | PreCondition | Input | Input | Input | Input |
| **Output** | Output | PostCondition | Output | Output | Output | Output |
| **PreCondtion** | PreCondition | Assumption | PreCondition | PreCondition | Input Condition | PreCondition |
| **Effect** | Result | Effect | Effect | Effect | Output Condition | Effect |
| **Service Implementation Technology** | Web Services. Extensible by means of Grounding | Web Services | Web Services. Extensible by means of Grounding | Web Services | Web Services | Web Services. Extensible by means of Grounding |
| **Language for ontology definition** | OWL DL | WSML | SWSL-FOL | Agnostic | Agnostic | Same as OWL |
| **Framework for service definition and development** | Protégé | WSMO Studio | Not specified | Semantic Tools for Web Services, Radian, WSMO Studio | Radiant, WSMO Studio | Not specified |
| **Maximum number of Preconditions** | Multiple | Multiple | Multiple | One for operation (recommended), it can be a complex Boolean expression | Multiple | Same as OWL |
| **Maximum number of effects** | Multiple | Multiple | Multiple | One for operation (recommended) | Multiple | Same as OWL |
| **Effect and PreCondition definition level in the service** | Process | Capability | Atomic Process | Operation | Operation | Capability |
| **Effect and PreCondition rules definition language** | SWRL, KIF, DRS | SWML | SWSL-Rules | SWRL, OCL | SWRL,WSML | Same as OWL |
| **Development level** | High | High | Low | Low | Medium | Low |
| **API implementation** | OWL-S API | SWMO4J | Not specified | WSDL4J API | Woden4SAWSDL, SWMO4J, SAWSDL4J | Not specified |

### 1.4.5 Selección de servicios

Los mecanismos de emparejamiento son dependientes del lenguaje o formalismo en el que se definen los servicios. Es por ello que en los últimos años se han definido diferentes mecanismos para el emparejamiento que han ido evolucionando desde simples servicios descritos en base a palabras clave, pasando por servicios descritos en base a Input/Ouput hasta llegar a servicios que pueden ser descritos teniendo en cuenta condiciones asociadas a las Input/Output además del cambio en el estado del mundo previo y posterior a la ejecución del servicio. Estos diferentes métodos de representación de servicios están directamente relacionados con diferentes enfoques de descubrimiento de servicios. A continuación se describen los tres principales enfoques de emparejamiento de servicios que han sido empleados en los últimos años (ver Ilustración 24):

- **Keyword-Based**: Es el enfoque más básico de descubrimiento de servicios, basado en el emparejamiento de términos simples empleando técnicas de Information Retrieval (emparejamiento sintáctico). De esta manera el conjunto de palabras clave que representa a la solicitud del servicio es emparejado con los conjuntos de palabras clave contenidas en las descripciones de servicios. Estas técnicas basadas en palabras clave son limitadas debido a la ambigüedad del lenguaje natural y su carencia de semántica. Sin embargo, su gran ventaja es la escalabilidad ya que pueden ser empleados con grandes conjuntos de servicios y además pueden emplear tecnologías maduras para el emparejamiento de palabras clave.

- **Lightweight Semantic**: Este es un enfoque más avanzado que el anterior en el que los servicios son descritos empleando conceptos abstractos que los representan. Por lo tanto, los servicios son descritos como un conjunto de objetos y el emparejamiento entre ellos se determina comprobando si los conjuntos de objetos están interrelacionados, por ejemplo, empleando para ello un marco teórico que determine las relaciones entre los objetos del servicio solicitado y los publicados, como puede ser el propuesto por Paolucci y otros [93, 181].

- **Heavyweight Semantic**: Esta basada en descripciones semánticamente más ricas que el enfoque *Lightweight* tomando en cuenta la relación existente entre

Input, Output, Efectos y Precondiciones. Esto permite que los estados de las propiedades puedan ser tomadas en consideración tanto antes como después de la ejecución del servicio. Empleando este enfoque será por lo tanto posible, determinar cuáles pueden ser los efectos que pueden derivar tras la invocación de un servicio, siendo estos dependientes de los valores de entrada del Input y del estado del entorno en ese instante.

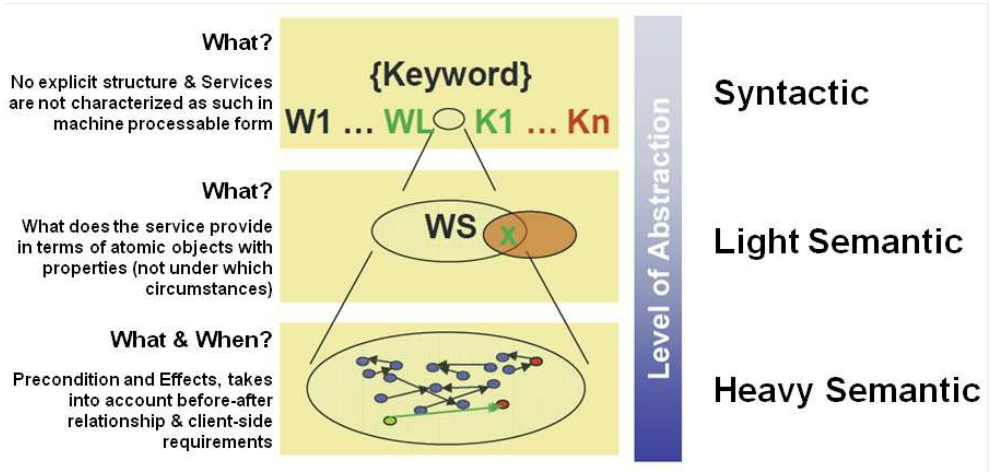


**Ilustración 24. Enfoques de descubrimiento sintácticos y semánticos, adaptado de [222]**

Basado en los formalismos descritos por los servicios semánticos, se han llevado a cabo diversos enfoques que tienen como objetivo el emparejamiento entre servicios con el objetivo de comparar la conveniencia entre el servicio requerido y el servicio publicado. La efectividad del emparejamiento de servicios depende de la expresividad de la descripción del servicio y en la determinación adecuada del grado de emparejamiento entre las descripciones de los servicios. Klusch [223] describe que los actuales enfoques de emparejamiento semánticos de servicios pueden ser clasificados en base a dos parámetros:

- **Parte del modelo de servicios es empleada para realizar el matching**. La mayoría de enfoques emplean el emparejamiento basado en el perfil del servicio (*Service Profile*). Este tipo de emparejamiento, denominado, emparejamiento de servicios "black-box" determina la correspondencia semántica entre los servicios en base a la descripción de sus perfiles. El perfil del servicio describe qué es lo que el servicio hace en términos de su signatura (entradas, salidas, precondiciones y efectos) y aspectos no funcionales (categoría, nombre, QoS, privacidad, localización del servicio, etc.) En cambio

el emparejamiento basado en el modelo de proceso (*Process Model*), denominado emparejamiento "glass-box", tiene como objetivo comparar el comportamiento operacional de los servicios en base al control de proceso y al flujo de datos de los mismos.

- **Método que se emplea para realizar el emparejamiento** La mayoría de enfoques de emparejamiento de servicios realizan emparejamiento semántico de servicios basado en lógica, pero existen otra serie de enfoques denominados no-lógicos que no emplean ningún mecanismo de razonamiento basado en lógica para calcular el grado de emparejamiento entre descripciones, para ello se emplean técnicas de cálculo de similitud sintáctica, emparejamiento basado en grafos estructurados, o calculo numérico de distancia entre conceptos de una ontología. Además existe otro grupo que aglutina a los enfoques híbridos que mezclan tanto el razonamiento lógico como el no lógico para conseguir un grado de emparejamiento más exacto.

#### 1.4.5.1 Emparejamiento basado en el modelo de servicio

En las siguientes líneas se ofrece una agrupación de los diferentes tipos de emparejamiento existentes (*Service Profile* y *Model Process*) en base al modelo de servicio empleado (ver Ilustración 25):

- **Emparejamiento basado en el modelo de perfil:** Aquí se engloban todos los enfoques relativos al emparejamiento basado en el modelo de perfil, es decir en las propiedades funcionales (entradas, salidas, efectos y precondiciones) y los no-funcionales (QoS, privacidad, etc.):

    - **Emparejamiento basado en propiedades no-funcionales**: En este tipo de emparejamiento se encontraría los enfoques que emplean las propiedades no-funcionales de los perfiles de servicios para el emparejamiento y selección de servicios. Entre las propiedades no-funcionales más empleadas destacan la calidad de servicios (QoS), tipo de interacción en el que el servicio puede ser empleado. Además en enfoques de emparejamiento para entornos de computación ubicua las propiedades no-funcionales que más se han empleado han sido las relativas al contexto y en muchas ocasiones relativas a la localización

del servicio-usuario (ver apartado 1.4.5.2). Sin embargo, creemos que el contexto está estrechamente ligado con la funcionalidad ofrecida por el servicio, ya que depende de él para invocarse correctamente y los cambios que produce éste afectan directamente al entorno.

- **Emparejamiento basado en propiedades funcionales**: Aquí se engloban los enfoques que emplean tanto las salidas, entradas efectos y precondiciones (IOPE), clasificándolas en función de cuáles de ellas es empleada para el emparejamiento. El modelo de proceso de emparejamiento aceptado por la comunidad científica [181, 224-226] describe el emparejamiento como la suma de dos subprocesos: Emparejamiento= Emparejamiento de las Signaturas + Emparejamiento de las especificaciones. Donde se puede determinar que las signaturas serían el equivalente a las entradas y a las salidas y las especificaciones las equivalentes a los efectos y precondiciones. Todas estas propiedades formarían parte de las propiedades funcionales.

- **Emparejamiento Combinado**: Este tipo de emparejamiento surge de la combinación de las propiedades funcionales así como las no funcionales, pero ello no quiere decir que todos los enfoques empleen tanto las entradas, salidas, efectos, precondiciones y propiedades no-funcionales para el matching, ya que no existe en la actualidad enfoque destacable que emplee precondiciones y efectos combinado con atributos no-funcionales. Entre los enfoque que pueden encontrarse en este grupo destacan: GSD-MM de Chakraborty [227] un mecanismo de emparejamiento basado en lógica orientado a entornos de computación ubicua y distribuidos, iMatcher1 [228] y iMatcher2 [229] de Bernstein basados en OWL-S siendo el primero no-lógico y el segundo híbrido, FC-Match [230] de Bianchini realiza un emparejamiento híbrido basado en lógica y similitud textual (empleando WordNet y el coeficiente Dice) basado en perfiles OWL-S redefinidos como expresiones OWL-DL.

- **Emparejamiento basado en el modelo de proceso**: El emparejamiento de servicio semántico basado en el modelo de proceso, es en general, poco común. Y no es tomado en cuenta por lo actuales desarrolladores de lenguajes para describir servicios web semánticos. Además, los modelos de proceso de OWL-S y WSML no están actualmente definidos de manera formal y en el caso

de SAWSDL y las descripciones de servicio monolíticas no ofrecen ni siquiera la posibilidad de definir el modelo de proceso. Sin embargo, este problema puede ser solventado parcialmente reescribiendo la descripción del modelo de proceso de manera que pueda ser descrito en un lenguaje lógico que tanga un sistema automático de verificación que disponga de herramientas para el correcto análisis del proceso [223]. Como este método de emparejamiento queda fuera de nuestro enfoque y no lo consideramos de interés no vamos a profundizar más en ello.

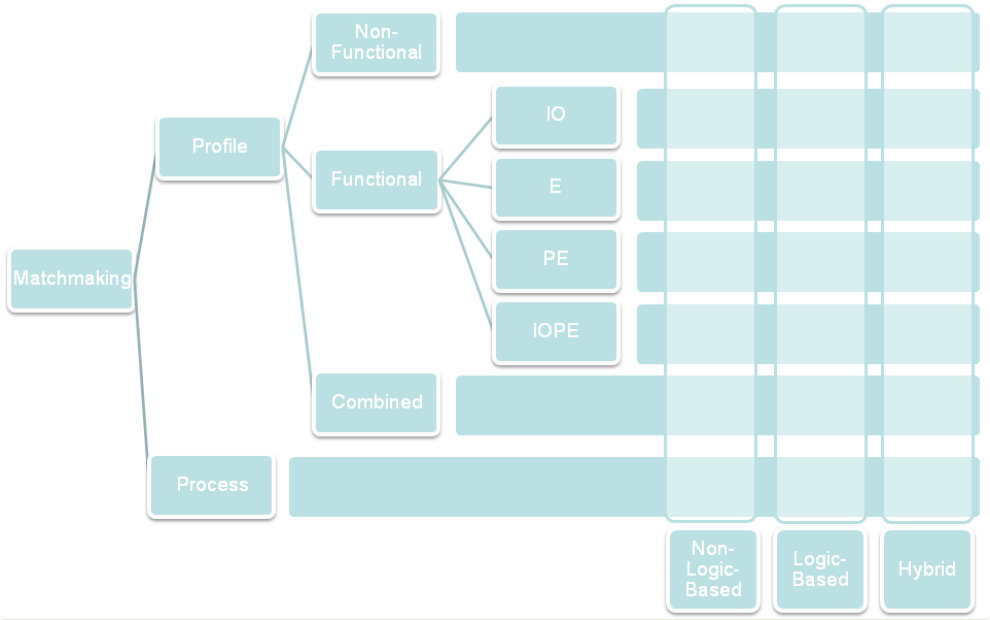


**Ilustración 25. Enfoques de emparejamiento semántico de servicios**

En las siguientes páginas se ofrece un análisis más detallado de los enfoques de emparejamiento basado en el perfil de servicio funcional, ya que es en estos enfoques en donde son empleados los efectos y condiciones. Pero además de lo anterior se mencionarán los diferentes métodos (lógico, no-lógico e híbrido) que se han empleado en cada uno de ellos.

#### 1.4.5.1.1 Emparejamiento funcional basado en Entradas y Salidas

El tipo de emparejamiento basado en lógica trata la identificación de relaciones tipo subsumption entre conceptos que describen las operaciones que realizan los servicios. Esta relación permite relacionar conceptos con conceptos más genéricos en una ontología basándose en su definición formal descrita en lógica descriptiva. Tras el

razonamiento tipo subsumption en la ontología, la estructura jerárquica resultante permite calcular los tipos de relación entre el concepto solicitado y los conceptos ofrecidos. A continuación se describirán varios enfoques que emplean lógica para el emparejamiento, seguido de varios que emplean un método no-lógico para el emparejamiento para acabar con un enfoque híbrido.

Varios esfuerzos han sido llevados a cabo en el área de emparejamiento (matching) de Servicios Web Semánticos basándose en las entradas/salidas de los servicios. Un algoritmo base para el emparejamiento basado en signaturas es el propuesto por Paolucci y otros [93, 181]. Este algoritmo permite el emparejamiento de una funcionalidad solicitada, siendo esta descrita como un conjunto de entradas ofrecidas y salidas requeridas, contra un número de funcionalidades expuestas, siendo descrita cada una como un conjunto de entradas requeridas y salidas ofrecidas. Las entradas y las salidas son semánticamente definidas empleando conceptos de ontologías. El algoritmo describe cuatro niveles de emparejamiento entre los conceptos ontológicos requeridos y los ofrecidos. Los cuatro niveles son los siguientes:

- **Exact**: Si la salida de la petición es igual a la salida del servicio registrado entonces se da este tipo de correspondencia. También existe una correspondencia exacta si el concepto de la petición es subclase del concepto del servicio registrado, pero sólo en el caso especial de que se pueda asegurar que la salida del servicio registrado proporciona todos los subtipos posibles.

- **Plug In**: Se da si el concepto de servicio registrado subsume al concepto de la petición del usuario. En este caso del concepto del servicio registrado puede usarse en el lugar del concepto de la petición.

- **Subsumes**: Si es la petición del usuario la que subsume el concepto del servicio registrado significa que el servicio no satisface completamente las necesidades del usuario. Debido a que un servicio de estas características proporciona resultados incompletos, el usuario debería realizar más operaciones para obtener los resultados deseados.

- **Fail**: Si no se da ningún caso de los anteriores el emparejamiento se considera un fallo. Es decir, la petición del usuario y la descripción del servicio registrado son incompatibles.

El grado de correspondencia total, entre las salidas de una petición de un usuario y las de la descripción semántica del servicio, se calcula como el peor de todos los resultados obtenidos al realizar las comparaciones individuales. En el caso de comparar las entradas del servicio, se realizará de la forma descrita, pero invirtiendo el orden de la comparación, es decir, se en ese caso se comparará la entrada de la notificación del servicio con la entrada de la petición del usuario. Posteriormente, el módulo de emparejamiento ordena los resultados obtenidos dando prioridad a aquellos servicios que tienen un grado de correspondencia mayor para las salidas, y solo si existe igualdad entre dos servicios utiliza el grado de correspondencia de las entradas.

Una modificación del algoritmo anteriormente descrito es planteado en [231]. La definición de grados de emparejamiento semántico es extendida para incluir un nuevo grado denominado Intersección. Por lo tanto, los grados existentes, ordenados de mayor a menor correspondencia semántica, quedan de la siguiente forma: *Exact*, *Plug In*, *Subsume*, *Intersection* y *Disjoint* o *Fail*. Mientras que en el algoritmo de Paolucci las comparaciones de los perfiles de servicio se realizan mediante la comparación semántica de cada uno de las propiedades descriptivas, en este artículo se propone la necesidad de permitir el cálculo de la correspondencia considerando el perfil de servicio como un solo concepto. Además de este algoritmo, existen otros enfoques que también se han basado en el método de emparejamiento descrito anteriormente [232, 233]. En el caso de [116] modifican el algoritmo en dos sentidos: por una parte, no es necesario emparejar todos los *Input/Output* del servicio solicitado con todos los *Input/Ouput* de los servicios disponibles. Solamente requieren que todos los *Input/Output* del servicio requerido sean emparejados con algunos o todos los *Input/Output* de los servicios disponibles. Esto hace posible que servicios que de otra manera serian descartados puedan ser emparejados. Y por otra solamente calculan los matching tipo *Exact* y *Plug In* ya que creen que con Subsumes no se puede asegurar que el servicio ofrecido que hace este tipo de matching pueda ofrecer las funcionales requeridas, siendo así el algoritmo más rápido y eficiente.

Mandell y McIlraith [234, 235] emplean el sistema de emparejamiento SDS (Semantic Discovery Service) para la selección de servicios para su posterior uso en una composición creada mediante BPEL4WS, esta composición se realiza encadenando hacia atrás los servicios con el fin de obtener una composición con las entradas y salidas deseadas, empleando para ello un enfoque lógico.

Jaeger y otros [236] proponen un sistema de emparejamiento a 4 fases, en donde las

tres primeras fases (entradas, salidas y categoría) emplean razonamiento de tipo subsumption empleando ontologías en OWL y la cuarta fase estaría orientada a un sistema que pueda emparejar requisitos que no son soportados mediante razonamiento basado en ontologías. Los autores se centran en describir las tres primeras fases detallando 4 tipos de emparejamiento: *Fail*, *Unknown*, *Subsumes* y *Equivalent*.

Constantinescu y otros [194] emplean en el sistema de emparejamiento HotBlu los grados de emparejamiento definidos por Paolucci y otros, pero además añaden uno nuevo denominado Overlap, que representa al Intersection del enfoque de Li y Horrocks [231]. Además para mejorar la eficiencia del sistema implementan una clasificación jerárquica de servicios codificados mediante intervalos, reduciendo de esta manera el problema del emparejamiento a la búsqueda de intersecciones entre estructuras rectangulares en hiperespacios. Gracias el proceso de indexación y codificación el proceso de emparejamiento pasa a ser mucho más rápido pero en cambio el proceso de inserción de servicios en los árboles con más de 10000 servicios pasa a ser un proceso muy pesado, necesitando hasta 3 segundos para publicar un nuevo servicio. Ben Mokhtar y otros [237, 238] en el sistema EASY también emplean un enfoque similar que en vez de emplear intervalos emplean codificaciones en base a números primos, por lo que en proceso de emparejamiento todavía resulta más rápido. Estos dos enfoques podrían ser considerados como enfoques de emparejamiento no-lógico ya que emplean codificaciones de intervalos y números primos para realizar el emparejamiento.

