TESIS DOCTORAL

Coordinación de servicios Web: de las aplicaciones a los modelos formales.

Eloy Javier Mata Sotés

UNIVERSIDAD DE LA RIOJA

TESIS DOCTORAL

Coordinación de servicios Web: de las aplicaciones a los modelos formales.

Eloy Javier Mata Sotés

Universidad de La Rioja
Servicio de Publicaciones
2009

Esta tesis doctoral, dirigida por los doctores D. José Ángel Bañares Bañares y Julio Rubio García, fue leída el 13 de marzo de 2009, y obtuvo la calificación de Sobresaliente Cum Laude.

Edita: Universidad de La Rioja
Servicio de Publicaciones

ISBN 978-84-692-3913-1

UNIVERSIDAD DE LA RIOJA
Departamento de Matemáticas y Computación

Coordinación de servicios Web: de las aplicaciones a los modelos formales

Eloy Javier Mata Sotés

Memoria presentada para la
obtención del grado de Doctor

Directores:
Dr. D. José Ángel Bañares Bañares
Dr. D. Julio Rubio García

Logroño, Enero de 2009

Este trabajo ha sido parcialmente subvencionado por los proyectos TIC2002-01626 y TIN2006-13301 del Ministerio de Educación y Ciencia, el proyecto ACPI-2002/06 de la Comunidad Autónoma de La Rioja, los proyectos API-00/B28 y API-02/24 de la Universidad de La Rioja y las becas ATUR-02/22, ATUR-03/21 Y ATUR 04/32 de la Universidad de La Rioja.

A mi mujer, Marisa y
a mis hijos, Jorge y Silvia

Agradecimientos

En el instante de escribir estas líneas, uno echa la vista atrás y observa que son muchas las "grandes" personas a las que agradecer muchas "pequeñas" cosas. En cierto modo, la elaboración de una tesis se asemeja a una carrera de fondo. Aunque el objetivo final es llegar a la meta, lo realmente importante es que durante su desarrollo uno se conoce mejor a sí mismo y a los demás.

Debo comenzar por mis dos directores de tesis. José Angel Bañares me abrió las puertas para colaborar con el grupo de investigación IAAA de la Universidad de Zaragoza. Esto me permitió aprender sobre las áreas de los Sistemas de Información Geográfica y de los Servicios Basados en Localización, así como sobre las tecnologías asociadas a los sistemas distribuidos, especialmente los servicios Web. Sus aportaciones sobre estos temas siempre han sido muy certeras. Julio Rubio me dio la posibilidad de retomar mi carrera investigadora ofreciéndose para dirigir esta tesis. Durante este tiempo, él no ha sido sólo un director de tesis sino un hermano "mayor". Ha tenido más confianza en mí que yo mismo, y siempre ha estado ahí para solucionar cualquier problema. Su entusiasmo y su empuje han sido vitales en este trabajo.

Aparte de los directores de tesis, la persona que, sin duda, más ha contribuido en esta tesis ha sido Pedro Álvarez. Al él le debo el origen de este trabajo. Sin su desarrollo del Servicio de Coordinación, y sin su tesis doctoral, el tema de esta tesis no hubiera existido. Siempre ha estado disponible y dispuesto a explicarme todos aquellos aspectos relacionados con su trabajo previo.

Gracias también a Pedro Muro-Medrano y a Joaquín Ezpeleta, miembros de grupo de investigación IAAA, por permitirme colaborar en sus proyectos de investigación y proporcionarme los medios necesarios para mi investigación.

Mi agradecimiento a todos los compañeros del grupo de investigación al que pertenezco, Laureano Lambán, Angel Luis Rubio, Francisco García, Vico Pascual, Juanjo Olarte, César Domínguez, Eduardo Sáenz de Cabezón, Arturo Jaime, Mirian Andrés (a la que ya siempre le deberé una tarjeta postal), Ana Romero (gracias por las plantillas de LaTeX), Jesús María Aransay (gracias por su ayuda con el LaTeX); y a los últimos en incorporarse, Elsa García, Jonathan Heras y Beatriz Pérez. Con todos ellos he compartido charlas y

debates en el Seminario de Informática y más de un café.

También quiero agradecer al Departamento de Matemáticas y Computación de la Universidad de La Rioja por el buen ambiente que reina en el mismo, lo que facilita el trabajo tanto de docencia como de investigación. A todos aquellos que constantemente os habéis preocupado sobre cómo y cuándo iba a terminar la tesis (Judit, Ignacio, Luis, Víctor, ...), aquí tenéis la respuesta.

Por último, agradecer a toda mi familia, en particular a Marisa, Jorge y Silvia, su cariño, interés y su paciencia en los momentos de más duro trabajo.

Quiero terminar reiterando mis agradecimientos a Julio Rubio porque sin su ayuda este trabajo nunca hubiera visto su fin. Gracias Julio.

Índice General

Índice de figuras

Introducción

Esta memoria está dedicada a informar con detalle de una experiencia que relaciona la *práctica* del desarrollo de software con la *teoría* de dicho desarrollo. Así, este trabajo queda enmarcado dentro de la aplicación de los llamados *métodos formales* [HB95]. Es necesario precisar que dicha denominación agrupa dos tipos esencialmente distintos de investigación. Por una parte, tenemos a los investigadores dedicados al desarrollo de métodos formales, que, usualmente, también se ocupan de estudiar las propiedades de sus formalismos. Este tipo de investigación suele dar lugar a artículos y publicaciones que siguen los esquemas habituales dentro de las matemáticas llamadas *puras*. Por otra parte, hay una investigación relativa a los métodos formales que se dedica a *aplicar* (o, en muchas ocasiones, a *intentar aplicar*) los formalismos a casos reales de desarrollo industrial. El mayor peso del primer tipo de investigación (que podríamos denominar "métodos formales *teóricos*") frente al segundo ("métodos formales *aplicados*"), creemos que puede ser una de las causas de las conocidas polémicas en torno a los métodos formales o, más concretamente, en torno a su utilidad (véase a título de ejemplo [Gla04]). Respecto a esta clasificación, nuestra investigación se encuadra decididamente en el ámbito de los métodos formales *aplicados*, pues, pese a las dificultades, confiamos en que ciertas técnicas matemáticas puedan ser de ayuda para mejorar la comprensión y la fiabilidad de los desarrollos de software *reales*.

Respecto a las matemáticas utilizadas en nuestra investigación, conviene realizar una aclaración desde el principio. Habitualmente, los investigadores que estudian los métodos formales en sí (es decir, los dedicados a los métodos formales *teóricos*) utilizan un aparato matemático sofisticado y pueden llegar a demostrar teoremas de cierta complejidad (esto es así, aparte de por las necesidades técnicas propias requeridas por cada formalismo, porque dichos investigadores buscan el mismo concepto, elusivo, de *elegancia* que guía a muchas investigaciones en matemáticas puras). Por contra, suele darse el caso de que, cuando los métodos formales son aplicados a situaciones reales, el tipo de matemáticas necesarias es mucho más simple. Por ello, en esta memoria no se encontrarán complicadas estructuras matemáticas ni demostraciones ingeniosas. Solo el formalismo *imprescindible* ha sido utilizado. La justificación de este hecho es que el criterio para valorar las matemáticas que aparecen en los métodos formales aplicados no es su dificultad o su sofisticación sino su *adecuación* a los casos prácticos de estudio.