Otro enfoque de emparejamiento no basado en lógica es la infraestructura METEOR-SWSDI Discovery [214] y Lumina, el componente de búsqueda basado en UDDI. En Lumina la búsqueda es basada en palabras clave, mientras que en MWSDI el descubrimiento de servicios SAWSDL se basa en emparejamiento no-lógico empleando ello dos técnicas para el cálculo de la distancia semántica: por una parte algoritmo de emparejamiento estructural y por otra el algoritmo de emparejamiento a nivel de elemento. El algoritmo de emparejamiento de nivel de elemento calcula la similitud lingüística entre los conceptos, mientras que el algoritmo de emparejamiento estructural considera la similitud entre los sub-arboles de los conceptos para calcular la similitud. Además de Lumina existen otros dos enfoques principales que emplean descripciones de servicios basadas en SAWSDL, la primmera de ellas es SAWSDL-MX de Klusch y otros [239], el cual emplea un mecanismo híbrido de emparejamiento

al igual que lo hacen sus antecesores OWLS-MX [240, 241] y WSMO-MX [242, 243]. Y la segunda es FUSION [244], de Kourtesis y Paraskakis, el cual es un sistema de emparejamiento lógico que emplea registros UDDI. En FUSION, cualquier descripción de servicio es clasificado en el momento de su publicación y después es mapeado a UDDI para así facilitar búsquedas rápidas.

OWLS-MX [240] es hasta la actualidad el único sistema de emparejamiento híbrido para OWL-S. OWLS-MX complementa el razonamiento basado en lógica con varias métricas de similitud de IR (Information Retrieval) basados en *Token* para calcular el emparejamiento semántico entre pares de servicios. Los autores [241] además argumentan que un emparejamiento solamente basado en razonamiento de Logica Descriptiva (como son la mayoría de enfoques actuales) es insuficiente en práctica ya que un enfoque híbrido. Argumentan que la calidad del proceso de descubrimiento semántico puede mejorar considerablemente combinando de manera adecuada tanto métodos de lógica descriptiva como técnicas tipo *Information Retrieval*. En un artículo posterior [241] realizan un análisis de la calidad de los resultados obtenidos, analizando los posibles falsos positivos y falsos negativos con el objetivo de analizar los beneficios y puntos flacos que tienen tanto el emparejamiento basado en lógica como el híbrido. De este artículo se extrae que los falsos positivos/negativos pueden ser mitigados mediante medidas de solapamiento sintáctico de las métricas IR y mediante un emparejamiento híbrido integrado en conceptos de bajo nivel. Los autores resaltan que el sistema es mejor que otros enfoques puramente lógicos, pero como comentan Küster y otros en [245] no está claramente demostrado que sea mejor ya que el conjunto de servicios que se empleo para el test era completamente desarrollado por los autores y además resaltan lo siguiente: “OWLS-TC2 is suited to test and evaluate the features of this particular hybrid matchmaker...”. Por lo tanto, se puede considerar que la validación es parcial y no completa.

#### 1.4.5.1.2 Emparejamiento funcional basado en Efectos

El emparejamiento de descripciones de servicios basados en Lógica Descriptiva monolítica es llevado a cabo exclusivamente dentro de lo que se considera la teoría clásica del razonamiento en lógica descriptiva. Este tipo de emparejamiento de efectos basado en lógica es simple pero agnóstico para las estructuras impuestas por formatos de servicios web semánticos como OWL-S y WSMO. Klusch manifiesta en [223] que la investigación en este campo no ha hecho más que comenzar y que en parte esta investigación está relacionada con la investigación en razonamiento no

monotónico que se emplea en lenguajes basados en regla de la Web Semántica. Como este método de emparejamiento queda fuera de nuestro enfoque y no lo consideramos de interés no vamos a profundizar en ello.

#### 1.4.5.1.3 Emparejamiento funcional basado en Efectos y Precondiciones

El emparejamiento de especificaciones se basa en el emparejamiento de precondiciones y postcondiciones que describen la semántica operacional de los servicios. El primer trabajo de referencia relativo al emparejamiento funcional basado en efectos y precondiciones es el planteado por Zaremski y Wing en [224], donde el emparejamiento basado en especificaciones es llevado a cabo empleando técnicas de theorem proving (describiendo las precondiciones y postcondiciones mediante predicados de lógica de primer orden), por ejemplo: infiriendo reglas generales de tipo sumbsuption entre expresiones lógicas que describen las precondiciones y postcondiciones de los componentes. El autor describe varios tipos de relaciones Exact Pre/Post, Plug-in, Plug-in Post, Weak Post, Exact Predicate, Generalized y Specialized, donde las cuatro primeras son emparejamientos tipo Pre/Post y las últimas tres son emparejamientos basados en el predicado. A continuación de ofrece una descripción más amplia de cada uno de los tipos de relaciones:

- **Pre/Post Matches**: Tipo de emparejamiento en donde se relacionan por una parte precondiciones y por otra las postcondiciones de cada componente.

- **Predicate Matches**: Tipo de emparejamiento en donde se relaciona toda la especificación de predicados de los dos componentes.

Lin y Arpinar proponen en [246, 247] un mecanismo para el definición de relaciones semánticas entre Web Services, empleando para ello las precondiciones y postcondiciones de los mismos. Los tipos de relaciones que plantean están orientadas para su empleo en procesos de composición de servicios y no tanto a emparejamiento. Los autores definen dos grupos de relaciones, por una parte las relaciones entre Web Services y por otra las relaciones entre condiciones.

En este enfoque sin embargo, los autores no definen como representar las condiciones, sino que solamente mencionan que van a emplear tripletas RDF y además el mecanismo que emplean para determinar las relaciones entre condiciones solamente emplean acepta servicios con una sola precondición y una sola

postcondición. Además en este caso, de manera similar a como lo hacen en LARKS [225], los autores resaltan que las precondiciones representan condiciones asociadas a los Input y las postcondiciones representan las condiciones asociadas a los Output, sin embargo, las condiciones no solamente están relacionadas con los IO, sino que también están relacionadas con la información contextual.

Un sistema de emparejamiento más reciente es el PCEM [248], el cual está basado en lógica. Los servicios semánticos de PCEM están descritos mediante OWL-S, las anotaciones de éstos están descritos mediante ontologías en OWL y los efectos y precondiciones están descritos mediante PDDL. Sin embargo, internamente el mecanismo de emparejamiento emplea Prolog como mecanismo de representación de servicios, ontologías y efectos y precondiciones así como para el razonamiento. PCEM realiza dos tipos de emparejamiento:

- **Exacto**: Comprueba que las precondiciones del servicio solicitada emparejen de manera exacta con el publicado y lo mismo con los efectos. Para ello, comprueba si hay una posible sustitución de una variable vacía que cuando sea aplicada a una o a las dos proposiciones (representando tanto las dos precondiciones como los dos efectos) de lugar a dos expresiones equivalentes.

- **Basado en razonamiento**: El emparejamiento basado en razonamiento, es más complejo que el emparejamiento exacto. Para ello se emplean las reglas generales de inferencia de Prolog (por ejemplo, todas las reglas de deducción), además de reglas de inferencia específicas de dominio (por ejemplo, subPartOf). Para las primeras se emplea el mecanismo de razonamiento de Prolog empleando resolución y CWA (Closed World Assumption) y las últimas pueden ser específicas para ciertas precondiciones y efectos.

#### 1.4.5.1.4 Emparejamiento funcional basado en Entradas, Salidas, Efectos y Precondiciones

Un método para el emparejamiento de IOPEs es empleando el denominado query containment [225, 226, 249], para ello tanto los servicios publicados como las peticiones de los clientes son modelados mediante consultas con ciertas restricciones. Empezando con las restricciones definidas los posibles valores de ambas consultas, tanto la del servicio solicitado el servicio publicado, son evaluadas y la posible

inclusión entre los resultados de las consultas es inferida. Una consulta q1 es contenida en q2 si todas las respuestas de q1 están incluidas en q2. El sistema LARKS de Sycara y otros [225, 226], donde se comprueban las relaciones polinomiales tipo theta-subsumption de las precondiciones y postcondiciones descritas como clausulas def-Horn.

WSMO-MX [242, 243] es el único sistema híbrido de emparejamiento escrito basado en entradas, salidas, efectos y precondiciones junto con LARKS. Este sistema emplea una extensión LP (Logic Programming) de WSML-Rules, denominado WSML-MX para la descripción tanto de los servicios como metas a conseguir. El sistema adopta los principales elementos requeridos para el emparejamiento de servicios en WSML, que son, goal, service, capabilities, preconditions and postconditions, pero no effect y assumption. El sistema toma como base ideas del algoritmo hibrido de emparejamiento de OWLS-MX [240, 241], el emparejamiento basado en grafos orientados a objetos del sistema DSD-MM [112, 250] y el concepto de matching intencional de servicios propuesto por Keller [243]. Además de basarse también en general en LARKS [225, 226], pero permitiendo una parametrización más afinada combinando tanto emparejamiento sintáctico como semántico.

#### 1.4.5.2 Emparejamiento de servicios semánticos en dispositivos con recursos limitados

Los métodos más comunes de selección SDP, Salutation, SLP, SSDP y Zeroconf ofrecen métodos de emparejamiento (sintácticas) similares, pero ninguna de ellas semántica. Muchas de estas emplean patrones o palabras clave para el emparejamiento, pero este tipo de enfoque no es adecuado para entornos ubicuos, ya que uno de los mayores problemas a la hora de afrontar el problema de la riqueza y heterogeneidad de los entornos de computación ubicua es el problema de utilizar la información contextual para inferir cuales son las necesidades de los usuarios que se mueven en el entorno y localizar automáticamente los servicios más apropiados [251, 252].

Los protocolos de descubrimiento de servicios sensibles al contexto tienen como objetivo proveer a los usuarios los mejores servicios de red, basándose en sus preferencias, necesidades y condiciones de ejecución [253, 254]. En muchos de los casos se centran en la sensibilidad relativa a la localización (location awareness) [255,

256] mientras en otras consideran la información del contexto para personalizar así el proceso de selección de servicios [257, 258]. En otras en cambio emplean información relativa a la calidad de servicio (QoS) [227].

La evaluación de las propiedades del contexto debe ser llevado a cabo mediante la evaluación de reglas de contexto [259] o mediante la maximización de una función de utilidad [227], pero normalmente los enfoques solamente se centran en el emparejamiento sintáctico entre la información proveniente del cliente y la que proviene del servicio.

Se da el hecho que la mayoría de estos protocolos de descubrimiento de servicios sensibles al contexto asumen que el cliente y el proveedor de servicios emplean la misma terminología para describir la información contextual, pero tal hecho no es real en entornos de computación ubicua debido a que los diferentes componentes del entorno son diseñados, desarrollados y desplegados independientemente, es por ello que es necesario que se empleen lenguajes que permitan describir la semántica operacional de los servicios de manera adecuada con el objetivo de realizar un descubrimiento de servicios centrado en las necesidades de los usuarios. Diversos estudios han trabajado en el descubrimiento semántico y sensible al contexto de servicios en entornos ubicuos [98, 153, 237, 238], pero todavía ninguno de ellos ha empleado los efectos y precondiciones para la selección de servicios, simplemente han empleado las entradas y las salidas de los mismos e información relativa al contexto del usuario y su localización.

En la actualidad destacan dos propuestas basadas en el emparejamiento basado en signaturas (Input y Output) desarrolladas por investigadores del proyecto ARLES de INRIA Rocquencourt de reciente creación que tienen como objetivo el emparejamiento eficiente de Servicios Web Semánticos orientado a dispositivos con recursos limitados.

#### 1.4.5.2.1 EASY

EASY (Efficient SemAntic Service DiscoverY in Pervasive Computing Environments with QoS and Context Support) [260, 261] que provee de un marco para el descubrimiento semántico de servicios eficiente en entornos de computación ubicua, factor que es clave para dispositivos de recursos limitados que pueden encontrarse en los entornos de computación ubicua. Para ello codifica los conceptos de la ontología en modo offline (es decir, antes del proceso de descubrimiento) y clasifica

adecuadamente las descripciones de servicios en el repositorio basado en los conceptos codificados. El algoritmo de emparejamiento de EASY adopta (con una ligera modificación) el enfoque relativo a los niveles de emparejamiento de Paolucci y otros. Además, comparado con otros enfoques relativos al emparejamiento de signaturas el enfoque EASY no solamente soporta aspectos funcionales sino que además soporta aspectos no-funcionales como son la calidad de servicio (QoS) y las propiedades del contexto. Además ofrece medios para clasificar los servicios en función de las preferencias de los usuarios basándose en varias propiedades no-funcionales. El algoritmo resultante clasifica los servicios evaluando los niveles de emparejamiento entre los conceptos empleados en la petición de servicio y aquellos que están siendo publicados.

En el artículo de EASY realizan un estudio del tiempo necesario por los razonadores para clasificar los servicios anotados semánticamente y se concluye que el proceso es muy lento, llegando incluso a necesitar cerca de 5 segundos para parsear, clasificar y realizar el emparejamiento de conceptos. Por lo que consideran que son tiempos muy elevados para ser aplicados en entornos de computación ubicua y con el objeto de mejorar el rendimiento del proceso plantean una serie de mejoras:

- Desplazar el proceso de clasificación de online a offline.
- Codificar los conceptos de las ontologías para así reducir el problema a una simple comparación de códigos.
- Organizar y agrupar las descripciones de servicios en estructuras para aligerar el proceso de emparejamiento.

#### 1.4.5.2.2 PerSeSyn

Hasta la actualidad el emparejamiento sintáctico y el semántico han sido considerados como problemas separados, más allá, los enfoques de descubrimiento de servicios relativos a la interoperabilidad (que tiene como objetivo facilitar el descubrimiento de servicios de diferentes tipos de protocolos) se han centrado en el descubrimiento sintáctico, dejando de lado el semántico. Y al mismo tiempo los protocolos de descubrimiento de red semánticos no se han preocupado nunca de los aspectos relativos a la heterogeneidad. Y los protocolos de descubrimiento sensibles al contexto han dado por hecho que todos los clientes y proveedores emplean la misma ontología,

cosa que en la realidad no será así, ya que cada dispositivo tendrá la ontología que ha desarrollado el fabricante del mismo. Con el objetivo de integrar los problemas anteriormente descritos se plantea la plataforma de descubrimiento de servicios PerSeSyn (Pervasive Semantic Syntactic) [260, 262, 263] que tiene las siguientes características:

- Plantean un modelo y lenguaje para la descripción de servicios que abstrae y permite representar las diferentes tipos de lenguajes para describir servicios (UPnP, SLP, Servicios Web, Servicios Web Semánticos).

- El descubrimiento de servicios es interoperable, ya que permite integrar diversos tipos de servicios y protocolos diferentes, como son: UPnP, SLP, Servicios Web, Servicios Web Semánticos.

- El emparejamiento de servicios puede ser semántico, sintáctico o mixto (pueden existir servicios que tengan ciertas partes anotadas semánticamente y otras que no) dependiendo del tipo de servicio solicitado y de los servicios publicados.

- El emparejamiento puede ir desde un simple servicio descrito sintácticamente (SLP) hasta servicios semánticos con conversaciones asociadas (OWL-S).

- Plantean un mecanismo para la clasificación de los resultados de emparejamiento heterogéneo de servicios.

- El emparejamiento no solo tiene en cuenta los atributos funcionales sino que también los atributos no funcionales como son la calidad de servicios (Qos) y el contexto.

El algoritmo de emparejamiento empleado extiende el algoritmo del enfoque EASY mejorándolo y añadiendo funcionalidades para el emparejamiento semántico y de conversaciones. Para el prototipo se emplea el lenguaje SAWSDL para describir los servicios y BPEL para describir las conversaciones de los servicios y como resultados cabe manifestar que se obtiene un rendimiento del emparejamiento semántico igual o incluso menor en muchos de los casos, todo ello gracias a la codificación de las ontologías y a la agrupación de los servicios similares.

### 1.4.6 Conclusiones

De las dos últimas propuestas para emparejamiento semántico de servicios en entornos de computación ubicua se nota claramente la necesidad de disponer de algoritmos rápidos y eficaces que permitan realizar un emparejamiento lo más ajustado posible con el objeto de reducir el consumo de memoria, procesador y batería de los dispositivos con recursos limitados que conformarán el ecosistema de ISMED. Y como comenta Ben Mokhtar en [237] tanto si el emparejamiento de servicios semánticos es realizado mediante la signatura o bien mediante la especificación el factor clave en ambos (en clave de rendimiento) subyace en el rendimiento del motor de razonamiento empleado.

En los dos casos planteados el razonamiento y clasificación de ontologías es off-line, es decir, no es realizado en el momento del descubrimiento sino que es realizado al cargar los servicios, por lo que no es realmente aplicable en un entorno de computación ubicua real y dinámico donde los servicios anunciados cambian a lo largo del tiempo y mucho menos en dispositivos con recursos limitados. Además el enfoque de emparejamiento planteado en ambos es el basado en signatura, el cual no ofrece una respuesta real a los servicios existentes en entornos ubicuos, sería más adecuado poder describir el comportamiento de los servicios en base a las precondiciones que son necesarios para que pueda ejecutarse y en base a los efectos que genera la ejecución del mismo. De esta manera lo que se consigue es que tras la ejecución del servicio, cambie el estado del mundo, y no solo se ejecuten servicios que requieren y ofrecen información, como es el caso del emparejamiento basado en signaturas.

Del análisis de los enfoques de emparejamiento actuales 1.4.5.1 se extraen las siguientes conclusiones:

- El lenguaje más empleado para describir servicios es OWL-S, seguido de WSMO y de SAWSDL. Pero además de éstos existen otros lenguajes específicos y también mecanismos de emparejamiento que se basan en descripciones monolíticas de servicios descritas como un concepto simple en Lógica Descriptiva.

- El tipo de emparejamiento más empleado en base a la descripción del servicio es el basado en entradas y salidas seguido del basado en entradas, salidas, efectos y precondiciones y de los menos desarrollados el enfoque basado

exclusivamente en Precondiciones y Efectos.

- El tipo de emparejamiento más empleado en base a la técnica de emparejamiento es el basado en lógica, pero estos últimos años se está trabajando también en mecanismos híbridos para el emparejamiento, que están demostrando que un enfoque híbrido ofrece resultados más exactos que el puramente lógico.

Tal y como comenta Klusch [223], en la actualidad no hay sistemas de emparejamiento basados exclusivamente en precondiciones y efectos que empleen enfoques híbridos o no basados en lógica y ninguno aplicado a entornos de computación ubicua, por lo tanto, supone un reto implementar un mecanismo de descubrimiento/emparejamiento de servicios basados en efectos y precondiciones aplicado a dispositivos con recursos limitados ya que como comenta Ben Mokhtar [238] las soluciones y propuestas actuales de emparejamiento basado en efectos y precondiciones emplean costosas técnicas de prueba de teoremas [224], o bien emplean sistemas de consulta en bases de conocimiento centralizadas [225, 226, 242, 243] no siendo estas propuestas apropiadas para emplearlas en entornos distribuidos con dispositivos con recursos limitados. Sin embargo, antes de definir unos mecanismos de emparejamiento basado en efectos para dispositivos con recursos limitados y orientado a entornos de computación ubicua es necesario definir un mecanismo para representar los efectos y condiciones de los servicios de entornos de computación ubicua.

## 1.5 Razonamiento de dispositivos semánticos

### 1.5.1 Análisis de OWL

#### 1.5.1.1 Comparativa de OWL Lite, OWL DL y OWL Full

**Tabla 21. Ventajas y desventajas de los distintos tipos de OWL**

| | OWL Lite | OWL DL | OWL Full |
|---|---|---|---|
| **Advantages** | - Its inference capabilities are enough to do hierarchical classification using simple constraints.<br>- Reasoning over OWL Lite is efficient (computationally simple and fast). | - Computationally complete and decidable, can be computed in finite time.<br>- It models Description Logic, a decidable fragment of the first-order logic. | - Maximum expressiveness |
| **Disadvantages** | - Limited expressiveness | - Some restrictions against OWL Full: One class can not be an individual or property and one property can not be an individual or a class. Also there is a separation between object properties and data type properties | - No computational guarantees (computation has not to be finite) |

#### 1.5.1.2 Comparación de vocabulario

El siguiente gráfico explica la relación del vocabulario de OWL Lite, DL y Full. Tanto OWL DL como OWL Full tienen el mismo vocabulario mientras que OWL Lite es un subconjunto de dicho vocabulario.

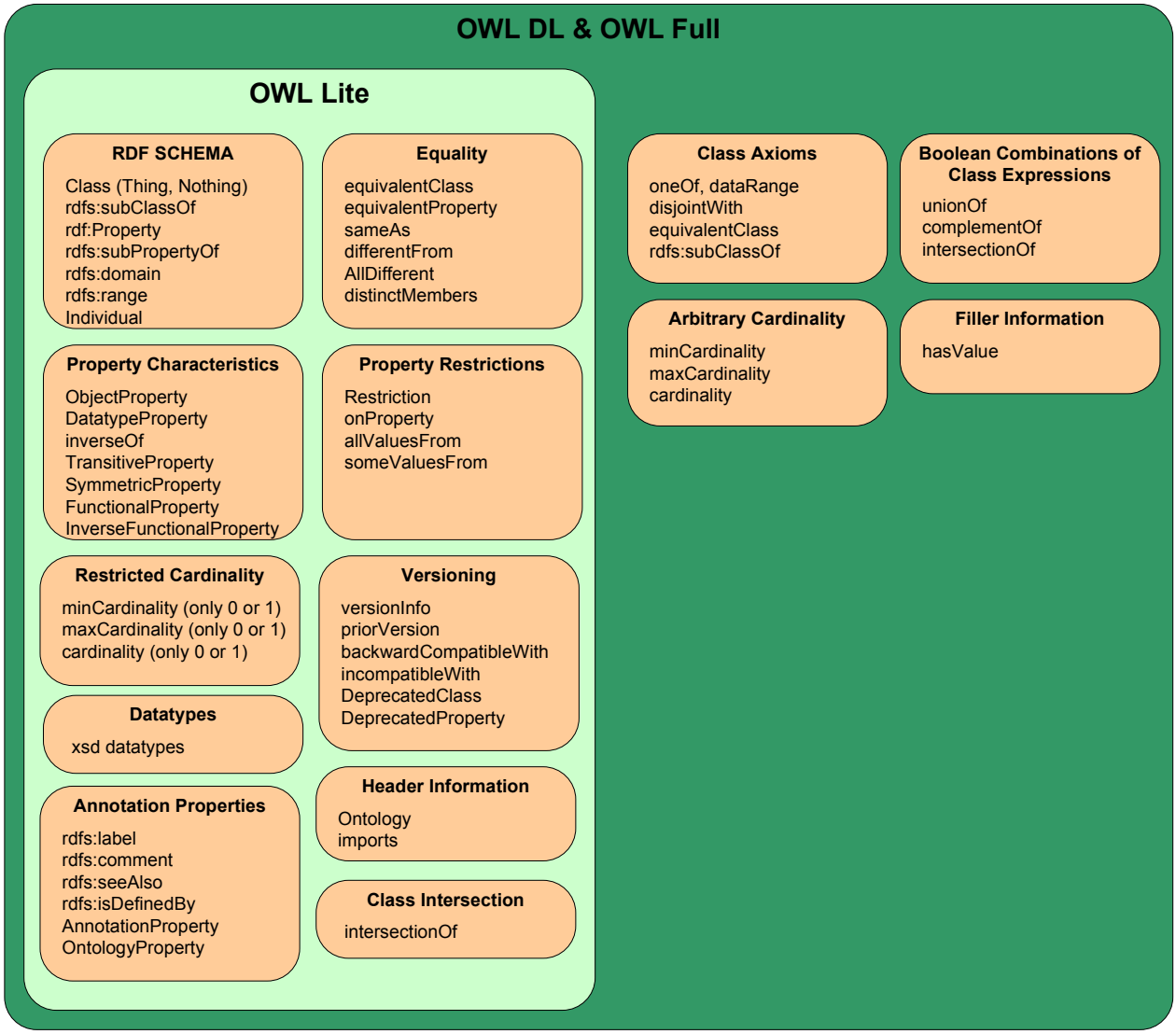


**Ilustración 26. Comparativa de los vocabularios.**

#### 1.5.1.3 Comparativa de OWL 1 y OWL 2

La siguiente información ha sido sacada de [264].

##### 1.5.1.3.1 Limitaciones en la expresividad

En lo que referente a la expresividad de OWL 1 sus inconvenientes han sido ampliamente reconocidos por la comunidad de DL, y se ha realizado una cantidad significativa de investigaciones tratando de buscar posibles soluciones a los mismos. Los resultados de dichos trabajos se incorporaron a DL SROIQ, que es estrictamente más expresivo que SHOIN.

**Tabla 22. Limitaciones en la expresividad de OWL.**

| Issue | Problem in OWL 1 | Solution in OWL 2 |
|---|---|---|
| **Qualified Cardinality Restrictions** | Cardinality restrictions cannot be qualified with classes other than owl:Thing (The following example cannot be modelled: “a medical oversight committee as a committee that consists of at least five members, of which two are medically qualified, one is a manager, and two are members of the public”). | Even while OWL 1 was being designed, it was known that QCRs could have been added to the language without any theoretical or practical problems: the resulting logics are still decidable and have been successfully implemented in practical reasoning systems. Thus, qualified number restrictions were incorporated into OWL 2 in the obvious way. |
| Relational Expressivity: Transitive propagation of properties. | Applications commonly need to model interactions that are sometimes described as one property “propagating” or being “transitive across” another. E.g. “we might want to assert that an abnormality of a part of an anatomical structure constitutes an abnormality of the structure as a whole”. | This is addressed in SROIQ and OWL 2 by the provision complex property inclusion axioms, which significantly increase the relational expressivity of the language. In particular, they provide for the propagation of one property along another. In addition to complex property inclusion |

| Issue | Problem in OWL 1 | Solution in OWL 2 |
|---|---|---|
| Relational Expressivity: Properties of properties. | The partOf relation is often defined to be transitive (if x is a part of y and y is a part of z, then x is a part of z), reflexive (every object is a part of itself), and asymmetric (nothing is a part of one of its parts). It was not included in OWL 1 because its effects in the decidability of the ontologies where unknown. | axioms, properties in SROIQ and OWL 2 can be transitive, reflexive, irreflexive, symmetric, or asymmetric, and pairs of properties can bemade disjoint. |

| Issue | Problem in OWL 1 | Solution in OWL 2 |
|---|---|---|
| **Datatype Expressivity** | OWL 1 provides very limited expressive power for describing classes whose features have concrete values such as integers and strings. Is not possible to model assertions like:<br>restrictions to a subset of datatype values (e.g., a gale as a wind whose speed is in the range from 34 to 40 knots).<br>relationships between values of concrete features on one object (e.g., a square table is a table whose breadth equals its depth).<br>relationships between values of concrete features on different objects (e.g., people who are older than their boss).<br>aggregation functions (e.g., the duration of a process is the sum of the durations of its subprocesses).<br>Another important limitation of the datatype support in OWL 1 is the lack of a suitable set of built-in datatypes. OWL 1 relies on XML Schema 11 for the list of built-in datatypes. | OWL 2 significantly extends the set of built-in datatypes of OWL 1 by reusing certain datatypes from XML Schema. Thus, OWL 2 now supports owl :boolean, owl :string, xsd:integer, xsd:dateTime, xsd:hexBinary, and a number of datatypes derived from these by placing various restrictions. In addition, xsd:decimal , xsd:double, and xsd:float will most likely be replaced with owl :real—a datatype interpreted as the set of all real numbers, providing a constant for all rational numbers. OWL 2 also provides a datatype restriction construct, which allows new datatypes to be defined by restricting the built-in datatypes in various ways. |

| Issue | Problem in OWL 1 | Solution in OWL 2 |
|---|---|---|
| **Keys** | OWL 1 does not provide means for expressing key constraints, which are a core feature of database technologies. | Extending DL-based languages such as OWL 2 with keys, however, poses both theoretical and practical problems . Therefore, the Working Group has decided to include a more restricted variant of keys that can be useful in practice as well as relatively easy to implement, commonly known as easy keys. Unlike the general keys, easy keys are not applied to individuals not known by name. |

1.5.1.3.2 Aspectos sintácticos

| Issue | Problem in OWL 1 | Solution in OWL 2 |
|---|---|---|
| **Frame-Based Paradigm** | OWL 1 Abstract Syntax is heavily based in frames. The problem is that the frames are somewhat different to Description Logic axioms, been this a source of confusion for developers. | The structure of OWL 2 ontologies has been unambiguously specified using OMG'sMeta-Object Facility (MOF). The classes of the MOF metamodel describe the canonical structure of OWL 2 |

| Issue | Problem in OWL 1 | Solution in OWL 2 |
|---|---|---|
| **Alignment with DL Constructs** | Even though OWL 1 is based on DLs, the constructs of OWL1 Abstract Syntax do not completely correspond to the constructs of DLs. | ontologies in a way that is independent of the syntax used to serialize the ontologies. The MOF metamodel for OWL 2 can be seen as analogous to the Document Object Model (DOM) specification 16 for XML. It unambiguously specifies what OWL 2 ontologies are in terms of their structure and thus makes the specification precise.<br><br>In addition to the MOF metamodel, the OWL 2 specification defines a Functional-Style Syntax as a simple encoding of the metamodel. OWL 2 departs in its conceptual design and in syntax from the OWL 1 Abstract Syntax. The main difference with respect to the OWL 1 Abstract Syntax is that the Functional-Style Syntax of OWL 2 does not contain the frame-like syntactic constructs of OWL 1; instead, ontology entities are described at a more fine-grained level using axioms. |

| Issue | Problem in OWL 1 | Solution in OWL 2 |
|---|---|---|
| **Determining the Types of Ontology Entities** | Neither Abstract Syntax nor OWL 1 RDF is fully context-fre, that is, an axiom containing a URI often does not contain sufficient information to disambiguate the type of ontology entity (i.e., a concept, a property, or an individual) that the URI refers to. To disambiguate the syntax, OWL 1 DL relies on a strict separation of the vocabulary into individuals, classes, and data and object properties. The specification of OWL 1 DL, however, does not precisely specify how to enforce this separation at the syntactic level. Thus, whereas the semantics of OWL 1 DL requires strict typing of all names, the syntax does not enforce it, which can prevent certain ontologies from being interpreted. | OWL 2 introduces the notion of declarations. All entities can, and sometimes even must, be declared in an OWL 2 ontology. OWL 2 imposes certain typing constraints on ontologies. Roughly speaking, the sets of object, data, and annotation properties, as well as the sets of classes and datatypes in an OWL 2 ontology must be mutually disjoint. |

| Issue | Problem in OWL 1 | Solution in OWL 2 |
|---|---|---|
| **Problems with OWL 1 RDF** | The vast majority of OWL 1 ontologies have been written in OWL1 RDF syntax; this syntax has, however, shown itself to be quite difficult to use in practice. The main difficulty is that RDF represents everything using triples, which specify relations hips between pairs of objects. In contrast, many OWL 1 constructs cannot be represented using triples without the introduction of new objects. | In order to provide an alternative to RDF, OWL 2 comes with a normative XML Syntax which presents a number of advantages for publishing ontologies on the Web. In particular, the XML Syntax is easy to parse and process; also, XML is a widely adopted format on the Web and it is supported by a variety of tools. Existing OWL 1 ontologies can still be used and further developed with OWL 2 implementations. Therefore, an important requirement in the development of OWL 2 was to maintain backwards compatibility with the RDF syntax of OWL 1. At the same time, a major design goal in OWL 2 was to clean up problems in the definition of OWL 1. Every OWL 1 DL ontology written in RDF is also a valid OWL 2 ontology. |

1.5.1.3.3 Metamodelado

**Tabla 23. Problemas de OWL 1 en lo que a metamodelado se refiere.**

| Issue | Problem in OWL 1 | Solution in OWL 2 |
|---|---|---|

| Issue | Problem in OWL 1 | Solution in OWL 2 |
|---|---|---|
| **Frame-Based Paradigm** | OWL 1 Abstract Syntax is heavily based in frames. The problem is that the frames are somewhat different to Description Logic axioms, been this a source of confusion for developers. | The structure of OWL 2 ontologies has been unambiguously specified using OMG'sMeta-Object Facility (MOF). The classes of the MOF metamodel describe the canonical structure of OWL 2 |

| Issue | Problem in OWL 1 | Solution in OWL 2 |
|---|---|---|
| **Alignment with DL Constructs** | Even though OWL 1 is based on DLs, the constructs of OWL1 Abstract Syntax do not completely correspond to the constructs of DLs. | ontologies in a way that is independent of the syntax used to serialize the ontologies. The MOF metamodel for OWL 2 can be seen as analogous to the Document Object Model (DOM) specification 16 for XML. It unambiguously specifies what OWL 2 ontologies are in terms of their structure and thus makes the specification precise.<br><br>In addition to the MOF metamodel, the OWL 2 specification defines a Functional-Style Syntax as a simple encoding of the metamodel. OWL 2 departs in its conceptual design and in syntax from the OWL 1 Abstract Syntax. The main difference with respect to the OWL 1 Abstract Syntax is that the Functional-Style Syntax of OWL 2 does not contain the frame-like syntactic constructs of OWL 1; instead, ontology entities are described at a more fine-grained level using axioms. |

| Issue | Problem in OWL 1 | Solution in OWL 2 |
|---|---|---|
| **Determining the Types of Ontology Entities** | Neither Abstract Syntax nor OWL 1 RDF is fully context-fre, that is, an axiom containing a URI often does not contain sufficient information to disambiguate the type of ontology entity (i.e., a concept, a property, or an individual) that the URI refers to. To disambiguate the syntax, OWL 1 DL relies on a strict separation of the vocabulary into individuals, classes, and data and object properties. The specification of OWL 1 DL, however, does not precisely specify how to enforce this separation at the syntactic level. Thus, whereas the semantics of OWL 1 DL requires strict typing of all names, the syntax does not enforce it, which can prevent certain ontologies from being interpreted. | OWL 2 introduces the notion of declarations. All entities can, and sometimes even must, be declared in an OWL 2 ontology. OWL 2 imposes certain typing constraints on ontologies. Roughly speaking, the sets of object, data, and annotation properties, as well as the sets of classes and datatypes in an OWL 2 ontology must be mutually disjoint. |

| Issue | Problem in OWL 1 | Solution in OWL 2 |
|---|---|---|
| **Problems with OWL 1 RDF** | The vast majority of OWL 1 ontologies have been written in OWL1 RDF syntax; this syntax has, however, shown itself to be quite difficult to use in practice. The main difficulty is that RDF represents everything using triples, which specify relations hips between pairs of objects. In contrast, many OWL 1 constructs cannot be represented using triples without the introduction of new objects. | In order to provide an alternative to RDF, OWL 2 comes with a normative XML Syntax which presents a number of advantages for publishing ontologies on the Web. In particular, the XML Syntax is easy to parse and process; also, XML is a widely adopted format on the Web and it is supported by a variety of tools. Existing OWL 1 ontologies can still be used and further developed with OWL 2 implementations. Therefore, an important requirement in the development of OWL 2 was to maintain backwards compatibility with the RDF syntax of OWL 1. At the same time, a major design goal in OWL 2 was to clean up problems in the definition of OWL 1. Every OWL 1 DL ontology written in RDF is also a valid OWL 2 ontology. |

Versionado e importado

**Tabla 24. Versionado e importado.**

| Issue | Problem in OWL 1 | Solution in OWL 2 |
|---|---|---|
| **Naming problems** | OWL 1 provides a basic mechanism that allows an ontology to import another ontology, and thus gain access to all entities and axioms in it. However, the two ontologies are kept physically separate. The importing ontology is required to contain a URI pointing to the location of the imported ontology, and this location must match with the name of it. However in many realistic use cases, the location of an ontology does not correspond with its name. | The import mechanism of OWL 2 Is also "by name and location." The main difference to OWL 1 is that OWL 2 allows implementations to provide a suitable redirection mechanism: when requested to access an ontology from a location u, a tool can redirect u to a different location v and retrieve the ontology from there. The accessed ontology should, however, be treated just as though it had been retrieved from u. For example, relative URIs in the ontology should be resolved as if the ontology had been retrieved from u, and the ontology URI, if present, should be equal to u. The design of concrete redirection mechanisms is left entirely to OWL 2 implementations. OWL 2 also provides a simple method for the management of ontology versions. In addition to an ontology URI, an OWL 2 ontology can contain a version URI, which identifies the version of the ontology. In such cases, the ontology URI identifies an ontology series—a set of all versions of a particular ontology—whereas the version URI identifies a particular version in the series. |

1.5.1.3.4 Anotaciones

**Tabla 25. Anotaciones.**

| Issue | Problem in OWL 1 | Solution in OWL 2 |
|---|---|---|
| **Inadequate annotation system** | The annotation system of OWL 1 has been found to be inadequate for many applications. In particular, OWL 1 does not allow axioms to be annotated, which can be necessary | In OWL 2 it is possible to annotate axioms as well as ontologies and entities. This can be used to capture additional information about axioms, such as their provenance or certainly values. |

1.5.1.3.5 Semánticas OWL

La existencia de dos semánticas para OWL 1 (semántica directa de modelado teórico basada en la semántica de SHOIN(D) y el modelo teórico RDF) ha causado muchos problemas:

**Tabla 26. Semánticas OWL**

| Issue | Problem in OWL 1 | Solution in OWL 2 |
|---|---|---|
| **OWL 1 DL Semantics** | The OWL 1 DL specification provides an explicit (and quite complex) definition of the semantics, which does not straightforwardly correspond to SHOIN(D). This caused a number of problems regarding presentation and understandability. A source of confusion is the ability to use unnamed individuals in OWL1 facts (this feature was inspired by RDF blank nodes). Unnamed individuals are not directly available in SHOIN(D), and it is not obvious how to simulate them. | OWL 2 defines a clean direct model theoretic semantics that clearly corresponds to the description logic SROIQ(D). At the moment, OWL 2 does not provide an RDF style semantics. Therefore, in OWL 2, the transformation of ontologies in Functional-Style Syntax into RDF graphs is a purely syntactic process: the triples obtained by the RDF serialization are not intended to be interpreted directly, and the meaning of the RDF semantics on them does not define the meaning of the corresponding OWL 2 ontology. |
| **RDF-Compatible Semantics** | OWL 1 ontologies written as RDF graphs can be interpreted using an extension of the RDF semantics. This semantics, however, has proved to be extremely complex to understand, use, and implement. | |
| **Relating the DL and the RDF Semantics** | A key idea behind the two semantics of OWL 1 was that they should be “equivalent” on OWL 1 DL and OWL 1 Lite ontologies. Maintaining this compatibility becomes increasingly problematical with increasing expressive power. | |

1.5.1.3.6 OWL 1 Full

**Tabla 27. OWL Full.**

| Issue | Problem in OWL 1 | Solution in OWL 2 |
|---|---|---|
| **Undecidabilty of RDF graphs** | Each RDF graph is a syntactically valid OWL 1 Full ontology. The basic reasoning problems for the RDF-compatible semantics are undecidable. no complete implementation of OWL 1 Full currently exists, and it is not clear whether OWL 1 Full can be implemented in practice. | See 3.8 |

1.5.1.3.7 OWL 1 Lite

**Tabla 28. OWL1 Lite**

| Issue | Problem in OWL 1 | Solution in OWL 2 |
|---|---|---|
| **Computational complexity** | Even though OWL 1 Lite seems to be much simpler than OWL 1 DL, most of the complexity of OWL1 DL can be captured due to implicit negations in axioms. In fact, of all the OWL 1 DL constructors, only nominals and cardinality restrictions with cardinality larger than one cannot be captured in OWL 1 Lite. From a computational perspective, basic reasoning problems are only slightly less complex for OWL 1 Lite than for OWL 1 DL. | The EL++ Profile has been designed to allow for efficient reasoning with large terminologies. The main reasoning service of interest is classification, the problem of computing the subclass relation between all the classes in an ontology. Reasoning in EL++ can be implemented in time that is polynomial in the size of the ontology. In order to achieve tractability, EL++ disallows the use of negation, disjunction, AllValuesFrom restrictions and cardinality restrictions.<br>The DL-Lite Profile has been designed to allow for efficient reasoning with large amounts of data structured according to relatively simple schemata. The main reasoning service in the DL-Lite profile is conjunctive query answering. DLLite forbids the use of disjunction and AllValuesFrom restrictions, as well as certain other features that require recursive query evaluation.<br>OWL-R is intended to be used by OWL 2 users who are willing to |

| Issue | Problem in OWL 1 | Solution in OWL 2 |
|---|---|---|
| | | trade some of the available expressivity in OWL 2 for the ability to implement reasoning using rule systems based on forward chaining. OWL-R allows for most constructs of OWL 2; however, to allow for rule-based implementations of reasoning, the way these constructs can be used in axioms has been estricted. These restrictions ensure that a reasoning engine only needs to reason with the individuals that occur explicitly in the ontology. Thus, in OWL-R DL it is not possible to use SomeValuesFrom restrictions on the right-hand side of a subclass axiom, as this would imply the existence of an “anonymous” individual—that is, an individual that is not explicitly known by name. Similar principles have been followed in the design of Description Logic Programs (DLP). |

1.5.1.3.8 Validación de tipos OWL

La validación de tipos OWL (species validation) Es el problema de determinar cuándo una ontología de OWL 1 es OWL 1 Lite, OWL 1 DL u OWL 1 Full.