Los métodos formales pueden ser utilizados de muy diferentes modos cuando se enfrentan a situaciones reales. En ocasiones pueden aparecer en las etapas *tempranas*

del desarrollo del software (por ejemplo, en la parte de formalización de requisitos o en la especificación formal de la funcionalidad del sistema a construir). En otras, los métodos formales se aplican a artefactos software ya construidos, con el objetivo de estudiar sus propiedades (o la falta de alguna de ellas). Éste es nuestro caso. La idea desarrollada en la memoria es que, partiendo de un sistema real, en ocasiones es útil obtener un modelo formal sobre el que estudiar sus propiedades. Esta propuesta, lejos de ser original, está en la base de uno de los métodos formales que más éxito tiene en la actualidad, conocido en inglés como *model checking* [GV08]. Las similitudes con el *model checking* no van más allá: tanto las técnicas como los objetivos que perseguimos son diferentes en nuestro caso. Aparte de analizar formalmente un sistema software, desarrollado por otros investigadores y que está actualmente en uso, nuestra contribución es, creemos, bastante original: utilizamos el modelo formal para comprender más a fondo el sistema, de modo que podemos proponer extensiones y mejoras que, tras un diseño abstracto en el modelo formal, pueden dar lugar a un proceso de reingeniería que vuelva a tener aplicación en la realidad. Esta idea se plasmó, incluso, en el título de uno de los artículos que preceden a esta memoria: "*From Practice to Theory and going back again*".

Estos principios generales han sido instanciados en esta memoria para un campo concreto de aplicación: la coordinación de Servicios Web. Es éste, el de los Servicios Web, un campo en plena ebullición, donde junto a continuos intentos de estandarización y homogeneización, conviven grandes ambigüedades, incluso desde el punto de vista terminológico. Aunque la estandarización ha tenido un (relativo) éxito en lo que respecta a cada servicio Web *aislado*, la situación es más confusa en lo que respecta a la *composición* de servicios Web, es decir, a aquellas aplicaciones que se basan en la interacción de varios servicios Web para ofrecer una funcionalidad con valor añadido. Más allá del simple *encadenamiento* (*chaining*) de servicios Web, qué mecanismos pueden ser convenientes para describir cómo hacer trabajar conjuntamente varios servicios Web continúa siendo un campo de intensa investigación. Tras una primera etapa en que se prestó atención a una noción difusa de *coordinación* de servicios Web (que se intentó fijar en el estándar WS-Coordination [OAS06]), el interés parece dirigirse en la actualidad a los conceptos de *orquestación* y *coreografía* de servicios Web. El primero se refiere a la organización rígida de un plan de ejecución de servicios Web, dirigidos por un proceso central (parece que el estándar BPEL (Business Process Execution Language) [OAS07], que a su vez se basa en WS-Coordination, está siendo el más utilizado para la orquestación de servicios Web). El segundo concepto, la coreografía de servicios Web, se refiere a una descripción más abstracta de cómo cada servicio Web se ofrece al exterior, pudiendo ser utilizado para describir procesos de composición de servicios Web o incluso de orquestaciones, sin suponer la existencia de ningún nodo central que organice el proceso (aquí, el proceso de difusión del estándar WS-Choreography [BK04] parece más lento). Más allá de las distintas iniciativas de estandarización, lo que nos interesa destacar aquí es que, pese a ellas, el campo de investigación y aplicación de la composición de servicios Web dista mucho de estar cerrado. Como botón de muestra, nótese que pese a la existencia de un buen artículo de referencia [Pel03], la distancia que existe entre la definición que hace el organismo W3C de orquestación ("el patrón de interacciones que un agente servicio Web debe seguir para conseguir su objetivo"[W3C04]) y la descripción de BPEL [OAS07] es

inmensa. Sirva esto para explicar que, pese a los enormes beneficios de la estandarización, la situación en el ámbito de la composición de servicios Web hace que la adopción de formatos cerrados de descripción pueda suponer más un lastre que una ventaja. Esto se pone claramente de manifiesto en el debate de dos tipos de estándares relacionados con los servicios Web: aquellos basados en SOAP, con diseños altamente acoplados muy cercanos a los inspirados en invocación a procedimientos/métodos remotos, y los basados en REST, con diseños muy desacoplados similares los enlaces de navegación de los documentos Web [zMNS05]. A pesar de que la estandarización es crucial para poder soportar la interoperabilidad en arquitecturas orientadas a servicios, uno de los puntos que se apuntan como deficiencia de las aproximaciones basadas en estándares para publicación/búsqueda/invocación (*publish/find/bind*) de servicios es que de ninguna manera esto es suficiente, especialmente en los niveles más altos de abstracción. Apuntando a continuación la necesidad en la adopción de nuevos marcos y sistemas para el modelado conceptual, análisis, gestión, simulación, y comprobación de las abstracciones y modelos de servicios [MBCT06]. Una vez definidas la infraestructura de comunicación e integración, las aplicaciones pueden interaccionar en Internet, pero no hay mecanismos para decir cuándo y cómo [AFG+02]. Así, siendo coherentes con la intención *práctica* de nuestra investigación, nuestras propuestas van dirigidas al análisis formal de un modelo de coordinación de Servicios Web (basado a su vez en el modelo Linda de interacción [GC92]), que ofrece una infraestructura no estandarizada de composición, pero que es muy flexible y que, sobre todo, ha mostrado su utilidad en proyectos industriales *reales*.

Para terminar esta introducción, pasamos a continuación a describir cómo está organizado el resto de esta memoria. En el capítulo 1, de preliminares, presentamos someramente los servicios Web, los problemas planteados para su coordinación, así como el formalismo de base de nuestro estudio: el modelo Linda o de comunicación generativa [GC92]. El modelo Linda ha inspirado varias implementaciones en distintos lenguajes de programación. Nosotros presentamos en este primer capítulo, brevemente, *JavaSpaces* [FHA99], la herramienta Java que fue utilizada en el sistema software que pretendemos estudiar.