**Tabla 29. Validación de tipos OWL.**

| Issue | Problem in OWL 1 | Solution in OWL 2 |
|---|---|---|
| **Differentiation between OWL DL and OWL Lite** | Determining whether an ontology in RDF is an OWL 1 Lite or an OWL 1 DL ontology becomes very hard in practice since it involves 'guessing' ontologies in Abstract Syntax and checking whether their normative transformation into RDF yields precisely the original ontology. | OWL 2 specification identifies several profiles, trimmed down versions of the full language that trade some expressive power for efficiency of reasoning, tool developers can easily use the corresponding grammars to create tools for checking which profile an ontology belongs to. |
| **Species Validation and Imports** | Imports can interact with OWL 1 species in a quite unpredictable and unintuitive way, changing the "species" of the importing ontology. | |

### 1.5.2 Motor de inferencia

A la hora de diseñar el motor de inferencia que regirá el funcionamiento del módulo de razonamiento de ISMED se presentan tres opciones:

1. Tener un motor semántico y un motor de reglas separados.

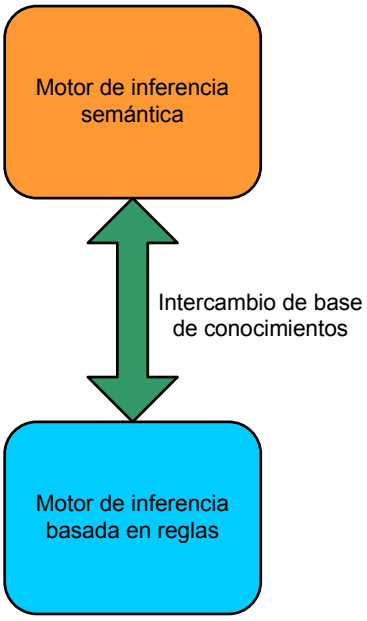


**Ilustración 27. Motor de reglas y semántico separados.**

2. Embeber el motor semántico dentro del motor de reglas.

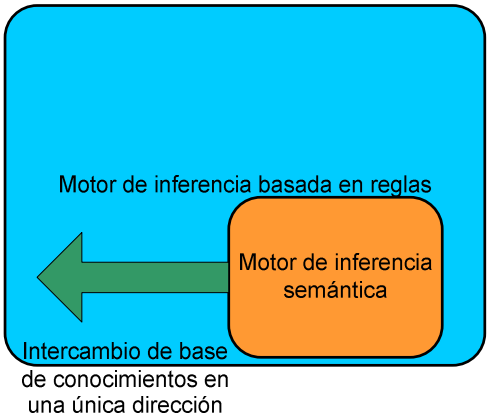


**Ilustración 28. Motor semántico embebido.**

3. Expresar la semántica de RDF y OWL como reglas y sólo utilizar el motor de reglas.

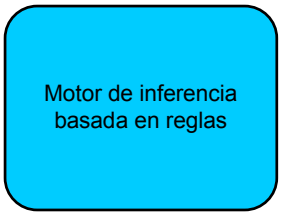


**Ilustración 29. Un único motor de reglas.**

Para decidir entre una de las tres opciones se han analizado las ventajas e inconvenientes de cada una de ellas, que se representan en la siguiente tabla:

**Tabla 30. Comparación entre diseños de motores de inferencia.**

| Motor | Ventajas | Inconvenientes |
|---|---|---|
| **Motores separados** | El motor semántico puede ser un motor externo como Pellet [265] o Racer [266], por lo que la inferencia semántica sería rápida y eficiente. | Habría que implementar manualmente el intercambio de la base de conocimiento entre ambos motores. Este intercambio sería necesario cada vez que la base de conocimiento sea modificada de alguna manera, lo que disminuiría el rendimiento del sistema. |
| **Motor embebido** | El intercambio de la base de conocimiento se haría de manera automática. | El intercambio de conocimiento sólo podría hacerse en una dirección, por lo que el motor semántico no podría observar los cambios hechos en la base de conocimiento por el motor de reglas. La eficiencia baja al estar un motor embebido en otro. |

| Motor | Ventajas | Inconvenientes |
|---|---|---|
| **Sólo motor de reglas** | No es necesario intercambio de la base de conocimiento. Se puede especificar que reglas semánticas que se deben tener en cuenta, pudiendo eliminar las que no sean necesarias en el sistema y aumentando el rendimiento. | Deben de especificarse las reglas semánticas manualmente. |

Analizando la tabla se ha llegado a la conclusión que el diseño más adecuado para el sistema es el tercero, ya que habrá cambios frecuentes en la base de conocimiento que serán provocados tanto por la inferencia de las reglas como por la inferencia semántica. Además el posible inconveniente de la tercera opción de diseño se convierte en este caso en una ventaja ya que permite seleccionar que reglas semánticas va a implementar el sistema. De esta manera no se implementaran aquellas reglas que la ontología diseñada no vaya a utilizar por lo que el número de comprobaciones que realizará el motor será menor.

El motor de reglas utilizado podría ser el propio motor de reglas incorporado en JENA, siempre que se pueda utilizar desde dispositivos empotrados y móviles, ya que tiene una serie de características que resultan necesarias para el sistema:

- Puede trabajar con una base de conocimientos expresada como una ontología.
- Permite expresar las reglas en formato de triples: (?a esUn Mamífero) -> (?a esUn Animal).
- Permite encaminamiento hacia delante, hacia atrás e híbrido.
- Permite introducir nuevos predicados en la ontología (la base de conocimiento) de manera sencilla.

El motor realizará tres labores de inferencia que se dividirán en dos fases. En una primera fase el motor extraerá el conocimiento implícito en la ontología y procesará sus relaciones semánticas para expandir la base de conocimientos. Después, haciendo uso de esta base de conocimientos expandida inferirá las acciones a llevar a

cabo por el sistema.

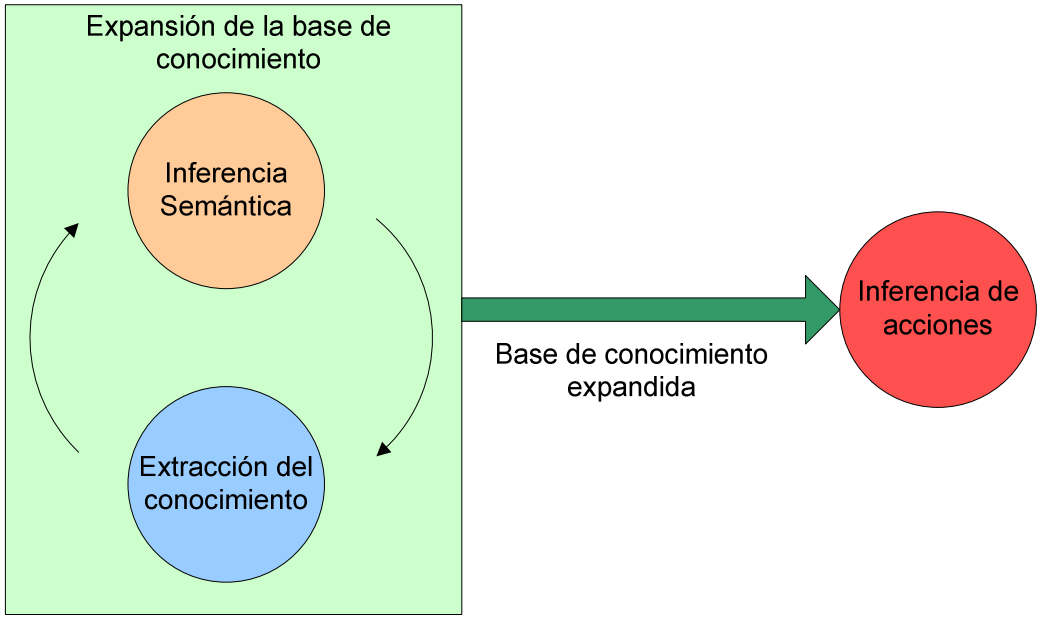


**Ilustración 30. Proceso de inferencia.**

### 1.5.3 Comparativa de razonadores

En la siguiente tabla se muestra una comparación de los razonadores más usados del mercado. Para construir la tabla se han tenido en cuenta las siguientes características:

- Expresividad: la expresividad de la lógica de descripción permitida.
- OWL-DL Entailment: la capacidad para realizar el entailment lógico necesario para permitir la teoría de modelos OWL-DL.
- Revisión de la consistencia: la capacidad para verificar la consistencia de la ontología.
- Soporte DIG [267]: índica si provee una interfaz de acceso al razonador de lógica de descripciones.
- Soporte SPARQL: permite usar el lenguaje de consultas SPARQL.
- Soporte de reglas: permite ejecutar reglas usando la ontología como base del conocimiento.

- Documentación disponible: algunos de los razonadores analizados no tienen ninguna documentación disponible.

- Versión: versión analizada del razonador.

- Licencia: licencia del razonador.

**Tabla 31. Comparativa de razonadores.**

| | Pellet [265] | KAON2 [268] | RacerPro [266] | Jena [269] | FaCT++ [270] | SweetRules [271] | Bossam [272] | Hoolet [273] |
|---|---|---|---|---|---|---|---|---|
| **Expressivity** | SROIQ(D) | SHIQ(D) | SHIQ | Incomplete DL support | SROIQ(D) | Not Applicable | ? | ? |
| **OWL-DL Entailment** | Yes | Yes | Yes | Incomplete. Jena does not implement a complete OWL DL model | Yes | No | Partial (still under development) | Yes |
| **Consistency checking** | Yes | ? | Yes | Incomplete. Jena does not implement a complete OWL DL model | ? | No | ? | Yes |
| **DIG Support** | Yes | Yes | Yes | Yes | Yes | No | No | No |
| **SPARQL Support** | Yes | Yes | No | Yes | No | No | No | No |
| **Rule Support** | Yes (SWRL -- DL Safe Rules) | Yes (SWRL) | Yes (SWRL) | Yes (Own rule format) | No | Yes (SWRL, RuleML, Jess) | Yes (SWRL & own rule format) | Yes (SWRL) |
| **Documentation available** | Yes | ? | Yes | Yes | ? | Yes | Yes | ? |

| | Pellet [265] | KAON2 [268] | RacerPro [266] | Jena [269] | FaCT++ [270] | SweetRules [271] | Bossam [272] | Hoolet [273] |
|---|---|---|---|---|---|---|---|---|
| **Version** | 2.0 RC1 | ? | 1.9.2 | 2.5.4 | 1.1.8 | 2.1 | 0.9b45 | ? |
| **Licensing** | Free/ open-source & Non-Free/ closed-source | Free/Closed Source | Non-Free/ closed-source | Free/ open-source | Free/ open-source | Free/ open-source | Free/ closed-source | Free/ open-source |

#### 1.5.3.1 Soporte SWRL

Algunas de las plataformas analizadas permiten el uso de SWRL [10] (Semantic Web Rule Language). SWRL es parte de una propuesta del W3C que combina OWL con RuleML. A pesar del soporte de SWRL, no todos los razonadores permiten todas las características que esté incorpora al usar reglas. Existen ocho tipos de built-ins en la especificación SWRL: Built-Ins para comparaciones, Built-Ins matemáticos, Built-Ins para valores boleanos, Built-Ins para cadenas de texto, Built-Ins para fechas, tiempo y duración, Built-Ins para URIs, Built-Ins para listas and Built-Ins personalizadas.

**Tabla 32. Comparativa del grado de soporte de SWRL por parte de los motores de razonamiento**

| | Pellet [265] | KAON2 [268] | RacerPro [266] | SweetRules [271] | Bossam [272] | Hoolet [273] |
|---|---|---|---|---|---|---|
| **SWRL Math Built-Ins** | No | Yes | No | Yes | Partial | No |
| **SWRL String Built-Ins** | No | Yes | No | Yes | Partial | No |
| **SWRL Comparison Built-Ins** | Partial | Yes | No | Yes | No | No |
| **SWRL Boolean Built-Ins** | No | Yes | No | Yes | No | No |
| **SWRL Date, Time and Duration Built-Ins** | No | Yes | No | Yes | No | No |
| **SWRL URI Built-Ins** | No | Yes | No | Yes | No | No |
| **SWRL Lists Built-Ins** | No | Yes | No | No | No | No |
| **SWRL custom Built-Ins** | Partial | Yes (Programmed in Java) | No | ? | Yes (Programmed in Java) | No |

# 2 DISEÑO DEL SISTEMA

## 2.1 Módulo de aprendizaje y razonamiento semántico

### 2.1.1 Alternativas seleccionadas

Dadas las características descritas en los epígrafes anteriores, a continuación se enumerarán los dispositivos hardware y tecnologías software que se tomarán como punto de partida para desarrollar el módulo de modelado y coordinación semántica.

#### 2.1.1.1 Plataformas empotradas elegidas

A continuación se enumeran las plataformas empotradas para las que se va a desarrollar el middleware ISMED. Es prematuro indicar si será posible implementar en ellas todos los módulos funcionales que componen ISMED. En los casos en los que no se pueda realizar tal implementación por falta de recursos computacionales se utilizará una arquitectura híbrida en la que elementos de la red ofrezcan de mayor capacidad computacional asistan al middleware desplegado en las siguientes plataformas:

- Gumstix
- Sunspot
- Motas

#### 2.1.1.2 Dispositivos móviles

Aparte de las plataformas de sistemas empotrados mencionadas en el apartado anterior también se hará uso de teléfonos móviles. Éstos son dispositivos de uso común que formarán parte del ecosistema de dispositivos cooperativos que pretende coordinar el middleware de ISMED.

- Nokia Series 60 soportando CLCD 1.1 Java ME. Concretamente se hará uso de dispositivos Nokia N96. Es posible que en una fase posterior del proyecto se considere también la adopción de dispositivos PDA con perfil CDC.

#### 2.1.1.3 Librerías y tecnologías

A continuación se enumeran varias librerías en Java, lenguaje de programación que principalmente se utilizará en ISMED, que hemos identificado para la implementación de ISMED.

- Jxta

  Se ha escogido por tratarse de un protocolo bien definido con versiones para distintas plataformas, que permite gestionar redes P2P con un alto nivel de abstracción.

  En concreto, se usarán las siguientes versiones:

  - JXTA-JXSE 2.5, se trata de la implementación principal de Jxta realizada en Java, que será usada en aquellas plataformas empotradas que lo permitan y en las máquinas que sirvan de intermediarias para dispositivos demasiado simples.

  - JXTA-JXME 2.1.3, es la versión para dispositivos móviles que usen Java ME, tanto CDC como CLDC.

- Repositorios semánticos

  - Sesame, para aquellas plataformas empotradas que permitan usar Java y en las máquinas que sirvan de intermediarias para dispositivos demasiado simples. Se ha escogido Sesame, por su mayor “simplicidad” frente a otros repositorios analizados.

- Microjena

  Se usará para expresar la información semántica en los dispositivos con Java ME.

- Motores de inferencia limitados

  Crearemos nuestro propio motor de inferencia apoyándonos en el trabajo realizado en el grupo sobre el MiniOWLReasoner, tesis de Juan Ignacio Vázquez. Si entretanto surgiera, de momento no nos consta, otro motor de

inferencia más completo para entornos empotrados y programado en Java, consideraríamos su adopción.

#### 2.1.1.4 Reutilización de proyectos analizados

De los proyectos analizados, TripCom tiene una arquitectura demasiado grande que no encaja nada bien con las dimensiones más reducidas y localizadas de la red que se establecerá entre los dispositivos de ISMED. Además, no existe una versión lo suficientemente madura a día de hoy.

Por otro lado, tsc++ tiene un enfoque más sencillo que hace uso de los distintos super-nodos de las redes Jxta para comunicar entre si distintas redes locales, pero aún así parece consumir muchos recursos.

Por ello, pese a no reutilizar el propio tsc++, si parece que puede resultar un buen punto de partida, y podrá ser usado a modo de referencia, adaptándolo a las mayores limitaciones impuestas por la naturaleza de dispositivos y mecanismos de comunicación dentro de ISMED. Sería conveniente en una primera iteración hacer que los dispositivos empotrados gobernados por el middleware de ISMED pudieran cooperar con aquellos desarrollados para TSC++.

#### 2.1.1.5 Primitivas a implementar

En base a los tipos de primitivas sobre triple space contemplados en otros proyectos, en ISMED se ha optado por implementar las siguientes (explicadas en el epígrafe 1.1.2.3.4.1):

- Write
- Read
- Take
- Query
- Advertise

- Subscribe
- Notify

Estas primitivas permiten a toda aplicación que se apoye en el módulo de modelado y coordinación semántica de ISMED funcionar tanto en modo PUSH (mediante las primitivas Advertise, Subscribe y Notify) como PULL (mediante las primitivas Write, Read, Query, Take).

### 2.1.2 Arquitectura propuesta

En base a lo indicado se pretenden usar las siguientes librerías en los siguientes tipos de hardware:

- Móviles CLDC: J2ME, microjena (maneja tripletas y filtrado básico), jxme-con proxy y repositorio normal (si es que fuese necesario guardar cosas que no son suyas en memoria)
- PDAs CDC: J2ME, jxme-proxyless
- Sunspot, los sunspots se comunicarán con su base mediante sockets de forma que la aplicación desplegada haga de nodo edge de JXSE.
- Motas, deberán comunicarse con una estación base conectada a un ordenador que será el verdadero nodo de la red Jxta y tendrá un repositorio semántico.
- Gumstix: JXSE y Sesame.
    - Podría incluso hacer de proxy para los dispositivos móviles que usen la versión de JXME con proxy.
- Además, cabe destacar que en la medida de lo posible sería interesante prescindir de superpeers “rendezvous”, dado que ralentizan la comunicación entre los peers que en principio estarán en la misma red local. Es decir, a diferencia de proyectos como TSC++ eliminaremos toda dependencia con infraestructura centralizada que sea posible.

A modo de resumen sobre lo comentado previamente, se muestra un esquema de la

arquitectura propuesta y las tecnologías a aplicar en cada uno de los dispositivos seleccionados. Esta arquitectura es el punto de partida para el diseño del resto de módulos que conforman el middleware ISMED: descubrimiento, composición, razonamiento y aprendizaje. Obsérvese que dadas las limitadas capacidades computacionales de las plataformas empotradas consideradas, al menos en una primera fase, tendremos que apoyarnos en diversas pasarelas que permitan la intermediación entre el mundo TCP/IP usado por los protocolos de coordinación adoptados en ISMED y los mundos de redes puras ad-hoc Zigbee usados por plataformas como Gumstix o SunSPOT. En esas pasarelas con mayor capacidad computacional podrá razonarse, aplicar algoritmos de aprendizaje e incluso en muchas ocasiones guardar en modo persistente el estado semántico generado por los distintos dispositivos que conforman el ecosistema de ISMED.