El capítulo 2, presenta dicho sistema software [Álv03, ÁBM+03c]. Aunque ese producto ha sido desarrollado por otros investigadores, la descripción que aquí realizamos tiene un cierto valor añadido: hemos realizado una tarea de análisis y abstracción que ha permitido destilar los puntos importantes, en los que un tratamiento formal sería conveniente. Así, se detectó que una cuestión clave era si el modo de interacción ofrecido por el sistema realmente respetaba la semántica del modelo Linda. Puesto que la implementación se basa en una serie de capas que reposan finalmente sobre *JavaSpaces*, la cuestión, en principio, distaba mucho de ser trivial. Por otra parte, puesto que las prestaciones del Servicio Web de Coordinación [Álv03] se apoyaban fuertemente en las de la comunicación generativa, podría entenderse que si la implementación no respetaba el modelo Linda, sería esencialmente incorrecta. Tras describir el diseño e implementación del sistema a formalizar, este segundo capítulo termina con una breve presentación del ámbito industrial en que se ha aplicado el software de coordinación de servicios Web (a saber, los sistemas basados en la localización geográfica [ÁBM+03c]). Esto nos permite

delimitar tanto la temática de aplicación como las características técnicas del modelo formal a desarrollar (en particular, en lo que concierne a la *generalidad* con la que debe plantearse la comunicación entre procesos).

En el capítulo 3 se define un modelo formal para el Sistema de Coordinación que ha sido descrito en el capítulo anterior. En primer lugar, particularizamos el modelo Linda, obteniendo un nuevo modelo que refleja aquellas características de *JavaSpaces* que serán relevantes para nuestro estudio. A continuación, emulando el modo en que el Sistema de Coordinación fue construido sobre *JavaSpaces*, introducimos un nuevo modelo de comunicación, basado en tuplas estructuradas (que representan los documentos XML no anidados utilizados en las aplicaciones industriales del Sistema de Coordinación). Demostramos, a continuación, nuestro teorema principal: el modelo estructurado respeta la semántica de Linda. Pese a que dicho resultado no pueda ser considerado una prueba formal de la corrección de Sistema de Coordinación (ya que el desarrollo formal ha sido llevado a cabo sobre una *abstracción* del sistema), sí que podemos afirmar que aumenta su fiabilidad. Nuestro teorema señala que las bases conceptuales y algorítmicas sobre las que se sustenta el Servicio de Coordinación son consistentes.

Más allá del valor del teorema principal, la aportación esencial de nuestro estudio formal reside en la mejor comprensión de los modelos conceptuales con los que se trabaja. En particular, una observación que aparece con claridad en nuestro modelo formal (pero que, hasta donde nosotros sabemos, no había sido reflejada en la literatura hasta la fecha) es que el procedimiento de emparejamiento (*matching*) propugnado por la comunicación generativa puede ser *complejo*. Es necesario recordar aquí que en el modelo Linda inicial, una serie de procesos se comunican a través de un espacio de tuplas compartido, en el que cada proceso puede dejar o tomar una tupla. Para leer una tupla del espacio, un proceso lanza una consulta en forma de tupla, posiblemente con máscaras o comodines (*wildcards*). Dichas tuplas se emparejan por medio de una igualdad *literal* entre las constantes que componen cada tupla. Puesto que nuestro modelo (formal) estructurado requiere un emparejamiento "en dos pasos", resulta natural preguntarse por qué no admitir otros tipos de emparejamiento más elaborados. Esta idea es aplicada en dos contextos diferentes en el siguiente capítulo.

La primera parte del capítulo 4 está dedicada a una aplicación del Servicio de Coordinación en la que los procesos interactúan a través de una emparejamiento semántico: en lugar de una igualdad literal entre cadenas de caracteres, se busca que las cadenas sean cercanas desde el punto de vista de su significado conceptual. Esta aproximación es útil en aquellos contextos en los que los términos provienen de fuentes heterogéneas (como es frecuente en los sistemas basados en la localización, que dependen de bases de información geográfica [BBG+01]) o en aquellos con necesidades de multilingüismo. Esta parte de la memoria tiene un estilo muy diferente del resto, ya que para abordar el problema del emparejamiento semántico tuvimos que utilizar métodos pragmáticos y heurísticos, pues en este punto nuestra investigación se acerca a problemas clásicos en Inteligencia Artificial y en Lingüística Computacional. Aquí, por tanto, como es usual en la investigación *aplicada*, el modelo formal se mantiene prácticamente sin cambios, y el peso de la investigación recae en la ingeniería.

La segunda propuesta de este cuarto capítulo es de naturaleza *teórica*, totalmente diferente por tanto a la anterior. Aquí se generaliza el emparejamiento admitiendo consultas que puedan ser expresadas en un pequeño lenguaje de restricciones. Este enfoque explicita la relación del modelo Linda con las Bases de Datos o los Sistemas Basados en Reglas. Precisamente el paralelismo con los Sistemas Basados en Reglas es la fuente de inspiración para las aportaciones que se recogen en el capítulo 5.

Una vez que en el capítulo 4, y más concretamente en su segunda parte, se ha dado el paso para permitir consultas basadas en restricciones, una generalización evidente (*evidente* una vez que un modelo formal suficientemente claro ha sido construido) consiste en extender el modelo para que las restricciones puedan plantearse inter-tuplas (en lugar de intra-tuplas como propusimos en el capítulo 4). Lejos de ser una generalización "teórica" más, esta idea incide en un ámbito de enorme importancia en la actualidad: el tratamiento de transacciones en Servicios Web [Pri08, LZH08]. Más concretamente, la mencionada generalización nos permite proponer un *modelo Linda transaccional*. En lugar de asociar el concepto de transaccionalidad a una consulta simple, e imitar el modo de trabajo en Bases de Datos (como se hace, por ejemplo, en [AS92]), nuestra propuesta consiste en extender el lenguaje de consultas. Hay que destacar que, a diferencia de lo hecho en los capítulos anteriores, esto implica un *cambio de modelo*, y ya no podemos apoyarnos en la semántica estándar de Linda para verificar la validez de nuestra propuesta. En esta ocasión se trata de generar un nuevo modelo que sea una extensión de Linda, con características transaccionales.

Así, debemos proponer, siguiendo la estela de Linda, una implementación estándar de nuestros operadores básicos. Dicha implementación debe ser tan clara como sea posible (pues va a actuar como *axiomática* del nuevo modelo), aunque sea a costa de la eficiencia. Para que sea útil, nos ha parecido conveniente realizar un diseño abstracto de una implementación más eficaz de nuestro modelo. Siguiendo el paralelismo de Linda con los sistemas basados en reglas (más que con las bases de datos), proponemos un algoritmo eficaz de lectura simultánea de varias tuplas, basado en el conocido procedimiento RETE [For82] que ha sido implementado inicialmente en OPS5, y entre otros, en el extendido sistema CLIPS [GR94]. Una vez realizada esta propuesta, encontramos la dificultad de saber si este algoritmo eficiente (pero de una relativa sofisticación) es *correcto*, es decir, si es equivalente al dado para *definir* nuestro modelo. Analizamos de un modo semi-formal esta cuestión, dejando para el siguiente y último apartado, de conclusiones y trabajo futuro, explorar técnicas (como el *testing automatizado*) que han sido propuestas en otras áreas de aplicación de los métodos formales [ALR07].

Tras las conclusiones, en el trabajo futuro indicamos, entre otros temas más técnicos, la necesidad de poner en práctica las extensiones teóricas del capítulo 5. Solo así podremos confirmar experimentalmente nuestra propuesta: que ir de la práctica (desarrollo de software) a la teoría (métodos formales) puede tener importantes impactos en las aplicaciones reales.

Capítulo 1

Preliminares

1.1. Internet como *middleware* para sistemas distribuidos

Las aplicaciones distribuidas constan de una colección de entidades software (componentes, aplicaciones, ...) que se ejecutan de forma concurrente sobre plataformas posiblemente heterogéneas conectadas a una misma red. Para facilitar y gestionar la interacción entre las entidades software que se ejecutan en plataformas heterogéneas se suele utilizar un *middleware*.

El *middleware* es un software de sistemas que reside entre la capa de aplicaciones y las capas inferiores (sistemas operativos subyacentes y capa de red). El objetivo del *middleware* es ofrecer un conjunto de servicios que facilite la comunicación entre las entidades software que componen una aplicación distribuida. Ofrece abstracciones de programación que liberan al programador de algunas de las dificultades de las comunicaciones en red, control de la concurrencia, gestión de transacciones, etc. Algunos ejemplos de *middleware* son CORBA, COM, RMI o EJB.

En definitiva, el *middleware* permite la interoperabilidad entre entidades software, es decir, les ofrece la capacidad de cooperar entre sí, a pesar de su heterogeneidad en cuanto a lenguajes de programación, sistemas operativos subyacentes o plataformas de ejecución.

Por otra parte, Internet, la red de redes, permite interconectar ordenadores situados en casi cualquier parte del mundo. Se basa en protocolos estándares como *Transmission Control Protocol* (TCP), encargado del envío y recepción fiable de datos en paquetes, o *Internet Protocol* (IP) que se encarga del direccionamiento de los paquetes a través de la red. La combinación de los protocolos TCP e IP es la base tecnológica sobre la que se sustenta Internet porque permite el envío de paquetes de datos a través de múltiples redes y el uso de otros muchos estándares, sobre TCP/IP, que han facilitado el intercambio de información en Internet. Algunos de estos protocolos son:

- El protocolo de sesión remota: *Telnet*.
- El protocolo de correo electrónico: *Simple Mail Transfer Protocol* (SMTP), extendido más tarde con *Multi-purpose Internet Mail Extensions* (MIME), que soporta el intercambio de ficheros no sólo de texto sino de otros tipos como audio, vídeo o imagen.
- El protocolo para la transferencia de ficheros: *File Transfer Protocol* (FTP), que permite a un sistema publicar un conjunto de ficheros alojándolos en un servidor FTP y a los usuarios acceder al servidor bien de forma registrada o incluso de forma anónima. Este protocolo permitió el desarrollo de los primeros sistemas de información distribuidos, antecesores de la Web, como *Archie* o *Gopher*.

La *World Wide Web* (popularmente conocida como Web), que surgió inicialmente sólo como otra tecnología para compartir información en Internet, pronto se convirtió en un medio para conectar usuarios remotos con aplicaciones, o como un medio para integrar aplicaciones a través de Internet. Principalmente, la Web se basa en el protocolo HTTP y el lenguaje de marcado HTML:

- *HyperText Transfer Protocol* (HTTP), es un protocolo genérico que gobierna la transferencia de ficheros en una red. Es genérico porque soporta el acceso a otros protocolos como FTP o SMTP. La información se intercambia con HTTP en forma de documentos, los cuales se identifican mediante Identificadores de Recursos Uniformes (URIs). El documento intercambiado puede ser estático, cuando es el propio recurso el que se envía, o dinámico, si su contenido es generado en el momento de acceso. El funcionamiento de HTTP está basado en el modelo cliente/servidor, generalmente usando *sockets* TCP/IP. Un cliente HTTP (e.g., un navegador) abre una conexión en un servidor HTTP (un servidor Web) y envía un mensaje de petición de recurso. El servidor devuelve un mensaje de respuesta, que generalmente contiene el documento solicitado, y cierra la conexión. El diálogo HTTP está basado en un conjunto reducido de operaciones: GET, POST, PUT y DELETE. HTTP es un protocolo sin estado, es decir, que no guarda ninguna información sobre conexiones anteriores, incluso aunque haya sido enviada desde el mismo cliente al mismo servidor, lo que hace necesario el uso de mecanismos de recuerdo como los *cookies*.
- *HyperText Markup Language* (HTML) es un lenguaje de marcado utilizado para la creación de páginas Web. Proporciona un medio para describir la estructura de un documento de texto (definiendo párrafos, títulos, listas, etc.) y presentarlo en forma de hipertexto mediante el uso de enlaces. Además permite enriquecer el texto con la inclusión de imágenes y otros objetos multimedia o la definición de formularios interactivos y código escrito en otros lenguajes.
- Actualmente, para dotar a la Web de mayor interactividad con el usuario y para la generación dinámica de contenido se utilizan herramientas muy diversas, como por ejemplo *JavaScript*, hojas de estilo (CCS), DHTML, o *applets* (en el lado del cliente) y CGI, ASP, PHP, *servlets* o JSP (en el lado del servidor).

1.2. Servicios Web

La tendencia de usar Internet como un *middleware* estándar, capaz de resolver distintos problemas de heterogeneidad e interoperabilidad ha inducido una nueva forma de desarrollo de aplicaciones distribuidas basada en la integración de servicios, más conocidos como *servicios Web*.

Los servicios Web son entidades software distribuidas cuya funcionalidad es accesible a través de una interfaz genérica y homogénea. Esta interfaz oculta los detalles de implementación del servicio, permitiendo que sea utilizado por otras entidades o servicios independientemente de la plataforma hardware o software sobre la que se estén ejecutando o de los lenguajes de programación utilizados [ACKM04].

Actualmente hay cuatro estándares básicos que son los más utilizados para desarrollar servicios Web. XML como formato para los datos que se vayan a intercambiar, SOAP para la transferencia de datos, WSDL para describir servicios y UDDI para publicar qué servicios están disponibles.

- *Simple Object Access Protocol* (SOAP) es un protocolo para la definición de mensajes e invocación de servicios. Los mensajes SOAP intercambiados son independientes del protocolo de transporte utilizado, del lenguaje de programación, del modelo de objetos, de la plataforma de ejecución y de las reglas para la representación de los datos en los mensajes (`http://www.w3.org/TR/soap`).

- *Web Services Description Language* (WSDL) es un lenguaje para la descripción, en formato XML, de la interfaz de un servicio. Permite definir la funcionalidad que ofrece el servicio, la localización del mismo y cómo puede ser invocado. Esta descripción especifica las características operacionales del servicio, pero no dice nada sobre cómo debe ser implementado (`http://www.w3.org/TR/wsdl`).

- *Universal Description, Discovery and Integration* (UDDI) establece una forma estándar, basada en XML, para el registro y búsqueda de servicios. La especificación de UDDI define una serie de APIs, basadas en SOAP, para la publicación y descubrimiento de servicios en repositorios (`http://uddi.xml.org/uddi-org`).