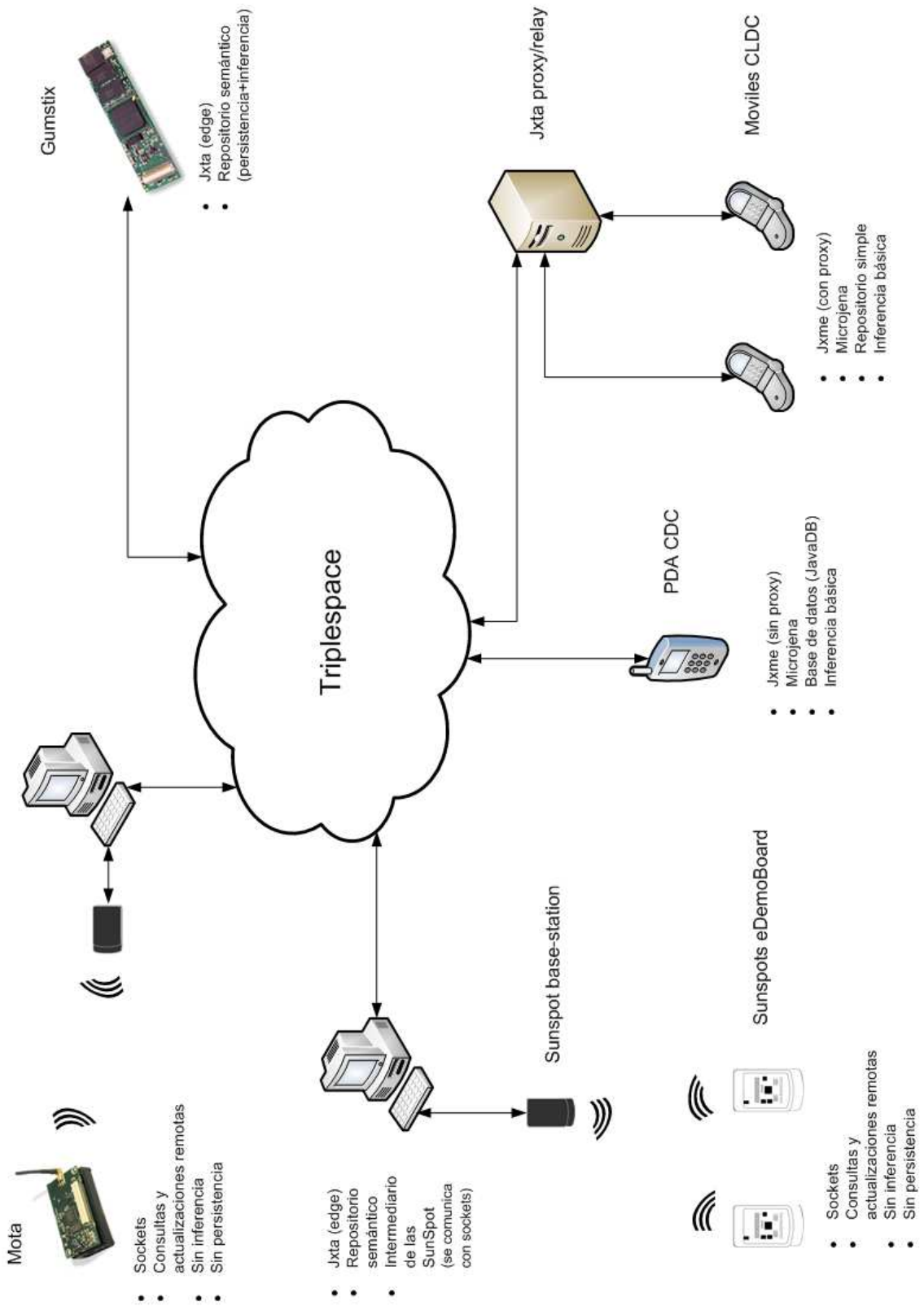


**Ilustración 31. Esquema de la arquitectura propuesto.**

### 2.1.3 Interrelación capa de coordinación semántica y módulos de descubrimiento, composición, razonamiento y aprendizaje

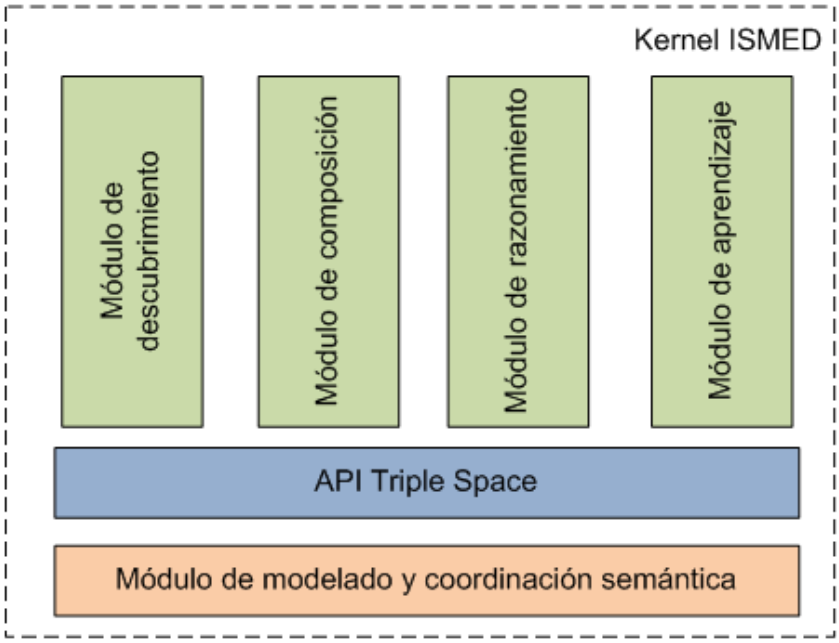


**Ilustración 32. Esquema de las capas que compodrán el kernel ISMED.**

Los módulos de descubrimiento, composición, razonamiento y aprendizaje van a mediar con el módulo de modelado y coordinación semántica a través de la API siguiendo el paradigma Triple Space Computing que se ha mencionado en 2.1.1.5.

Tal API permitirá al módulo de descubrimiento formular consultas por medio de las primitivas PULL que provee sobre servicios en el entorno. Asimismo, el módulo de descubrimiento podrá también registrarse para recibir notificaciones sobre la aparición de nuevos servicios.

El módulo de composición de servicios se apoyará en el módulo de descubrimiento para encontrar candidatos de servicios que puedan emparejarse. Además, se apoyará en módulo de razonamiento para poder inferir nuevas asociaciones entre servicios.

El módulo de razonamiento permitirá a los dispositivos equipados con middleware ISMED tener capacidad de reactividad, para de modo proactivo adaptarse a las nuevas circunstancias del entorno que les rodea.

El módulo de aprendizaje tendrá por misión identificar nuevas reglas de comportamiento del entorno y la optimización de las reglas actualmente existentes,

que nutran el módulo de razonamiento.

La cooperación y entendimiento entre los diferentes módulos de ISMED requiere **la existencia de una ontología común entre ellos**, es decir, un vocabulario que permita describir los servicios exportados por los diferentes dispositivos del ecosistema de ISMED. Además, estos servicios habrán de estar anotados para que los módulos de razonamiento y aprendizaje sean capaces de interrogarlos para obtener conocimiento sobre el que razonar y aprender. Nuestra primera tarea del segundo ejercicio del proyecto será acordar esta ontología.

En conclusión, la API de Triple Spaces que proporciona el módulo de modelado y coordinación semántica constituye la infraestructura de consulta, coordinación y comunicación que puede usarse en la implementación de los demás módulos. La ontología de servicios que será definida como primer entregable del segundo ejercicio de ISMED será la *lingua franca* que permita el entendimiento entre los objetivos dispares pero complementarios de los módulos de descubrimiento, composición, razonamiento y aprendizaje que conforman ISMED.

#### 2.1.3.1 Ontologías base para ISMED

Esta sección describe brevemente un par de ontologías desarrolladas tanto por UD como MGEP que servirán de base para la ontología de servicios a definir en ISMED.

##### 2.1.3.1.1 Smartlab

Para dar soporte a la base de conocimiento del sistema que pretendía dicho proyecto de la Universidad de Deusto, se decidió utilizar una ontología que aportaba las siguientes ventajas al sistema:

1. La base de conocimiento se encuentra expresada en unos lenguajes estándar (XML/RDF/OWL) que pueden ser procesados por otras aplicaciones.

2. Una ontología puede extenderse fácilmente con nuevas entidades sin que ello afecte a las entidades que existen anteriormente.

3. Varias ontologías pueden combinarse de manera sencilla siempre que exista una entidad en común por lo que otras aplicaciones podrían expandir la base

de conocimiento del sistema, ampliándolo y mejorándolo de esta manera.

4. Las relaciones semánticas expresadas en la ontología permiten hacer explícito el conocimiento implícito de una manera sencilla.

La ontología fue diseñada partiendo del estudio de anteriores sistemas de Inteligencia Ambiental semánticos. Se identificarón tres elementos clave dentro de una ontología que modelaba el contexto: el dónde, el cuándo y el quién. Por ello los elementos principales de esta ontología son el tiempo, el espacio y los actores (personas, agentes o dispositivos).

Después de estudiar las ontologías existentes para modelar el entorno y delimitar el alcance del sistema a desarrollar se definierón los siguientes elementos base:

- *TimeItem*: Los elementos que permiten modelar el tiempo dentro del sistema.
- *SpatialItem*: Los elementos que permiten modelar el espacio dentro del sistema.
- *LocableItem*: Elementos a los que se les puede asignar una localización dentro del sistema.
- *Event*: Un evento. En este caso un evento es un hecho que sucede en un tiempo dado y que puede que además esté asociado a una localización.

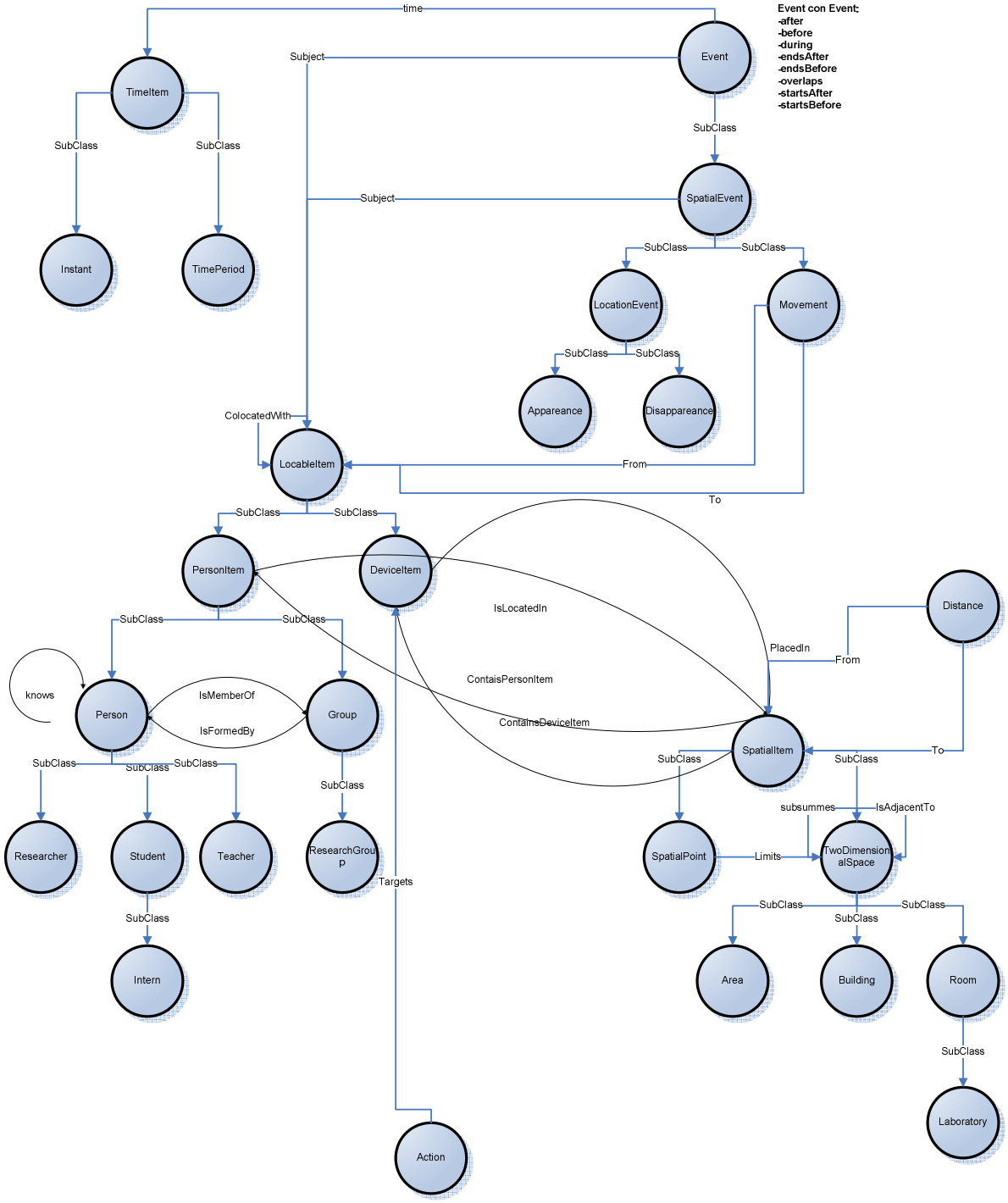


**Ilustración 33. Esquema de la ontología.**

### *2.1.3.1.1.1 TimeItem*

Los elementos hijo de *TimeItem* permiten que la ontología modele el tiempo. Esto permitirá identificar cuales son las relaciones temporales entre distintos eventos, pudiendo ordenarlos cronológicamente y facilitando la planificación de los mismos. El

tiempo puede definirse de dos maneras. Como un instante mediante la clase *Instant*:

```
<owl:Class rdf:about="#Instant">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >A time instant</rdfs:comment>
    <owl:disjointWith rdf:resource="#TimePeriod"/>
    <rdfs:subClassOf rdf:resource="#TimeItem"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="value"/>
        </owl:onProperty>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
```

O como un periodo delimitado por dos instantes (*end* y *start*) mediante la clase *TimePeriod*:

```
<owl:Class rdf:ID="TimePeriod">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="TimeItem"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
<owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="end"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
<owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="start"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
```

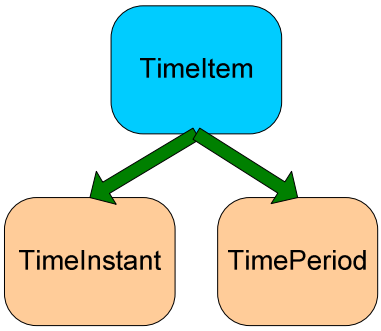


**Ilustración 34. Jerarquía de TimeItem.**

*2.1.3.1.1.2 SpatialItem*

Los elementos hijo de *SpatialItem* permiten que la ontología modele el espacio. Gracias a las relaciones espaciales el sistema podrá identificar que entidades se encuentran en la misma localización, calcular distancias y planificar rutas de movimiento. La jerarquía de *SpatialItem* es más compleja que la de *TimeItem*, teniendo dos elementos principales: los puntos y los espacios bidimensionales.

Los puntos representados por la clase *SpatialPoint* tienen propiedades para indicar su posición X e Y. Además tienen las siguientes propiedades que permiten modelar relaciones espaciales:

- *Limits*: Indica si forman parte de los limites de algún espacio bidimensional.
- *Contains*: Indica si alguna entidad está situada en ese punto.
- *subsummedBy*: Indica si el punto esta situado dentro de algún espacio bidimensional.

Los espacios bidimensionales se dividen en áreas, edificios y habitaciones. A su vez existe un tipo especializado de habitación, los laboratorios. Los espacios bidimensionales tienen las siguientes propiedades:

- *isAdjacent*: indica junto a que otros espacios se encuentra.
- *isLimitedBy*: indica los puntos que limitan el espacio.
- *contains*: indica que entidades contiene.

- *subsumes*: indica que otros espacios se encuentran dentro de él.
- *subsummedBy*: indica si se encuentra dentro de otro espacio.

Además de estas propiedades comunes los laboratorios tienen la propiedad *hasResearchers* que permite indicar que investigadores están asignados al mismo.

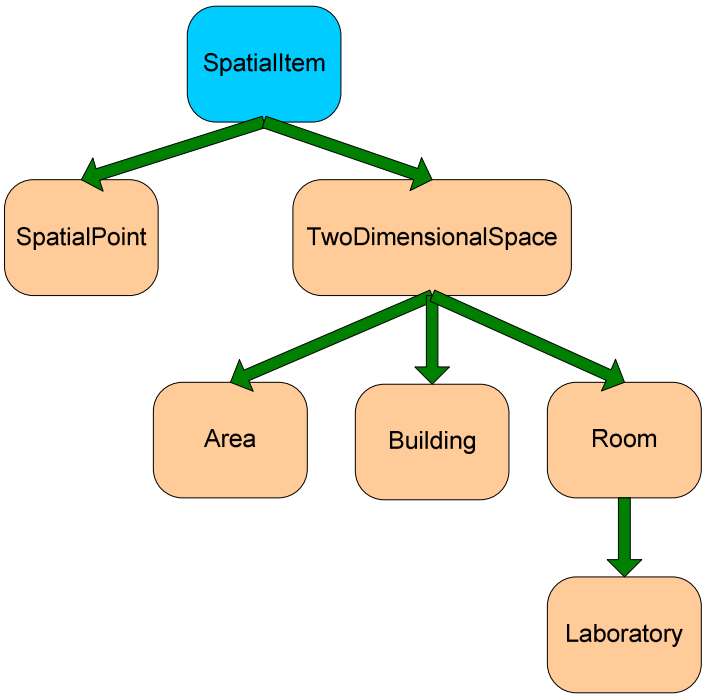


**Ilustración 35. Jerarquía de SpatialItem.**

*2.1.3.1.1.3 LocableItem*

Los elementos hijo de LocableItem son aquellas entidades que tienen una localización espacial. Se dividen en dos categorías PersonItem (entidades relacionadas con las personas) y DeviceItem (dispositivos).

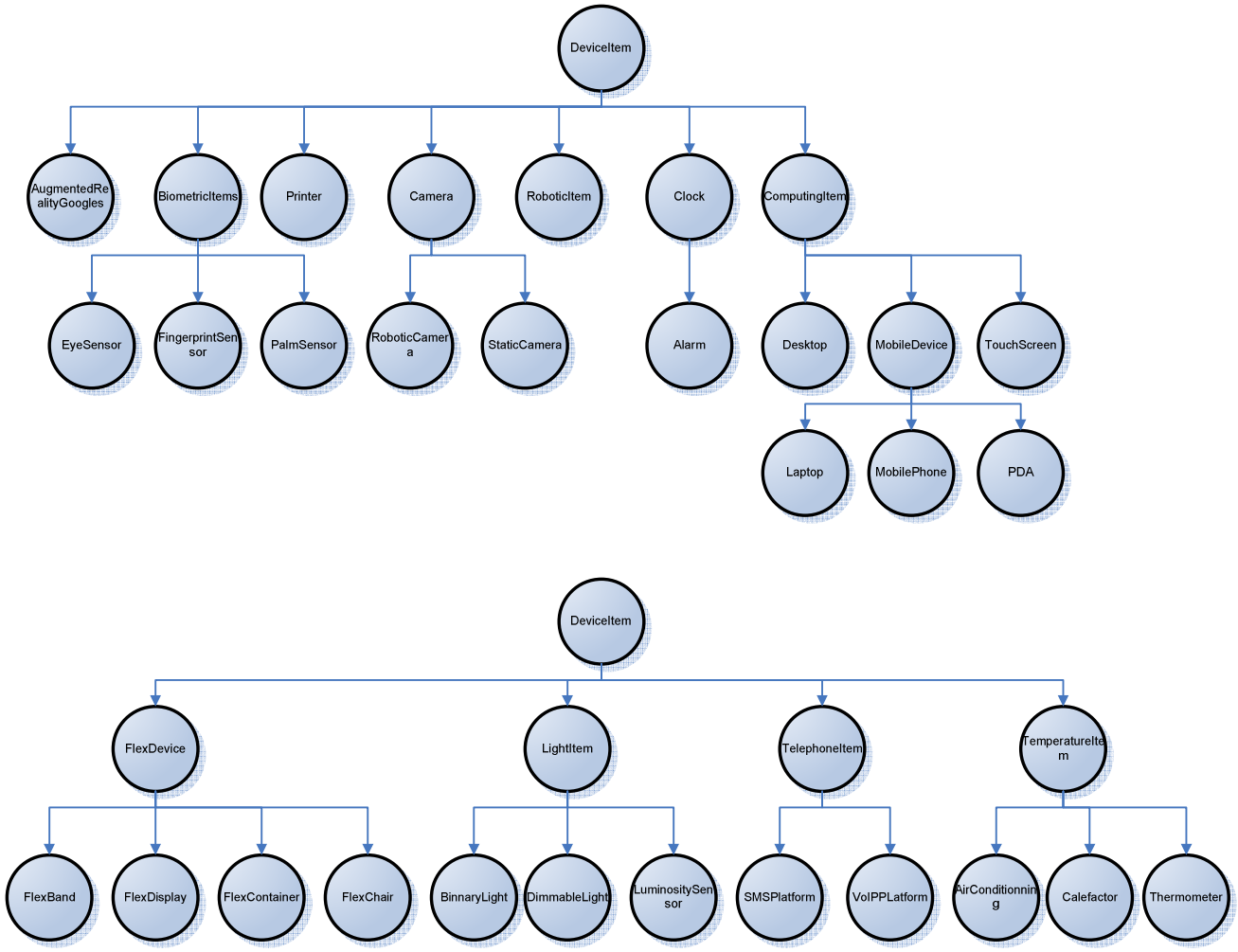


**Ilustración 36. Taxonomía de dispositivos.**

DeviceItem contiene todos los tipos de dispositivos que pueden existir en el sistema: dispositivos domóticos, cámaras, sensores, actuadores, etc… Los dispositivos tienen las siguientes propiedades:

- *deviceType*: Tipo del dispositivo.
- *Name*: nombre del dispositivo.
- *ID*: identificador del dispositivo.
- *PlacedIn*: indica el lugar donde se encuentra.
- *ColocatedWith*: indica que otras entidades se encuentran en el mismo lugar.

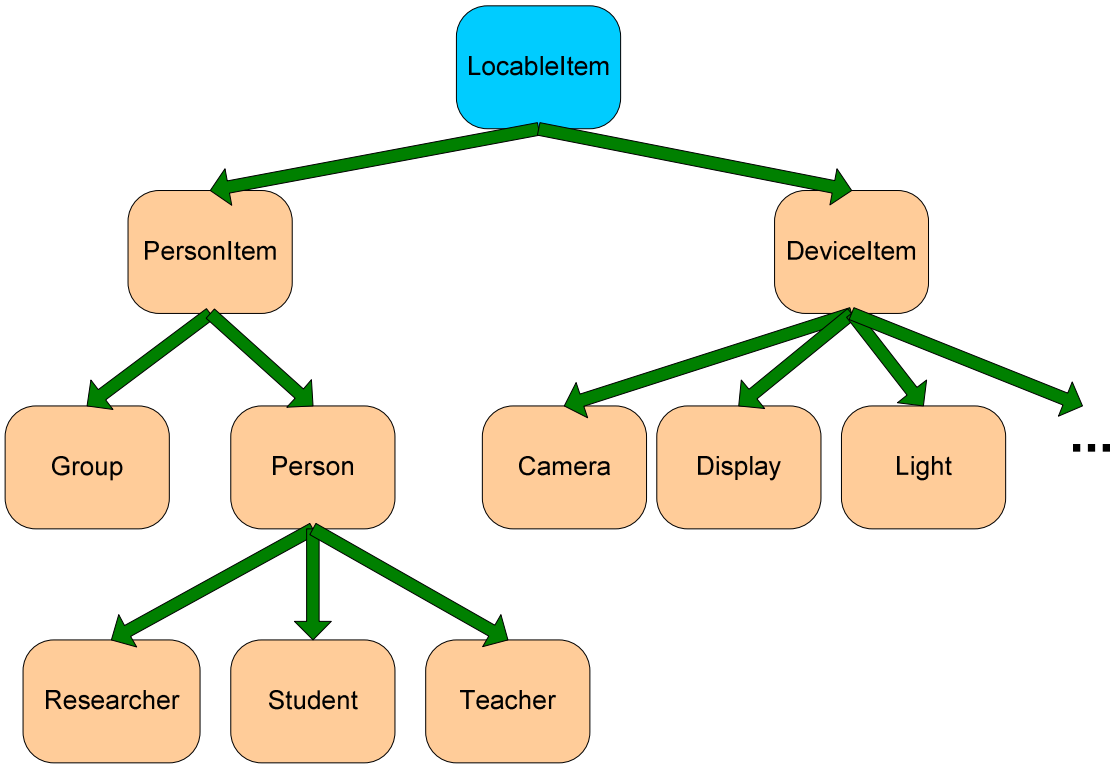


**Ilustración 37. Jerarquia de LocableItem.**

PersonItem se divide a su vez en dos categorías: grupos y personas. Un grupo esta formado por una serie de personas y tiene las siguientes categorías:

- *Name*: nombre del grupo.
- *IsFormedBy*: Personas que forman el grupo.
- *PlacedIn*: indica el lugar donde se encuentra. Todas las personas del grupo deben estar en el mismo lugar para que esta propiedad tenga algún valor asignado.
- *ColocatedWith*: indica que otras entidades se encuentran en el mismo lugar.

Las personas tienen una serie de propiedades que sirven para especificar sus datos (nombre, apellidos, dirección, teléfono, etc…). Además de las propiedades con los datos personales tienen también las siguientes:

- *IsMemberOf*: indica si es miembro de algún grupo.
- *Kwons*: indica a que otras personas conoce

- *PlacedIn*: indica el lugar donde se encuentra
- *ColocatedWith*: indica que otras entidades se encuentran en el mismo lugar.

Existen a su vez tres tipos especializados de personas: Profesores, estudiantes e investigadores.

#### *2.1.3.1.1.4 Event*

Las clases hijo de *Event* permiten modelar los sucesos del sistema. Un evento siempre esta ligado a un *TimeItem* (ya sea un instante o un periodo de tiempo) y puede tener asociado un *SpatialItem*. La clase *Event* tiene las siguientes propiedades:

- *After*: indica que sucede después de otro evento.
- *Before*: indica que eventos suceden antes.
- *During*: indica que eventos suceden dentro del periodo de tiempo del evento.
- *EndsAfter*: Indica que otro evento delimitado por un periodo acaba después.
- *EndsBefore*: Indica que otro evento delimitado por un periodo acaba antes.
- *Overlaps*: indica que el periodo de tiempo del evento se solapa con otros eventos.
- *StartsAfter*: Indica que otro evento delimitado por un periodo empieza después.
- *StartsBefore*: Indica que otro evento delimitado por un periodo empieza antes.
- *Time*: el *TimeItem* del evento.

Como se puede ver cuando varios eventos suceden en periodos de tiempo en vez de en instantes las relaciones temporales entre ellos pueden ser bastante complejas.

Además existe un tipo especializado de evento, el SpatialEvent. Este tipo de evento tiene una propiedad *location* que permite indicar donde ha sucedido. A su vez hay dos tipos especializados de *SpatialEvent* que tienen que ver con la localización de las diferentes entidades del sistema. No es realista esperar que todas las entidades

tengan una posición estática que no cambie a lo largo del tiempo. Esto puede ser así con luces o cámaras de vigilancia pero es de esperar que las personas, dispositivos móviles, microbots, etc… se muevan por el edificio. Los eventos *Appareance*, *Disappareance* y *Movement* permiten dejar constancia de estos movimientos.

El evento *Movement* indica cuando un *LocableItem* se ha movido de una posición a otra. El evento *Appareance* indica que un *LocableItem* a aparecido en el sistema y que anteriormente no se conocía su localización (si se conociera sería un movimiento). El evento *Disappareance* indica que un *LocableItem* con localización conocida ya no se puede ser encontrar.

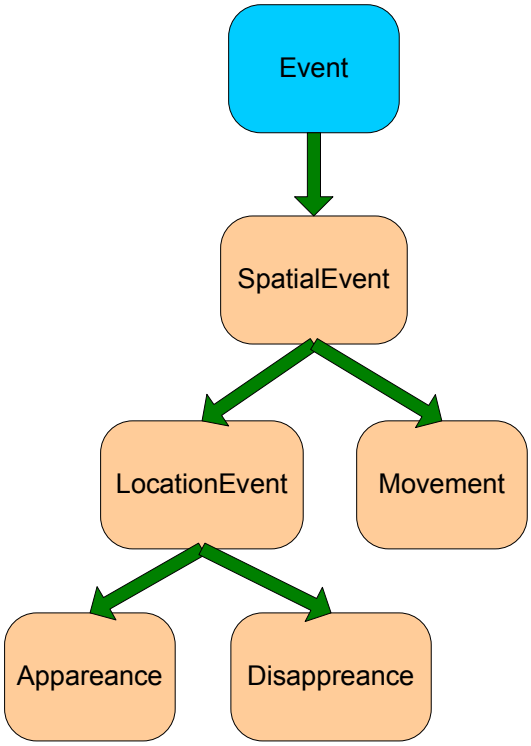


**Ilustración 38. Jerarquía de Event.**

#### 2.1.3.1.2 Ontología MGEP

En el proyecto realizado por MGEP, se describió de manera formal los entornos de computación ubicua. Este tipo de entornos están plagados de dispositivos que ofrecen servicios proporcionan mecanismos para modificar el entorno gracias a los efectos que producen en él.

Estas entidades necesitan interoperar entre ellas para conseguir ciertos objetivos.

Para hacer que las entidades sean sensibles al contexto (context-aware) el modelo que se presentó cumplía con los siguientes requisitos:

- Interpretable por maquinas: El modelo de conocimiento tenía que ser fácilmente intercambiado entre los distintos dispositivos a través de redes heterogéneas. Por lo tanto, debía ser interpretable por maquinas.

- Basado en semántica: El modelo de efectos y condiciones tiene que tener una semántica bien definida para que entidades de distintos entornos pudiesen entenderla e interoperar entre ellos de manera adecuada.

- Reusable: El conocimiento representado en el modelo de efectos y condiciones tenía que ser reusable para así reducir la información que tenía que ser transferida y procesada.

- Extensible: Diferentes sistemas de computación ubicua/pervasiva empleaban diferentes modelos de conocimiento específicos del dominio en el que son empleados. El modelo tiene que ofrecer extensión para representar los diferentes modelos de conocimiento específicos del dominio.

- Razonamiento lógico: El modelo de efectos y condiciones definido tiene que soportar inferencia basada en lógica formal para verificación y razonamiento.

Para poder desplegar sistemas sensibles al contexto resultaba necesario disponer de un modelo que representasé toda aquella información contextual necesaria, representando de esta manera el estado del mundo del entorno mediante entidades y sus relaciones.

Con tal fín, se diseño una ontología que se sustentaba en cinco conceptos básicos: entorno, servicios, efectos, condiciones y elementos.

##### *2.1.3.1.2.1 Entorno*

Un entorno está formado por entidades y sus relaciones.

*2.1.3.1.2.1.1 Entidades*

Todo entorno inteligente de computación ubicua está compuesto por un número finito de Entidades que pueblan el mismo y que pueden tener capacidades de sensorización y actuación sobre el entorno.

Según se demostró en trabajos relacionados con el modelado de contexto y analizado por Strang y Linnhoff-Popien en [274], los enfoques orientados a ontologías son muy adecuados para modelizar información de contexto como espacios, personas, objetos, etc.

El trabajo desarrollado por MGEP no se centraba en una definición detallada de las diferentes entidades del contexto, y se tuvo en cuenta que este modelo pudiese ser extendido y enriquecido con nuevas entidades con el fin de representar las características y necesidades específicas de los diferentes dominios de computación ubicua

Sin embargo, se propuso un modelo básico para la representación de las principales entidades del entorno, las cuales son empleadas como base para la definición del modelo de servicios basado en efectos. Esté modelo definía que un entorno inteligente de computación ubicua estaba formado por un número finito de Space, LivingBeing y Thing.

- *LivingBeing = (Person, Vegetal, Animal)*, aquí se encuentran representadas todas aquellas entidades tales como individuos y grupos que se benefician de las funcionalidades ofrecidas por *Thing*.

- *Object = (Resource) = (Source, Device)*, aquí se encuentran representadas el resto de las *Entity* del entorno, como pueden ser objetos físicos, computadoras, etc. La principal característica de estas *Entity* es que podrán tener la capacidad de ofrecer servicios y ser invocados. La diferencia entre *Source* y *Device* estriba en que el segundo de ellos se ubica en el entorno de manera física (por ejemplo, un reproductor multimedia), si embargo, el primero puede estar accesible, pero se desconoce su naturaleza (por ejemplo, un servicio web accesible a través de internet).

- *Space*, en este grupo se encuentran representadas aquellas *Entity*, que relacionadas con los dos grupos anteriores, permiten representar la

localización de las mismas. Estas *Entity* pueden representar localizaciones tales como habitaciones, edificios, etc.

*2.1.3.1.2.1.2 Relaciones*

Cada una de las Entity del entorno inteligente está representada por un conjunto de información que describe las características principales de dicha entidad, denominadas Relation. Este tipo de relaciones pueden ser simples o complejas.

- Las relaciones simples son aquellas que están representadas mediante datos de tipo básico, como son: *Integer*, *String*, *Float*, *Boolean*, *Time*, *Date*, *DateTime*, etc., siendo posible representar que la temperatura de una habitación es de 30ºC, la humedad del 50%, cual es la música que se está reproduciendo, producto que se está cocinando, volumen del televisor, etc.

  Además, las *SimpleRelation* tienen dos atributos que definen cual es el rango de valores válidos para la *SimpleRelation*, mediante un valor máximo (*hasMax*) y otro mínimo (*hasMin*).

- Las relaciones complejas representan las relaciones de *Entity* a *Entity*, pudiendo representar de esta manera la relación entre una persona y un dispositivo móvil. Tal y como lo define Dey en [275], estas relaciones pueden ser clasificadas en cuatro categorías principales:

  - *Identity*: cada identidad tiene un identificador único.

  - *Location*: la posición de una entidad, proximidad, etc.

  - *Status* (o actividad): relativo a las propiedades intrínsecas de una *Entity* (temperatura y luminosidad de una habitación, procesos en activo en un dispositivo en concreto, etc.).

  - *Time*: empleado para representar situaciones con mayor precisión, ordenar eventos, etc.

*2.1.3.1.2.2 Servicios*

En los entornos inteligentes no solo hay entidades que cambian sus relaciones a lo largo del tiempo, sino que es posible encontrar entidades que son capaces de modificar las relaciones de otras entidades (es decir, entidades que realicen cambios sobre el entorno, como por ejemplo: lámparas, televisores, reproductores multimedia, etc.).

Este tipo de entidad (*Entity*) se denominó *Resource*, y se caracteriza porque ofrece la capacidad de poder invocarse remotamente gracias a la arquitectura SOA que se emplea para ello. Cada *Resource* es capaz de ofrecer más de un servicio (*Service*) y éstos a su vez pueden ofrecer varias funcionalidades (*Capability*) que pueden ser invocadas para realizar diferentes tareas.

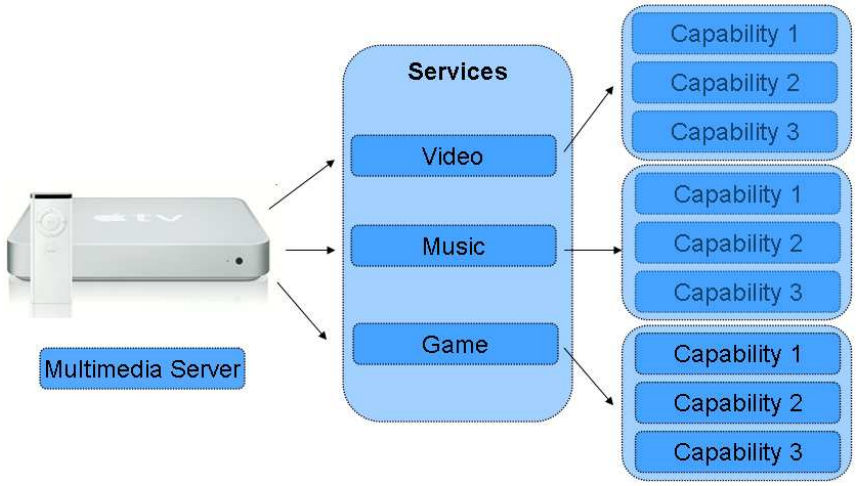


**Ilustración 39. Representación de Services y Capabilities.**

*2.1.3.1.2.3 Efectos*

Los efectos (*Effect*) representan el cambio del estado del mundo que se produce tras la correcta invocación de un servicio (*Service*). Los *Effect* ofrecidos por una *Capability* pueden variar en función de las condiciones presentes en el entorno (contexto) y de la información introducida (en los Inputs) a la hora de invocar a la *Capability*.

De esta manera, tanto la condición que definirá si un *Effect* se da o no, así como el valor del propio *Effect* estarán en función del estado del entorno de la *Entity* en la que se ejecuta la *Capability* y del *Input*.

Existen dos tipos de efectos en base a la naturaleza de los mismos (ver Ilustración 40):

- Efecto directo (*DirectEffect*). Este tipo de *Effect* está directamente relacionado con el dispositivo (*Device*) que ofrece el *Service* (en el caso de *setPosición* de una persiana, el cambio de posición de la misma).

- Efecto indirecto (*IndirectEffect*). Aquí se engloban el resto de *Effect* que son derivados del *DirectEffect*, es decir, aquellos cambios indirectos que se producen gracias al DirectEffect (siguiendo el ejemplo anterior, los *IndirectEffect* producidos por el *Service* Persiana serían el aumento del consumo eléctrico del hogar y el aumento de la luminosidad de la habitación).

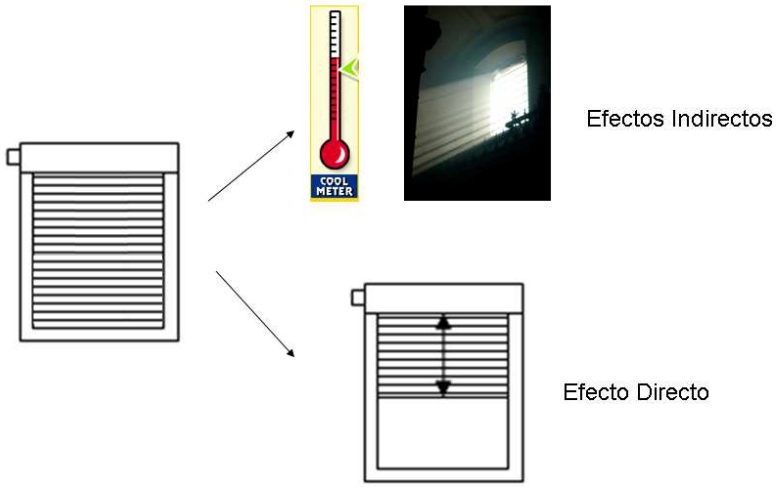


**Ilustración 40. Efectos indirectos y directos del movimiento de una persiana.**

*2.1.3.1.2.4 Condiciones*

Los enfoques de modelado de servicios y modelado de contexto existentes plantean emplear efectos y precondiciones para representar el comportamiento del servicio. Así existen precondiciones, condiciones directas e indirectas.

- PreCondition: Representan el conjunto de condiciones que deben cumplirse para que el servicio pueda ser invocado de manera adecuada.

- DirectCondition: Representa el conjunto de condiciones asociadas a un efecto directo, definiendo las condiciones necesarias para que se den los efectos directos.

- IndirectCondition: Representan el conjunto de condiciones indirectas que representan las condiciones para que se de un efecto indirecto.

#### *2.1.3.1.2.5 Elementos*

La naturaleza de las condiciones (*PreCondition*, *DirectCondition* e *IndirectCondition*) y los efectos (*DirectEffect* e *IndirectEffect*) hace que se puedan representar empleando la misma estructura, es decir, mediante la construcción *Element*. De esta manera será posible representar mediante un *Element* tanto una *PreCondition* que evalua que en la última media hora la temperatura haya subido 5ºC, como que un *Effect* represente que el *Effect* de la *Capability* invocada en el *Service* produzca un cambio de la temperatura de 5ºC en media hora. La construcción *Element* puede ser tanto un *SimpleElement* como una *CompositeElement*.

- *SimpleElement*: Sirve para representar el tipo de *Element* más simple el cual a su vez puede representar los tres tipos de condiciones o efectos descritos anteriormente.

- *CompositeElement*: Es empleado para representar conjuntos de elementos (*Element*), es decir, para representar conjuntos de elementos que estén representados en base a *SimpleElement* y *CompositeElement*.

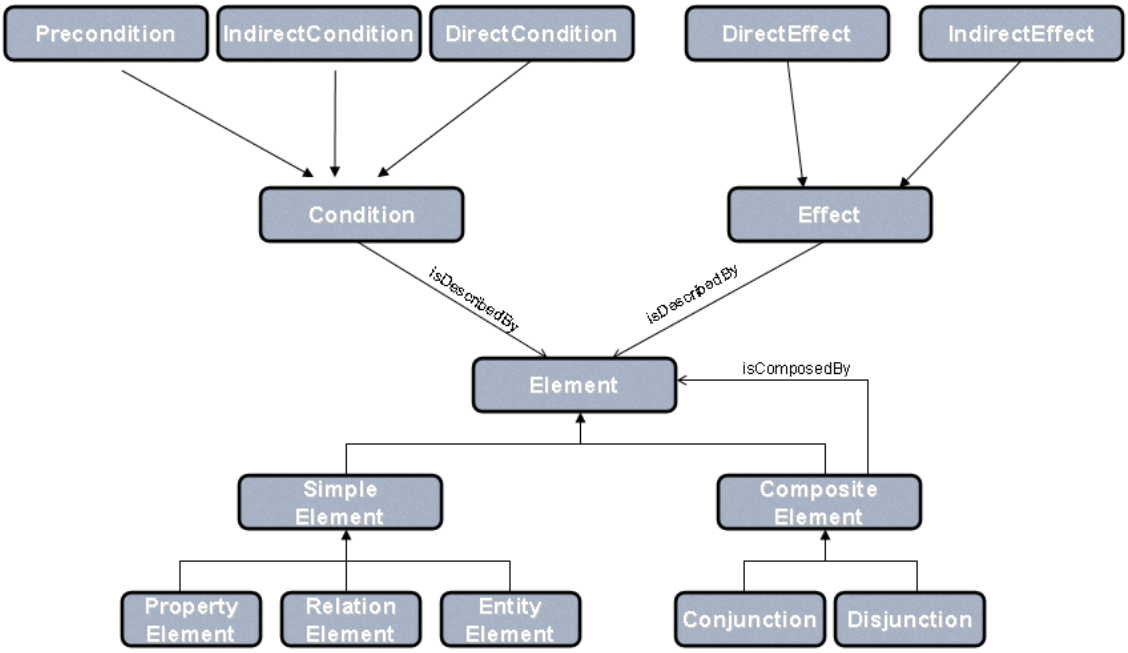


**Ilustración 41. Estructura de los tipos de Element y la relación de estos con Effect y Condition.**

# 3 BIBLIOGRAFÍA

[1] B. Angerer, "Space-Based Programming," O'Reilly Media, 2003.