Muchas aplicaciones, basadas en la interacción de servicios Web, lo hacen a través de estos protocolos. No obstante, estos estándares no constituyen lo esencial de la tecnología de los servicios Web y otros protocolos pueden ser utilizados en su lugar.

Desde la aparición de los servicios Web, se han utilizado con frecuencia como patrón para la construcción de aplicaciones distribuidas las conocidas como Arquitecturas Orientadas a Servicios (SOAs), que describen cómo se organizan los servicios y cómo se comunican entre sí [GSB+02]. Aunque los servicios Web no son el origen de las SOAs, actualmente es la tecnología más empleada como implementación de SOAs y han contribuido a su difusión y adopción.

Siguiendo estas arquitecturas se pretende que los componentes software desarrollados en las aplicaciones sean más reusables, puesto que la interfaz de los servicios siguen unos estándares (e.g., WSDL). Se basan en una definición formal de la interfaz que encapsula las particularidades de la implementación, lo que la hace independiente de la plataforma subyacente, del lenguaje de programación o de la tecnología de desarrollo.

Al contrario de las arquitecturas orientadas a objetos, las SOAs están formadas por servicios de aplicación débilmente acoplados y altamente interoperables. Proporcionan un modelo de programación estándar que permite a los servicios residir en cualquier red, ser publicados, descubiertos e invocados por otros servicios. Para interactuar entre sí, los servicios se intercambian mensajes, en vez de operaciones, e interpretan alguno de estos tres roles: proveedor de servicios, cliente de servicios o registro de servicios (figura 1.1).

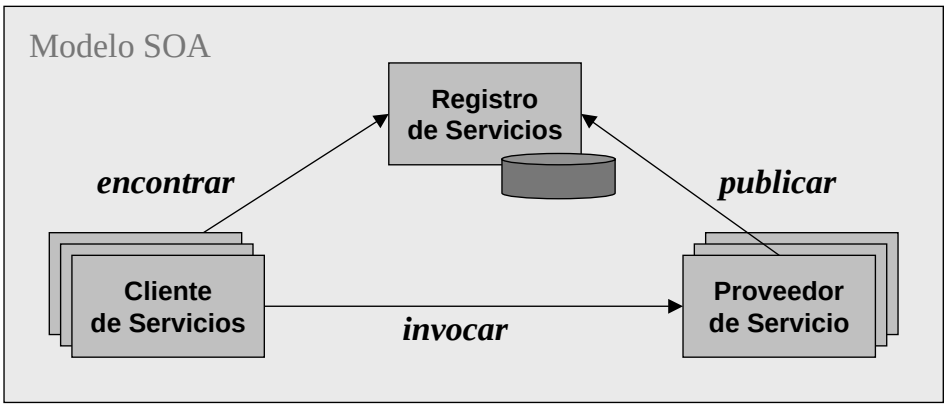

Figura 1.1: Arquitectura Orientada a Servicios

- El **proveedor de servicios** es responsable de crear servicios y de definir las interfaces para su invocación. Los servicios puede ser nuevas aplicaciones o envoltorios de sistema heredados (*wrapped legacy systems*) que los hacen accesibles a través de la red. Además tiene que publicar una descripción de cada uno de sus servicios en algún registro de servicios. La información incluida en la descripción del servicio será usada por el registro de servicios para catalogar cada servicio y encontrarlo cuando reciba peticiones de búsqueda de servicios.

- El **cliente de servicios** localiza entradas en el registro de servicios que se ajusten a sus necesidades utilizando diversas operaciones de búsqueda. Una vez encontrada la descripción de un servicio, debe ser capaz de interpretarla para invocar correctamente las operaciones del proveedor del servicio.

- El **registro de servicios** es el responsable de hacer accesible a los clientes de servicios las descripciones de servicios publicadas previamente por sus correspondientes proveedores. El registro proporciona a los clientes diversas operaciones de búsqueda que facilitan la localización de servicios.

Habitualmente, cada servicio Web desempeña uno de estos tres roles e interacciona con otros servicios, según su rol, con una de estas tres operaciones: *publicar un servicio*, *encontrar un servicio* e *invocar un servicio*. Sin embargo, no existe ninguna restricción para que un mismo servicio no pueda desempeñar simultáneamente más de un rol, por ejemplo, encontrar e invocar a otro servicio cuando resuelve una petición a su propio servicio.

1.3. Un ejemplo de servicio Web

El siguiente ejemplo[1] ilustra el funcionamiento de los servicios Web.

Una agencia de viajes ofrece los servicios de contratación de un vuelo de avión y de una habitación de hotel en la ciudad de destino mediante un servicio Web. El usuario del servicio Web solicita información para viajar en avión a alguna ciudad. El servicio consistirá en la reserva de un vuelo y de una habitación de hotel en la ciudad de destino. Para proporcionar al cliente la información que solicita cuando realiza la consulta, el servicio Web de la agencia de viajes solicita, a su vez, información a otros servicios Web. La agencia de viajes será cliente de estos otros servicios Web, en concreto un servicio Web proporcionará información sobre vuelos y otro sobre hoteles. Además, el usuario realizará el pago con una tarjeta de crédito a través de otro servicio Web seguro (figura 1.2).

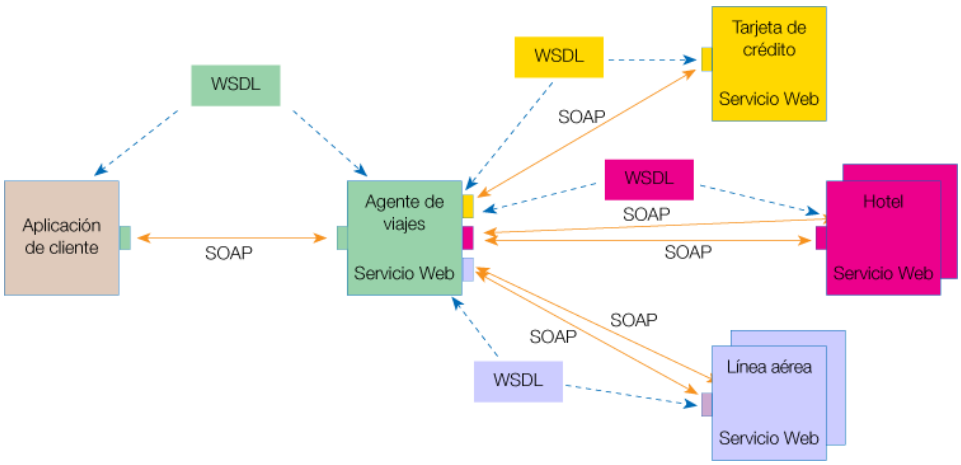

Figura 1.2: Ejemplo de un servicio web

El código SOAP del cuadro 1.1 muestra cómo se solicitaría un vuelo de Madrid a Londres.