[2] D. Fensel, "Triple-Space Computing: Semantic Web Services Based on Persistent Publication of Information," in *Intelligence in Communication Systems*, 2004, pp. 43-53.

[3] W. W. W. consortium, "Guía Breve de Servicios Web," 2008.

[4] J. M. Alonso, "Estándares del W3C en BPM," W3C Oficina Española, 2005.

[5] B. Sapkota, D. Foxvog, D. Wutke, D. Martin, M. Murth, O. Shafiq, A. Turati, E. Della Valle, N. Sanchez, and J. Kopecky, "D4.1 Architectural Integration of Triple Spaces with Web Service Infrastructures," TripCom2007.

[6] C. Bussler, "A Minimal Triple Space Computing Architecture," Digital Enterprise Research Institute2005.

[7] J. Riemer, "Implementation of CORSO Extension (Intranet, Internet, Multi-Port, Notification, Firewall, Security)," Vienna University of Technology, Vienna2007.

[8] R. Krummenacher, M. Fried, and D. Blunder, "tsc++ user guide," STI Innsbruck2005.

[9] TripCom, "State of the Art and Requirements Analysis," 2006.

[10] R. Krummenacher, E. Simperl, D. Foxvog, V. Momtchev, D. Cerizza, L. Nixon, D. Cerri, B. Sapkota, K. Teymourian, P. Obermeier, D. Martin, H. Moritsch, O. Shafiq, and D. de Francisco, "D6.5 Towards a Scalable Triple Space," TripCom2008.

[11] M. Murth, G. Joskowicz, K. Eva, D. Cerizza, Cerri,Davide, D. de Francisco, A. Ghioni, R. Krummenacher, D. Martin, L. Nixon, N. Sanchez, B. Sapkota, O. Shafiq, and D. Wutke, "D6.2 Triple Space reference architecture.," TripCom2008.

[12] L. J. Nixon, D. Martin, D. Wutke, M. Murth, Simperl,Elena, R. Krummenacher, B. Sapkota, Z. Zhou, H. Moritsch, C. Schreiber, O. Shafiq, G. Toro del Valle, D. Cerri, and V. Momtchev, "D6.3 Platform API Specification for Interaction Between All Components," TripCom2008.

[13] Tripcom, "Triplespace Computing: Large-Scale Integrated Knowledge Applications," 2008.

[14] J. Community, "JXTA Community Projects," 2008.

[15] P-Grid, "The P-Grid Project." vol. 2008.

[16] Q. H. Mamoud, "Getting Started With JavaSpaces Technology: Beyond Conventional Distributed Programming Paradigms," 2005.

[17] "The Blitz Project," 2008.

[18] "D1.1 State-of-the-Art and Requirements Analysis," TripCom2006.

[19] ORDI, "Ontology Representation and Data Integration," 2008.

[20] B. Sapkota, V. Momtchev, and O. Shafiq, "High-Performance Storage Implementation," TripCom2008.

[21] Ontotext, "SwiftOWLIM System Documentation," 2007.

[22] B. Fitzpatrick, "Memcached Official Site," 2008.

[23] A. Riddoch, N. Gibbins, and S. Harris, "Sitio web del proyecto 3store," 2008.

[24] C. Min and F. Martin, "RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network," in *Proceedings of the 13th international conference on World Wide Web* New York, NY, USA: ACM, 2004.

[25] O. Lab., "Fact Sheet," Sirma Group Corp.2007.

[26] J. I. Vazquez, "A Reactive Behavioural Model for Context-Aware Semantic Devices," 2007.

[27] O. Lassila, "Semantic Web, quo vadis?," Nokia Research Center, Cambridge, MA, 2006.

[28] D. Beckett, "SWAD-Europe Deliverable 3.11: Developer Workshop Report 4 - Workshop on Semantic Web Storage and Retrieval," 2004.

[29] F. Crivellaro and G. Genovese, "uJena : Gestione di ontologie sui dispositivi mobili," in *Facoltà di Ingegneria dell'Informazione*: Politecnico di Milano, 2007.

[30] "JXTA Java Micro Edition Project," 2008.

[31] T. Rybicki, "MobileSpaces - JavaSpaces for mobile devices," in *Faculty of Electronics and Information Technology* Warsaw University of Technology: Warsaw, 2004.

[32] "Gumstix Web Page," 2008.

[33] "Gumstix Wiki," 2008.

[34] "Sun SPOT World," 2008.

[35] S. Nik, N. S. Douglas, and R. B. William, "A java virtual machine architecture for very small devices," *SIGPLAN Not.,* vol. 38, pp. 34-41, 2003.

[36] "Millennial, Wireless Sensor networks," Noviembre 2008.

[37] "Crossbow, Wireless Sensor Network," 2008.

[38] "DustNetworks, Wireless Sensor Network," 2008.

[39] "Sensicast, Wireless Sensor Network," 2008.

[40] "SquidBee Project," 2008.

[41] J. C. Augusto, *Ambient Intelligence: The Confluence of Pervasive Computing and Artificial Intelligence*

*Intelligent Computing Everywhere*: Springer, 2007.

[42] J. C. Augusto and P. McCullagh, "Ambient Intelligence: Concepts and Applications," pp. 1-28.

[43] S. K. Das and D. J. Cook, "Designing and modeling smart environments," pp. 490-494.

[44] J. C. Augusto and C. D. Nugent, "Smart homes can be smarter," pp. 1-15.

[45] C. Ramos, J. Augusto, and a. D. Shapiro, "Ambient Intelligence - the next step for Artificial Intelligence (Guest Editors' Introduction to the Special Issue on Ambient Intelligence)," *IEEE Intelligent Systems,* vol. 23, pp. 15-18, 2008.

[46] H. Hagras, V. Callaghan, M. Colley, G. Clarke, A. Pounds-Cornish, and H. Duman, "Creating an Ambient-Intelligence Environment Using Embedded Agents," in *IEEE Intelligent Systems*. vol. 19, 2004, pp. 12-20.

[47] M. Galushka, D. Patterson, and N. Rooney, "Temporal data mining for smart homes," Springer-Verlag, 2006, pp. 85-108.

[48] D. Leake, A. Maguitman, and T. Reichherzer, "Cases, context, and comfort: opportunities for case-based reasoning in smart homes," Springer-Verlag, 2006, pp. 109-31.

[49] D. J. Cook, M. Huber, K. Gopalratnam, and M. Youngblood, "Learning to control a Smart home environment."

[50] E. O. Heierman and D. J. Cook, "Improving Home Automation by Discovering Regularly Occurring Device Usage Patterns," pp. 537-540.

[51] M. C. Mozer, R. H. Dodier, M. Anderson, L. Vidmar, R. F. Cruickshank, and D. Miller, "The neural network house: an overview," Erlbaum, 1995, pp. 371-380.

[52] F. Rivera-Illingworth, V. Callaghan, and H. Hagras, "A neural network agent based approach to activity detection in AmI environments," pp. 92-99.

[53] L. Liao, D. J. Patterson, D. Fox, and H. Kautz, "Behavior recognition in assisted cognition," pp. 41-42.

[54] H. Kautz, D. Fox, O. Etzioni, G. Borriello, and L. Arnstein, "An overview of the Assisted Cognition Project," pp. 60-65.

[55] D. Leake, A. Maguitman, and T. Reichherzer, "Cases, context, and comfort: opportunities for case-based reasoning in smart homes," Springer-Verlag, 2006, pp. 109-131.

[56] S. Marzano, "The home of the future: Smarter but simple."

[57] M. C. Mozer, "Lessons from an Adaptive Home," Wiley-Interscience, 2004, pp. 273-298.

[58] U. Rutishauser, J. Joller, and R. Douglas, "Control and Learning of Ambience by an intelligent building," in *IEEE on Systems, man and cybernetics: a special issue on ambient intelligence*, 2004, pp. 1-12.

[59] A. Kulkarni, "A reactive behavioral system for the intelligent room."

[60] M. Garofalakis, K. P. Brown, M. J. Franklin, J. M. Hellerstein, D. Z. Wang, E. Michelakis, L. Tancau, E. Wu, S. R. Jeffery, and R. Aipperspach, "Probabilistic Data Management for pervasive computing: The data furnace project," 2006, pp. 1-7.

[61] M. Chan, C. Hariton, P. Ringeard, and E. Campo, "Smart house automation system for the elderly and the disabled," pp. 1586-1589.

[62] P. Huuskonen, *Run to the Hills! Ubiquitous Computing Meltdown*

*Advances in Ambient Intelligence*: IOS Press, 2007.

[63] M. C. Mozer, "The neural network house: an environment that adapts to its inhabitants," pp. 110-114.

[64] A. Boisvert and R. B. Rubio, "Architecture for intelligent thermostats that learn from occupants' behavior," pp. 124-130.

[65] E. Campo, S. Bonhomme, M. Chan, and D. Esteve, "Learning life habits and practices: an issue to the smart home," pp. 355-358.

[66] M. M. Luaces, C. Gayoso, and S. Suarez, "Intelligent virtual environments: Operating conditioning and observational learning in agents using neural networks," pp. 127-134.

[67] J. Choi and D. Shin, "Research and implementation of the context-aware middleware for controlling home appliances," *IEEE Transactions on Consumer Electronics,* vol. 51, pp. 301-306, 2005.

[68] R. Begg and R. Hassan, "Artificial neural networks in smart homes," Springer-Verlag, 2006, pp. 146-164.

[69] C. L. Gal, J. Martin, A. Lux, and J. L. Crowley, "SmartOffice: Design of an Intelligent Environment," *IEEE Intelligent Systems,* vol. 16, pp. 60-66, 2001.

[70] O. Brdiczka, P. Reignier, and J. L. Crowley, "Supervised Learning of an Abstract Context Model for an Intelligent Environment," pp. 259-264.

[71] T. M. Mitchell, *Machine Learning*: The McGraw-Hill and MIT Press, 1997.

[72] V. R. Jakkula and D. J. Cook, "Using Temporal Relations in Smart Environment Data for Activity Prediction."

[73] C. Hamblin, "Instants and Intervals," pp. 325-331.

[74] J. Allen, "Towards a general theory of action and time," pp. 123-154.

[75] R. Agrawal and R. Srikant, "Mining Sequential Patterns," pp. 3-14.

[76] N. M. Sadeh, F. L. Gandom, O. B. Kwon, and I. Cmu, "Ambient Intelligence: The MyCampus Experience," ISRI2005.

[77] D. J. Cook and S. K. Das, "How smart are our environments? An updated look at the state of the art," pp. 53-73.

[78] S. P. Rao and D. J. Cook, "Predicting inhabitant action using action and task models with application to smart homes," *International Journal on Artificial Intelligence Tools (Architectures, Languages, Algorithms),* vol. 13, pp. 81-99, 2004.

[79] V. Duong, Q. Phung, H. Bui, and S. Venkatesh, "Human behavior recognition with generic exponential family duration modeling in the hidden semi-Markov model," pp. 202-207.

[80] F. Doctor, H. Hagras, and V. Callaghan, "A fuzzy Embedded Agent-Based Approach for realizing Ambient Intelligence in Intelligent Inhabited Environments," in *IEEE Transactions on systems, man and cybernetics*. vol. 35, 2005, pp. 55-65.

[81] H. Hagras, V. Callaghan, M. Colley, G. Clarke, A. Pounds-Cornish, and H. Duman, "Creating an Ambient-Intelligence Environment Using Embedded Agents," *IEEE Intelligent Systems,* vol. 19, pp. 12-20, 2004.

[82] University Essex, "iSpace," 2008.

[83] E. Aarts, "365 days Ambient Intelligence in Homelab," in *Philips Research*, 2003, pp. 1-36.

[84] P. L. Emiliani and C. Stephanidis, "Universal access to ambient intelligence environments: Opportunities and challenges for people with disabilities," in *IBM Systems Journal*. vol. 44, 2005, pp. 605-619.

[85] T. Martin, "Fuzzy ambient intelligence in home telecare," pp. 12-13.

[86] S. Luhr, G. West, and S. Venkatesh, "Recognition of emergent human behaviour in a smart home: A data mining approach," *Pervasive and Mobile Computing,* vol. 3, pp. 95-116, 2007.

[87] M. Chan, C. Hariton, P. Ringeard, and E. Campo, "Smart house automation system for the elderly and the disabled," pp. 1586-1589.

[88] G. Demiris, M. Skubic, M. Rantz, J. Keller, M. Aud, B. Hensel, and Z. He, "Smart home sensors for the elderly: a model for participatory formative evaluation," 2006.

[89] M. E. Pollack, "Intelligent technology for an aging population: The use of AI to assist elders with cognitive impairment," *AI Magazine,* vol. 26, pp. 9-24, 2005.

[90] B. Raman and R. H. Katz, "An architecture for highly available wide-area service composition," *Computer Communications,* vol. 26, pp. 1727-1740, 2003.

[91] U. Küster, M. Stern, and B. König-Ries, "A Classification of Issues and Approaches in Automatic Service Composition."

[92] A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, and T. Payne, "DAML-S: Web Service Description for the Semantic Web," *The Semantic Web-ISWC,* vol. 363, 2002.

[93] K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan, "Automated Discovery, Interaction and Composition of Semantic Web Services," *Journal of Web Semantics,* vol. 1, pp. 27-46, 2003.

[94] M. Vallee, F. Ramparany, and L. Vercouter, "Dynamic service composition in ambient intelligence environments: a multi-agent approach."

[95] M. Vallee, F. Ramparany, and L. Vercouter, "A Multi-Agent System for Dynamic Service Composition in Ambient Intelligence Environments," pp. 165-171.

[96] K. Fujii and T. Suda, "Dynamic service composition using semantic information," pp. 39-48.

[97] B. Srivastava and J. Koehler, "Planning with Workflows-An Emerging Paradigm for Web Service Composition," *Workshop on Planning and Scheduling for Web and Grid Services,* 2004.

[98] D. Chakraborty, "Service Discovery and Composition in Pervasive Environments," 2004.

[99] D. Chakraborty and A. Joshi, "Dynamic Service Composition: State-of-the-Art and Research Directions," *Technical Report TR-CS-01-19,* vol. 19, 2001.

[100] P. Wisner and D. N. Kalofonos, "A Framework for End-User Programming of Smart Homes Using Mobile Devices."

[101] J. Rao and X. Su, "A Survey of Automated Web Service Composition Methods," *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition,* 2004.

[102] J. Peer, "Web Service Composition as AI Planning-a Survey," University of St.Gallen2005.

[103] K. S. May Chan, J. Bishop, and L. Baresi, "Survey and Comparison of Planning Techniques for Web Services Composition," University of Pretoria2007.

[104] M. Ghallab, D. S. Nau, and P. Traverso, *Automated Planning: Theory and Practice*: Morgan Kaufmann, 2004.

[105] A. Alamri, M. Eid, and A. E. Saddik, "Classification of the state-of-the-art dynamic web services composition techniques," *International Journal of Web and Grid Services,* vol. 2, pp. 148-166, 2006.

[106] M. ter Beek, A. Bucchiarone, and S. Gnesi, "Web Service Composition Approaches: From Industrial Standards to Formal Methods," *Second International Conference on Internet and Web Applications and Services,* vol. 0, p. 15, 2007.

[107] A. Bucchiarone and S. Gnesi, "A Survey on Services Composition Languages and Models."

[108] M. H. ter Beek, A. Bucchiarone, and S. Gnesi, "A Survey on Service Composition Approaches: From Industrial Standards to Formal Methods," *ISTI-CNR Technical report,* 2006.

[109] S. R. Ponnekanti and A. Fox, "SWORD: A Developer Toolkit for Web Service Composition," *Proceedings of the Eleventh World Wide Web Conference, Web Engineering Track,* 2002.

[110] M. Sheshagiri, M. desJardins, and T. Finin, "A Planner for Composing Services Described in DAML-S," *Workshop on Planning for Web Services, International Conference on Automated Planning and Scheduling,* 2003.

[111] D. Wu, E. Sirin, J. Hendler, D. Nau, and B. Parsia, "Automatic Web Services Composition Using SHOP2," *Twelfth World Wide Web Conference,* 2003.

[112] M. Klein and A. Bernstein, "Toward High-Precision Service Retrieval," *IEEE Internet Computing,* vol. 8, pp. 30-36, 2004.

[113] A. Brogi, S. Corfini, and R. Popescu, "Composition-oriented Service Discovery."

[114] S. Bansal and J. M. Vidal, "Matchmaking of web services based on the DAML-S service model," pp. 926-927.

[115] R. Aggarwal, K. Verma, J. Miller, and W. Milnor, "Constraint Driven Web Service Composition in METEOR-S," pp. 23-30.

[116] S. Majithia, D. W. Walker, and W. A. Gray, "A Framework for Automated Service Composition in Service-Oriented Architectures," pp. 269-283.

[117] R. Masuoka, B. Parsia, and Y. Labrou, "Task Computing - The Semantic Web Meets Pervasive Computing," pp. 866-881.

[118] A. Ranganathan and S. McFaddin, "Using workflows to coordinate Web services in pervasive computing environments," *Proceedings of the IEEE International Conference on Web Services,* pp. 288-295, 2004.

[119] D. Berardi, "Automatic Composition Services: Models, Techniques and Tools,"

2005.

[120] S. B. Mokhtar, N. Georgantas, and V. Issarny, "COCOA: COnversation-based Service Composition in PervAsive Computing Environments with QoS Support," *Journal Of System and Software,* vol. 80, pp. 1941-1955, 2007.

[121] S. B. Mokhtar, N. Georgantas, and V. Issarny, "Cocoa: Conversation-based service composition in pervasive computing environments," *Proceedings of the IEEE International Conference on Pervasive Services,* 2006.

[122] D. Nau, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and Y. Fusun, "SHOP2: An HTN planning system," *Journal of Artificial Intelligence Research,* vol. 20, pp. 379--404, 2003.

[123] A. Qasem, J. Heflin, and H. Muñoz-Avila, "Efficient Source Discovery and Service Composition for Ubiquitous Computing Environments."

[124] E. Oren, G. Keith, and S. W. Daniel, "Sound and efficient closed-world reasoning for planning," *Artif. Intell.,* vol. 89, pp. 113-148, 1997.

[125] D. Wu, E. Sirin, B. Parsia, J. Hendler, and D. Nau, "Automatic Web Service composition using SHOP2," in *Workshop in Planning for Web Services - ICAPS*, Trento, Italy, 2003.

[126] S. McIlraith and T. Son, "Adapting Golog for Composition of Semantic Web Services," *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning,* pp. 482-493, 2002.

[127] H. J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. B. Scherl, "GOLOG: a logic programming language for dynamic domains," *Journal of Logic Programming,* vol. 31, pp. 59-83, 1997.

[128] M. Sheshagiri, M. desJardins, and T. Finin, "A Planner for Composing Services Described in DAML-S," in *Proceedings of the AAMAS Workshop on Web Services and Agent-based Engineering*, 2003.

[129] "Jess - the Rule Engine for Java Platform," 2006.

[130] "Rete Algorithm - http://herzberg.ca.sandia.gov/docs/52/rete.html," 2006.

[131] C. Ion, F. Boi, and B. Walter, "Type based service composition," in *Proceedings of the 13th international World Wide Web conference on Alternate track papers \& posters* New York, NY, USA: ACM, 2004.

[132] C. Peltz, "Web Service orchestration and choreography: a look at WSCI and BPEL4WS," *Web Services Journal,* 2003.

[133] C. Peltz, "Web Services Orchestration: a review of emerging technologies, tools and standards," *Technical report, HP Technical white paper,* 2003.

[134] G. B. Chafle, S. Chandra, V. Mann, and M. G. Nanda, "Decentralized orchestration of composite web services," pp. 134-143.

[135] W. Binder, I. Constantinescu, and B. Faltings, "Decentralized Orchestration of CompositeWeb Services," pp. 869-876.

[136] D. N. Kalofonos, F. D. Reynolds, and T. R. Nrc, "Task-Driven End-User Programming of Smart Spaces Using Mobile Devices," Nokia2006.

[137] K. Carey, D. Lewis, S. Higel, and V. Wade, "Adaptive Composite Service Plans for Ubiquitous Computing."

[138] T. Cottenier and T. Elrad, "Adaptive Embedded Services for Pervasive Computing."

[139] M. Vukovic and P. Robinson, "Adaptive, planning based, web service composition for context awareness," *Proceedings of the Second International Conference on Pervasive Computing,* 2004.

[140] A. Bottaro, J. Bourcier, C. Escoffier, and P. Lalanda, "Autonomic Context-Aware Service Composition," *2nd IEEE International Conference on Pervasive Services,* 2007.

[141] L. Qiu, Z. Shi, and F. Lin, "Context Optimization of AI planning for Services Composition," pp. 610-617.