[1] tomado de http://www.w3c.es/Divulgacion/GuiasBreves/ServiciosWeb

```
<?xml version='1.0' ?> <env:Envelope
   xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
 <m:reserva xmlns:m="http://agenciaviajes.ejemplo.org/reserva"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
   <m:referencia>
      uuid:A003595
   </m:referencia>
   <m:fechaReserva>2008-07-17</m:fechaReserva>
   <m:horaReserva>11:37CET</m:horaReserva>
  </m:reserva>
  <n:pasajero xmlns:n="http://miempresa.ejemplo.com/empleados"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
   <n:nombre>Paula Casado</n:nombre>
  </n:pasajero>
 </env:Header>
<env:Body>
  <p:itinerario
    xmlns:p="http://agenciaviajes.ejemplo.org/reserva/viaje">
   <p:trayecto>
     <p:tipo>vuelo</p:tipo>
     <p:origen>Madrid</p:origen>
     <p:destino>Londres</p:destino>
     <p:fechaSalida>2008/07/26</p:fechaSalida>
     <p:horaSalida>10:30</p:horaSalida>
    </p:ida>
 </env:Body>
</env:Envelope>
```

Cuadro 1.1: Ejemplo de mensaje SOAP

1.4. Coordinación de servicios Web

1.4.1. Modelos y lenguajes de coordinación

En [MC94] se define la *coordinación* como el proceso de gestionar dependencias entre actividades. Esta definición se centra en la interdependencia entre las actividades. En el ámbito de la programación, la coordinación se entiende como el proceso de separar la computación de los aspectos de comunicación ("*Programming=Computation + Coordination*" [GC92]).

Separando la computación de la coordinación se facilita, no sólo la reutilización de los componentes computacionales, sino también la de la lógica de la coordinación. Podemos concluir que la coordinación implica la interacción de entidades activas en determinados momentos y en determinados lugares siguiendo unas reglas previamente establecidas [ZW03].

Un *modelo de coordinación* es un marco conceptual para modelizar el espacio de interacción. Según [Cia96], los modelos de coordinación permiten representar los sistemas como ensamblajes multicomponentes, definiendo cuáles son las entidades cuya interacción mutua es reglada por el modelo, proporcionando las abstracciones que permiten la interacción entre las entidades, y expresando las reglas que gobiernan el sistema.

Por tanto, un modelo de coordinación define el medio de coordinación sobre el que los componentes actúan y las reglas que gobiernan la interacción entre los componentes y el propio medio de coordinación. Por otra parte, la interacción entre los componentes, según el modelo de coordinación, se programará mediante un lenguaje de coordinación.

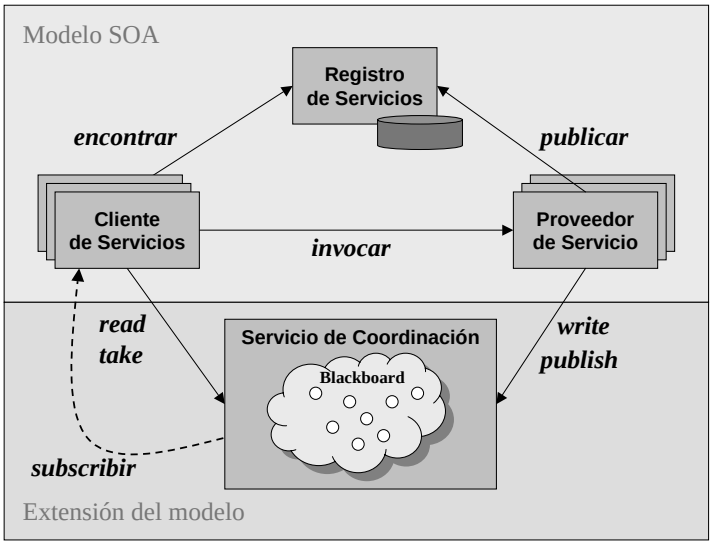

Figura 1.3: Extensión del modelo SOA

En [Álv03] se propone una extensión de la arquitectura SOA, añadiendo un nuevo rol, llamado *Coordinador de Servicios*, cuya responsabilidad es proporcionar modelos de

interacción adicionales entre los proveedores de servicios y sus clientes (ver figura 1.3). El coordinador está inspirado en el modelo de coordinación Linda, siguiendo el patrón de la arquitectura de pizarra [BMR+96], y debe proporcionar una serie de servicios para comunicar y sincronizar aplicaciones heterogéneas y distribuidas.

1.4.2. El modelo de coordinación Linda

El modelo de coordinación Linda [Gel85, CG89] se basa en la comunicación generativa. La comunicación generativa es un mecanismo de comunicación asíncrona entre procesos basada en una estructura de datos compartida. La comunicación asíncrona se realiza mediante la inserción y extracción de datos sobre el espacio compartido.

El espacio compartido recibe el nombre de *espacio de tuplas*, debido a que contiene un multiconjunto de tuplas producidas por los procesos. Una vez insertada una tupla en el espacio de tuplas, ésta tiene una existencia independiente del proceso que la generó. Las tuplas se representan mediante listas de valores (al estilo de Lisp), separados por comas y encerrados entre paréntesis.

Ejemplo 1.1. *Ejemplos de tuplas*

```
("Nueva York","Los Ángeles",2001/12/14,media-mañana)
(vehículo1,42.46583,-2.42716)
("Miguel Cervantes","El Quijote")
(vuelo,Madrid,Londres,10:00,12:00)
```

El modelo Linda fue concebido originalmente para sistemas cerrados, permitiendo la interacción y coordinación de un conjunto cerrado de procesos concurrentes ejecutados por sistemas de computación paralelos. Sin embargo, sus ideas son también adecuadas para sistemas abiertos distribuidos heterogéneos como los servicios Web. Si dos procesos necesitan comunicarse, no se requiere que los procesos estén fuertemente acoplados para enviar mensajes de un proceso a otro siguiendo algún protocolo, ni que compartan una variable. En el modelo Linda los procesos están desacoplados, comunicándose únicamente a través del espacio de tuplas, que tiene una existencia independiente a los mismos. Un proceso no necesita conocer nada acerca de los otros procesos, excepto la forma de las tuplas producidas o consumidas.

El primer lenguaje de coordinación para procesamiento paralelo y distribuido, basado en este paradigma y el más representativo, es el lenguaje Linda. Linda proporciona una abstracción para la programación concurrente mediante un reducido conjunto de primitivas de coordinación sobre el espacio de tuplas. Las primitivas de coordinación son ortogonales a cualquier lenguaje de programación y pueden ser añadidas a él. Comparado con otros modelos de procesamiento paralelo, Linda es más ortogonal porque considera la coordinación de procesos como una actividad separada de la computación. Su ortogonalidad permite que los procesos puedan ser codificados en diferentes lengua-

jes de computación y ejecutados en diferentes plataformas para interoperar usando las mismas primitivas.

Los procesos acceden al espacio de tuplas mediante cinco operaciones simples. En las primeras versiones aparecía una sexta operación, eval, que permitía crear un nuevo proceso desde una tupla activa. Con el paso del tiempo se comprobó que esta operación era difícil de definir semánticamente y, en la práctica, difícil de implementar, por lo que no se incluyó en la mayoría de las implementaciones. Además su funcionalidad ha quedado cubierta por otros medios, por ejemplo, se puede sustituir por la llamada a una función que cree un nuevo hilo por cada campo de la tupla activa que haya que evaluar y posteriormente añadir el resultado con la operación in (ver a continuación). Las cinco operaciones que actualmente se entiende que conforman el modelo básico Linda son las siguientes.

out Añade desde el proceso una tupla al espacio de tuplas

in Elimina una tupla del espacio de tuplas y se la devuelve al proceso. El proceso se bloquea si la tupla no está disponible.

rd Devuelve al proceso una copia de una tupla del espacio de tuplas. El proceso se bloquea si la tupla no está disponible.

inp Versión no bloqueante de la operación in. Si la tupla no está disponible se devuelve una indicación de fallo, en vez de bloquear al proceso.

rdp Versión no bloqueante de la operación rd.

Las operaciones de lectura (i.e. in, rd, inp y rdp) hacen uso de una técnica de *matching* asociativo o correspondencia. En este caso, se especifican los valores de un subconjunto de campos en la tupla, que son usados para localizar una tupla válida en el espacio de tuplas. Por ejemplo, si se busca una tupla que represente la posición de un vehículo (`longitud` y `latitud`) entonces debería utilizarse la siguiente operación para obtenerla:

$$in(vehículo1, ?x, ?y)$$

La especificación de la tupla usada en la operación in se denomina plantilla. Los campos sin especificar en la plantilla (en el ejemplo los valores x e y) se denominan *campos formales* o también como *wildcards*.

Una simple comunicación punto a punto entre dos servicios Web se puede modelar mediante la combinación de una operación out por parte del servicio emisor y de una operación in por parte del servicio receptor (ver figura 1.4). En este caso el *servicio1* deposita la tupla $(vehículo1, 42{,}46583, -2{,}42716)$ en el espacio de tuplas y el *servicio2* utiliza la plantilla $(vehículo1, ?x, ?y)$ para obtener la tupla. La plantilla está formada por un campo con un valor definido (i.e. *vehículo1*) y dos campos libres, cuyos valores

comienzan con el símbolo ?. Los valores 42,46583 y −2,42716 serán asignados, respectivamente, a las variables x e y, cuando la operación in tenga éxito. No es necesario que la operación de salida se invoque antes que la operación de entrada, porque la propiedad bloqueante de esta última hace que el *servicio2* espere hasta que se complete con éxito. Además, si en el espacio de tuplas hay más de una tupla que se corresponda con la plantilla, entonces cualquiera de ellas tiene la misma oportunidad de ser recogida por el *servicio2*.

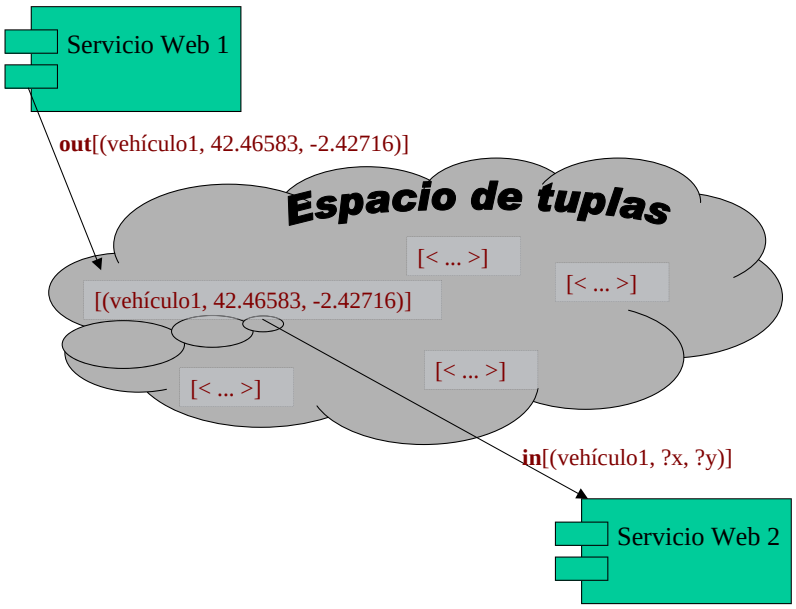

Figura 1.4: Comunicación mediante un espacio de tuplas

1.5. *JavaSpaces*

Pueden encontrarse implementaciones de Linda para diferentes lenguajes de programación. Dos de las implementaciones para Java más conocidas son *JavaSpaces* [FHA99], desarrollado por Sun Microsystems como un componente del proyecto *Jini* [Arn99]; y *TSpaces* [WMLF98], espacio de tuplas implementado por IBM como un simple servidor en red.

La tecnología *Jini* [Arn99] es una arquitectura de software abierta, definida por un conjunto de especificaciones, que describe la construcción de sistemas distribuidos en Java que sean fácilmente adaptables a los cambios. Sobre esta arquitectura se construye *JavaSpaces* [FHA99]. *JavaSpaces* proporciona un plataforma simple para diseñar e implementar sistemas distribuidos. Permite a un conjunto de procesos distribuidos colaborar mediante repositorios de objetos Java, llamados *espacios*. Un espacio es un repositorio compartido de objetos Java accesible a través de la red. Los procesos usan el espacio compartido para el almacenamiento permanente de objetos y como un medio de inter-

cambio de información. Con *JavaSpaces*, una aplicación es una colección de procesos que cooperan vía el flujo de objetos en o desde uno o más espacios.

Algunas de las propiedades de los espacios son:

Compartidos Los espacios son memorias compartidas accesibles a través de la red con las cuales pueden interactuar multiples procesos remotos concurrentemente.

Persistentes Los espacios proporcionan a los objetos un almacenamiento permanente fiable. Una vez que un objeto es almacenado en el espacio, permanecerá en él hasta que un proceso lo borre explícitamente. Los procesos pueden también especificar para cada objeto un tiempo máximo de permanencia (leasing time) después del cual el objeto será automáticamente eliminado del espacio.

Que los objetos sean persistentes conlleva que los objetos pueden permanecer en el espacio incluso después de que los procesos que los han creado hayan terminado.

Asociativos Los objetos son localizados en el espacio mediante una búsqueda asociativa, en lugar de una búsqueda por posición o identificador. La búsqueda asociativa permite encontrar objetos según su contenido, sin necesidad de conocer el nombre del objeto, ni el proceso que lo creó.

El mecanismo de *búsqueda asociativa* de *JavaSpaces*, es decir, por el contenido del objeto, es un simple emparejamiento de valores entre objetos de la misma clase. Para buscar un objeto en *JavaSpaces* se debe crear primero un objeto plantilla, cuyos campos pueden tener valores concretos o bien el valor nulo (*null*). Este valor se interpreta como un comodín. Así, un objeto almacenado en el espacio se corresponde con el objeto plantilla si: 1) ambos objetos son de la misma clase; y 2) los campos no nulos de la plantilla son iguales a los del objeto.