[142] C. L. S. Ko, W. L. S. Lee, and A. Helal, "Context-Aware Service Composition for Mobile Network Environments."

[143] C. Hesselman, A. Tokmakoff, P. Pawar, and S. Iacob, "Discovery and Composition of Services for Context-Aware Systems," pp. 67-81.

[144] A. Mingkhwan, P. Fergus, O. Abuelma'Atti, M. Merabti, B. Askwith, and M. B. Hanneghan, "Dynamic service composition in home appliance networks," *Multimedia Tools and Applications,* vol. 29, pp. 257-284, 2006.

[145] G. Kaefer, R. Schmid, G. Prochart, and R. Weiss, "Framework for Dynamic Resource-Constrained Service Composition for Mobile Ad Hoc Networks," *8th Annual Conference on Ubiquitous Computing, Workshop on System Support for Ubiquitous Computing,* 2006.

[146] H. Pourreza and P. Graham, "On the Fly Service Composition for Local Interaction Environments," p. 393.

[147] S. Y. Lee, J. Y. Lee, and B. I. Lee, "Service Composition Techniques Using Data Mining for Ubiquitous Computing Environments," *International Journal of Computer Science and Network Security,* vol. 6, pp. 110-117, 2006.

[148] Z. Maamar, S. K. Mostefaoui, and H. Yahyaoui, "Toward an Agent-Based and Context-Oriented Approach for Web Services Composition," *IEEE Transactions on Knowledge and Data Engineering,* vol. 17, pp. 686-697, 2005.

[149] U. Bellur and N. C. Narendra, "Towards service orientation in pervasive computing systems," *International Conference on Information Technology: Coding and Computing,* vol. 2, pp. 289-295, 2005.

[150] S. K. Mostefaoui, A. Tafat-Bouzid, and B. Hirsbrunner, "Using Context Information for Service Discovery and Composition," *Proceedings of the Fifth International Conference on Information Integration and Web-based Applications and Services,* vol. 3, pp. 15-17, 2003.

[151] M. Sheshagiri, N. M. Sadeh, and F. Gandon, "Using Semantic Web Services for Context-Aware Mobile Applications," *MobiSys - Workshop on Context Awareness,* 2004.

[152] S. B. Mokhtar, D. Fournier, N. Georgantas, and V. Issarny, "Context-Aware Service Composition in Pervasive Computing Environments," pp. 129-144.

[153] D. Chakraborty, A. Joshi, T. Finin, and Y. Yesha, "Service Composition for Mobile Environments," *Journal on Mobile Networking and Applications, Special Issue on Mobile Services,* vol. 10, pp. 435-451, 2005.

[154] Z. Song, Y. Labrou, and R. Masuoka, "Dynamic Service Discovery and Management in Task Computing," *Mobiquitous,* vol. 00, pp. 310-318, 2004.

[155] Q. Ni, "Service Composition in Ontology enabled Service Oriented Architecture for Pervasive Computing."

[156] Q. Ni and M. Sloman, "An ontology-enabled Service Oriented Architecture for pervasive computing," pp. 797-8.

[157] A. Ranganathan and R. H. Campbell, "Autonomic pervasive computing based on planning," *Proceedings International Conference on Autonomic Computing,* pp. 80-87, 2004.

[158] M. Vallee, F. Ramparany, and L. Vercouter, "Flexible Composition of Smart Device Services," pp. 165-171.

[159] N. C. N. Sattanathan Subramanian and Z. Maamar, "ConWeSc: Context based semantic Web Services composition."

[160] W. L. S. L. C. Lee. S. Ko and A. Helal, "Context-Aware Service Composition for Mobile Network Environments."

[161] M. Weiser, "The Computer for the 21st century," in *Scientific American*. vol. 256, 1991, pp. 94-104.

[162] O. Lassila and M. Adler, "Semantic Gadgets: Ubiquitous Computing Meets the Semantic Web," pp. 363-376.

[163] E. Aarts, *The New Everyday. Views on Ambient Intelligence*: Uitgeverij 010 Publishers, 2003.

[164] E. C. I. S. T. A. Group, "Scenarios for Ambient Intelligence in 2010," *Technical Report, EU Commission,* 2001.

[165] J. I. Vazquez, D. López de Ipiña, and I. Sedano, "SoaM: A Web-powered Architecture for Designing and Deploying Pervasive Semantic Devices,"

*International Journal of Web Information Systems,* 2007.

[166] S. Grimm, "Discovery - Identifying Relevant Services," Springer, 2007, pp. 211-244.

[167] F. Zhu, M. W. Mutka, and L. M. Ni, "Service Discovery in Pervasive Computing Environments," *IEEE Pervasive Computing,* vol. 04, pp. 81-90, 2005.

[168] J. Nakazawa, H. Tokuda, W. K. Edwards, and U. Ramachandran, "A Bridging Framework for Universal Interoperability in Pervasive Systems," *icdcs,* vol. 00, p. 3, 2006.

[169] P. Grace, G. S. Blair, and S. Samuel, "ReMMoC: A Reflective Middleware to Support Mobile Client," pp. 1170-1187.

[170] T. Koponen and T. Virtanen, "A Service Discovery: A Service Broker Approach," *hicss,* vol. 09, p. 90284b, 2004.

[171] Y.-D. Bromberg and V. Issarny, "INDISS: Interoperable Discovery System for Networked Services," pp. 164-183.

[172] P. G. Raverdy, V. Issarny, R. Chibout, and A. de La Chapelle, "A Multi-Protocol Approach to Service Discovery and Access in Pervasive Environments."

[173] W. Zhao and H. Schulzrinne, "Enhancing service location protocol for efficiency, scalabiliy and advanced discovery," *J.Syst.Softw.,* vol. 75, pp. 193-204, 2005.

[174] M. Balazinska, H. Balakrishnan, and D. Karger, "INS/Twine: A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery."

[175] K. Arabshian and H. Schulzrinne, "GloServ: Global Service Discovery Architecture," *mobiquitous,* vol. 00, pp. 319-325, 2004.

[176] J. Liu and V. Issarny, "Signal Strength based Service Discovery (S3D) in Mobile Ad Hoc Networks."

[177] F. Zhu, M. Mutka, and L. Ni, "PrudentExposure: A Private and User-centric Service Discovery Protocol," *percom,* vol. 00, p. 329, 2004.

[178] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz, "An architecture for a secure service discovery service," pp. 24-35.

[179] A. Urbieta, E. Azketa, I. Gomez, J. Parra, and N. Arana, "Analysis of effects-and preconditions-based service representation in ubiquitous computing environments."

[180] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, and T. Payne, "OWL-S: Semantic Markup for Web Services," *W3C Member Submission,* vol. 22, 2004.

[181] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara, "Semantic Matching of Web Services Capabilities," pp. 333-347.

[182] D. L. McGuinness and F. V. Harmelen, "OWL Web Ontology Language Overview," 2004.

[183] M. K. Smith, C. Welty, and D. L. McGuinness, "OWL Web Ontology Language Guide," 2004.

[184] D. McDermott, "DRS: A Set of Conventions for Representing Logical Languages in RDF," 2004.

[185] M. R. Genesereth and R. E. Fikes, "Knowledge Interchange Format, Version 3.0 Reference Manual," 1994.

[186] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, "SWRL: A Semantic Web Rule Language Combining OWL and RuleML," *W3C Member Submission,* vol. 21, 2004.

[187] D. McDermott, "PDDL-the planning domain definition language," *The AIPS-98 Planning Competition Comitee,* 1998.

[188] M. Paolucci, A. Ankolekar, N. Srinivasan, and K. Sycara, "The DAML-S Virtual Machine," *2nd International Semantic Web Conference,* vol. 2870, pp. 290-305, 2002.

[189] D. Roman, H. Lausen, and U. Keller, "Web service modeling ontology standard (WSMO-standard)," *Working Draft D2v0,* vol. 2, 2004.

[190] D. Roman, "Web Service Modeling Ontology," *Applied Ontology,* vol. 1, pp. 77-106, 2005.

[191] C. Feier, D. Roman, A. Polleres, J. Domingue, M. Stollberg, and D. Fensel, "Towards Intelligent web Services: Web Service Modeling Ontology (WSMO)," *Proceedings of the International Conference on Intelligent Computing (ICIC),* pp. 23-26, 2005.

[192] D. Fensel and C. Bussler, "The Web Service Modeling Framework WSMF," *Electronic Commerce Research and Applications,* vol. 1, pp. 113-137, 2002.

[193] K. Verma, R. Akkiraju, and R. Goodwin, "Semantic Matching of Web Service Policies," *SDWP Workshop,* 2005.

[194] I. Constantinescu and B. Faltings, "Efficient matchmaking and directory services," pp. 75-81.

[195] J. de Bruijn, H. Lausen, R. Krummenacher, A. Polleres, L. Predoiu, M. Kifer, and D. Fensel, "The Web Service Modeling Language WSML," *WSML Final Draft,* vol. 16, 2005.

[196] M. Kifer and G. Lausen, "F-logic: a higher-order language for reasoning about objects, inheritance, and scheme," pp. 134-146.

[197] M. Kifer, G. Lausen, and J. Wu, "Logical foundations of object-oriented and frame-based languages," *Journal of the Association for Computing Machinery,*

vol. 42, pp. 741-843, 1995.

[198] O. M. Group, "Meta object facility (MOF) specification v1.4," 2004.

[199] D. Roman, L. Vasiliu, and C. Bussler, "Orchestration in WSMO," *WSMO working draft,* 2004.

[200] J. Scicluna, A. Polleres, and D. Roman, "Ontology-based Choreography and Orchestration of WSMO," *WSMO Working Draft,* 2005.

[201] S. Battle, A. Bernstein, H. Boley, B. Grosof, M. Gruninger, R. Hull, M. Kifer, D. Martin, S. McIlraith, and D. McGuinness, "Semantic Web Services Framework (SWSF)," *W3C Member Submission,* vol. 9, 2005.

[202] D. Berardi, H. Boley, B. Grosof, M. Gruninger, R. Hull, M. Kifer, D. Martin, S. McIlraith, J. Su, and S. Tabet, "Semantic Web Services Language (SWSL)," *Technical report, Semantic Web Services Initiative,* 2005.

[203] S. Battle, "Semantic Web Services Ontology (SWSO)," *Member submission, W3C, September,* 2005.

[204] D. Berardi, M. Gruninger, R. Hull, and S. McIlraith, "Towards a first-order ontology for semantic web services," *Working notes of the W3C Workshop on Constraints and Capabilities for Web Services,* 2004.

[205] M. Gruninger, "A Guide to the Ontology of the Process Specification Language," *Handbook on Ontologies.Sringer,* 2003.

[206] M. Gruninger and C. Menzel, "Process Specification Language: Principles and Applications," *AI Magazine,* vol. 24, pp. 63-74, 2003.

[207] C. Schlenoff, *The Process Specification Language (PSL) Overview and Version 1.0 Specification*: US Dept. of Commerce, Technology Administration, National Institute of Standards and Technology, 2000.

[208] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M. Schmidt, A. Sheth, and K. Verma, "Web Service Semantics-WSDL-S," *A joint UGA-IBM Technical Note, version,* vol. 1, 2005.

[209] J. Kopecký, D. Roman, M. Moran, and D. Fensel, "Semantic Web Services Grounding," p. 127.

[210] M.-S. Project, "METEOR-S Semantic Web Services and Processes." vol. 2006, 2006.

[211] A. Sheth, "Semantic Web Process Lifecycle: Role of Semantics in Annotation, Discovery, Composition and Orchestration," *Invited Talk, WWW 2003 Workshop on E-Services and the Semantic Web,* vol. 20, 2003.

[212] J. Cardoso, A. P. Sheth, J. A. Miller, J. Arnold, and K. Kochut, "Quality of service for workflows and web service processes," *J.Web Sem.,* vol. 1, pp. 281-308, 2004.

[213] A. A. Patil, S. A. Oundhakar, A. P. Sheth, and K. Verma, "Meteor-s web service annotation framework," pp. 553-562.

[214] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller, "METEOR-S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services," *Journal of Information Technology and Management, Special Issue on Universal Global Integration,* vol. 6, pp. 17-39, 2005.

[215] N. Oldham, K. Verma, A. P. Sheth, and F. Hakimpour, "Semantic WS-agreement partner selection," pp. 697-706.

[216] K. Sivashanmugam, "The METEOR-S Framework for Semantic Web Process Composition," *Master's Thesis,* 2003.

[217] K. Sivashanmugam, J. A. Miller, A. P. Sheth, and K. Verma, "Framework for Semantic Web Process Composition," *International Journal of Electronic Commerce,* vol. 9, pp. 71-106, 2004.

[218] H. Lausen and D. Innsbruck, "Semantic Annotations for WSDL (SAWSDL)," in *W3C Working Draft*. vol. 2006, 2006.

[219] R. Chinnici, M. Gudgin, J. J. Moreau, J. Schlimmer, and S. Weerawarana, "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language," *W3C Working Draft,* vol. 26, 2004.

[220] R. Akkiraju, B. Sapkota, and F. of, "Semantic annotations for WSDL and XML schema - Usage Guide (Working Group Note)." vol. 2008, 2007.

[221] I. S. T. A. Project and Ist, "Amigo D3.2 Amigo middleware core: Prototype implementation and documentation," 2006.

[222] U. Keller, H. Lausen, and M. Stollberg, "On the Semantics of Functional Descriptions of Web Services," p. 605.

[223] M. Klusch, "Semantic Web Service Coordination," Birkhäuser Verlag, Springer, 2008.

[224] A. M. Zaremski and J. M. Wing, "Specification matching of software components," pp. 6-17.

[225] K. Sycara, M. Klusch, S. Widoff, and J. Lu, "Dynamic service matchmaking among agents in open information environments," *ACM SIGMOD Record,* vol. 28, pp. 47-53, 1999.

[226] K. Sycara, J. Lu, M. Klusch, and S. Widoff, "Matchmaking among Heterogeneous Agents on the Internet."

[227] L. Capra, S. Zachariadis, and C. Mascolo, "Q-CAD: QoS and Context Aware Discovery Framework for Adaptive Mobile Systems," pp. 453-456.

[228] A. Bernstein and C. Kiefer, "Imprecise RDQL: towards generic retrieval in

ontologies using similarity joins," pp. 1684-1689.

[229] C. Kiefer and A. Bernstein, "The Creation and Evaluation of iSPARQL Strategies for Matchmaking," p. 463.

[230] D. Bianchini, V. D. Antonellis, M. Melchiori, and D. Salvi, "Semantic-Enriched Service Discovery."

[231] L. Li and I. Horrocks, "A Software Framework for Matchmaking Based on Semantic Web Technology," *Int.J.Electron.Commerce,* vol. 8, pp. 39-60, 2004.

[232] J. G. P. Filho and M. van Sinderen, "Web service architectures - semantics and context-awareness issues in web services platform," Telematica Instituut2003.

[233] D. Trastour, C. Bartolini, and J. Gonzalez-Castillo, "A Semantic Web Approach to Service Description for Matchmaking of Services."

[234] D. J. Mandell and S. A. McIlraith, "A Bottom-Up Approach to Automating Web Service Discovery, Customization, and Semantic Translation," *Twelfth International World Wide Web Conference Workshop on E-Services and the Semantic Web,* 2003.

[235] D. J. Mandell and S. A. McIlraith, "Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation," pp. 227-241.

[236] M. C. Jaeger, G. Rojec-Goldmann, C. Liebetruth, G. Muhl, and K. Geihs, "Ranked Matching for Service Descriptions using OWL-S," *Kommunikation in verteilten Systemen (KiVS 2005), Informatik Aktuell,* pp. 91-102, 2005.

[237] S. B. Mokhtar, D. Preuveneers, N. Georgantas, V. Issarny, and Y. Berbers, "EASY: Efficient SemAntic Service DiscoverY in Pervasive Computing Environments with QoS and Context Support," *Journal Of System and Software,* vol. 81, pp. 785-808, 2008.

[238] S. B. Mokhtar, "Semantic Middleware for Service-Oriented Pervasive Computing," 2007.

[239] M. Klusch and P. Kapahne, "Semantic Web Service Selection with SAWSDL-MX," pp. 3-17.

[240] M. Klusch, B. Fries, and K. Sycara, "Automated semantic web service discovery with OWLS-MX," pp. 915-922.

[241] M. Klusch and B. Fries, "Hybrid OWL-S Service Retrieval with OWLS-MX: Benefits and Pitfalls," pp. 47-61.

[242] F. Kaufer and M. Klusch, "Performance of Hybrid WSML Service Matching with WSMO-MX: Preliminary Results," pp. 63-77.

[243] F. Kaufer and M. Klusch, "WSMO-MX: a logic programming based hybrid service matchmaker."

[244] D. Kourtesis and I. Paraskakis, "Combining SAWSDL, OWL-DL and UDDI for Semantically Enhanced Web Service Discovery," pp. 614-628.

[245] U. Kuster, H. Lausen, and B. Konig-Ries, "Evaluation of Semantic Service Discovery-A Survey and Directions for Future Research," pp. 37-53.

[246] A. M. Zaremski and J. M. Wing, "Specification matching of software components," *ACM Transactions on Software Engineering and Methodology (TOSEM),* vol. 6, pp. 333-369, 1997.

[247] L. Lin and I. B. Arpinar, "Discovering Semantic Relations between Web Services Using Their Pre and Post-Conditions," pp. 237-238.

[248] L. Botelho, A. Fernandez, B. Fries, M. Klusch, L. Pereira, T. Santos, P. Pais, and M. Vasirani, "Service Discovery," Birkhäuser Verlag, Springer, 2008.

[249] E. Sirin, B. Parsia, and J. Hendler, "Template-based composition of semantic web services."

[250] M. Klein and B. Konig-Ries, "Coupled Signature and Specification Matching for Automatic Service Binding."

[251] R. C. Wang, Y. C. Chang, and R. S. Chang, "Design Issues of Semantic Service Discovery for Ubiquitous Computing," pp. 880-885.

[252] C. Lee and S. Helal, "Context Attributes: An Approach to Enable Context-awareness for Service Discovery," p. 22.

[253] G. Lee, P. Faratin, S. Bauer, and J. Wroclawski, "A User-Guided Cognitive Agent for Network Service Selection in Pervasive Computing Environments," p. 219.

[254] G. Chen and D. Kotz, "Solar: An open platform for context-aware mobile applications."

[255] G. Privat and T. Flury, "Position-Based Interaction for Indoor Ambient Intelligence Environments," pp. 192-207.

[256] F. Zhu, M. Mutka, and L. Ni, "Splendor: A Secure, Private, and Location-Aware Service Discovery Protocol Supporting Mobile Services," p. 235.

[257] P. Busetta, T. Kuflik, M. Merzi, and S. Rossi, "Service delivery in smart environments by implicit organizations," pp. 356-363.

[258] Y. Zhang, Y. Hou, Z. Huang, H. Li, and R. Chen, "A Context-Aware AmI System Based on MAS Model," pp. 703-706.

[259] P.-G. Raverdy, O. Riva, A. de La Chapelle, R. Chibout, and V. Issarny, "Efficient Context-aware Service Discovery in Multi-Protocol Pervasive Environments," p. 3.

[260] S. B. Mokhtar, P. G. Raverdy, R. Cardoso, A. Urbieta, and N. Georgantas,

"Discovering Social Services in Pervasive Environments with Privacy."

[261] S. B. Mokhtar, A. Kaul, N. Georgantas, and V. Issarny, "Efficient Semantic Service Discovery in Pervasive Computing Environments," pp. 240-259.

[262] S. B. Mokhtar, P. G. Raverdy, A. Urbieta, and R. Cardoso, "Interoperable Semantic {\&} Syntactic Service Matching for Ambient Computing Environment."

[263] S. B. Mokhtar, P. G. Raverdy, A. Urbieta, and R. Cardoso, "Interoperable Semantic {\&} Syntactic Service Matching for Ambient Computing Environments," *International Journal of Ambient Computing and Intelligence (IJACI),* 2009.

[264] B. Cuenca Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, and U. Sattler, "OWL 2: The Next Step for OWL," *Journal of Web Semantics,* 2008.

[265] "Pellet," 2008.

[266] "Racer Pro," 2008.

[267] "DIG interface," 2008.

[268] "Kaon," 2008.

[269] "Jena Framework," 2008.

[270] "Fact++ URL," 2008.

[271] "SweetRules," 2008.

[272] "Bossam," 2008.

[273] "Hoolet," 2008.

[274] T. Strang and C. Linnhoff-Popien, "A Context Modeling Survey."

[275] A. K. Dey, G. D. Abowd, and D. Salber, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications," *Human-Computer Interaction,* vol. 16, pp. 97-166, 2001.

[276] C. Preist, "A Conceptual Architecture for Semantic Web Services," pp. 395-409.