Transaccionalmente seguros La tecnología de *JavaSpaces* proporciona un modelo que asegura la atomicidad de las operaciones, es decir una operación sobre el espacio se completa o si no, no se realiza. La atomicidad se cumple para operaciones simples, pero también se pueden agrupar varias operaciones simple en un transacción, de forma que se si no se completan todas ellas, entonces no se realiza ninguna.

La interfaz de estos espacios de objetos permite a los procesos realizar un número reducido de operaciones (figura 1.5): la operación write, para escribir un nuevo objeto en el espacio; la operación take, para recuperar un objeto; o la operación read, para recuperar una copia de un objeto. Además, *JavaSpaces* extiende las capacidades del modelo Linda proporcionando mecanismos orientados a eventos básicos. Por ejemplo, la operación notify ofrece un estilo de programación más reactivo que permite a los procesos ser notificados cuando un determinado objeto sea insertado en el espacio.

Un espacio *JavaSpaces* contiene *entries*, objetos pertenecientes a alguna clase que implementa la interfaz `net.jini.core.entry.Entry` (cuadro 1.2). La interfaz `Entry` extiende a la interfaz `Serializable`, y está vacía, o sea, no tiene ningún método que

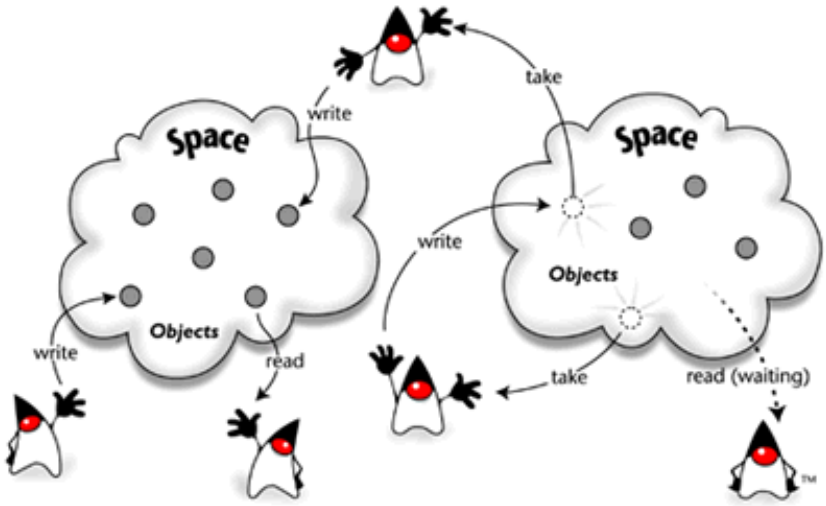

Figura 1.5: Operaciones sobre *JavaSpaces*

```
package net.jini.core.entry;

public interface Entry extends java.io.Serializable{
  // this interface is empty
}
```

Cuadro 1.2: La interfaz `Entry`

tenga que ser implementado. El único propósito de la interfaz es marcar una clase como apropiada para usarla dentro de un espacio. La clase necesita siempre un constructor sin argumentos `public` para que las *entries* entren y salgan del espacio.

Los campos de la clase que implementa la interfaz `Entry` se declaran necesariamente como públicos para permitir que otros procesos puedan encontrar los objetos del espacio basándose en el contenido de sus campos (búsqueda asociativa). Además, debido también al proceso de correspondencia, los campos de una *entry* deben contener referencias a objetos y no tipos primitivos.

Ejemplo 1.2. *Ejemplo de una entry*

```
package net.jini.core.entry;

public class tupla implements Entry{
  public String vehiculo;
  public Integer longitud;
  public Integer latitud;

public tupla(String vehiculo, Integer longitud, Integer latitud) {
```

```
this.nombre = vehiculo;
this.longitud = longitud;
this.latitud = latitud;
}
```

Las operaciones básicas sobre un espacio *JavaSpaces* son:

write Escribe la *entry* dada en el servicio *JavaSpaces*.

take Lee una *entry* del servicio *JavaSpaces* que se corresponde con la plantilla dada, eliminándola del espacio. Si la operación no encuentra ningún objeto, el proceso se bloquea hasta que llegue alguno.

read Lee una *entry* del servicio *JavaSpaces* que se corresponde con la plantilla dada. Igual que la operación take, si la operación no encuentra ningún objeto, el proceso se bloquea hasta que llegue alguno.

takeIfExists Versión no bloqueante de la operación in. Si la operación no encuentra ningún objeto, se devuelve una indicación de fallo, en vez de bloquear al proceso.

readIfExists Versión no bloqueante de la operación rd.

notify Notifica la llegada al espacio de entries que se corresponden con una plantilla dada previamente.

Capítulo 2

Descripción del Diseño e Implementación de un Servicio Web de Coordinación de Servicios Web (WCS)

Como ya se comentó en el capítulo anterior, Pedro Álvarez, en [ÁBM03b, Álv03], propone una extensión del modelo SOA, ampliamente utilizado en el desarrollo de aplicaciones basadas en servicios Web, añadiendo un nuevo rol llamado *Coordinador de Servicios*, cuya responsabilidad es proporcionar modelos de interacción adicionales entre los proveedores de servicios y sus clientes (ver figura 1.3). Este coordinador de servicios está inspirado en el patrón de la arquitectura de pizarra [BMR+96] y debe proporcionar una serie de servicios para comunicar y sincronizar aplicaciones heterogéneas y distribuidas.

En este capítulo vamos a describir un servicio Web, llamado *Servicio Web de Coordinación* (WCS) y desarrollado por Pedro Álvarez [Álv03, ÁBM+03c], que desempeña el papel de coordinador en una implementación concreta de la arquitectura basada en servicios. El servicio de coordinación está basado en el modelo Linda y hace el papel de un *broker* de mensajes. Esto es, el servicio actúa como intermediario entre los procesos externos y el núcleo interno basado en *JavaSpaces*. Dicho *broker* forma parte del desarrollo de un *middleware* para entornos basados en Web.

El servicio de coordinación proporciona servicios para comunicar y sincronizar aplicaciones heterogéneas distribuidas en Internet. Mediante este servicio las aplicaciones distribuidas pueden cooperar entre ellas, independientemente del sistema operativo sobre el que estén ejecutando y del lenguaje de programación en el que estén codificadas.

Conceptualmente, podemos ver al sistema compuesto por varias capas con el objetivo de hacer accesible *JavaSpaces* mediante Internet y desde aplicaciones desarrolladas en cualquier lenguaje y ejecutadas en cualquier sistema operativo. Partiendo del modelo de

comunicación generativa, cada capa está encapsulada en una capa de nivel superior hasta llegar a la capa accesible a través de Internet (ver figura 2.1). Este diseño está motivado por el hecho de que *JavaSpaces* sólo permite el acceso a aplicaciones desarrolladas en Java. Envolviendo la tecnología *JavaSpaces* con la capa basada en HTTP, el servicio de coordinación puede ser usado de forma remota por aplicaciones Web escritas en otros lenguajes de programación.

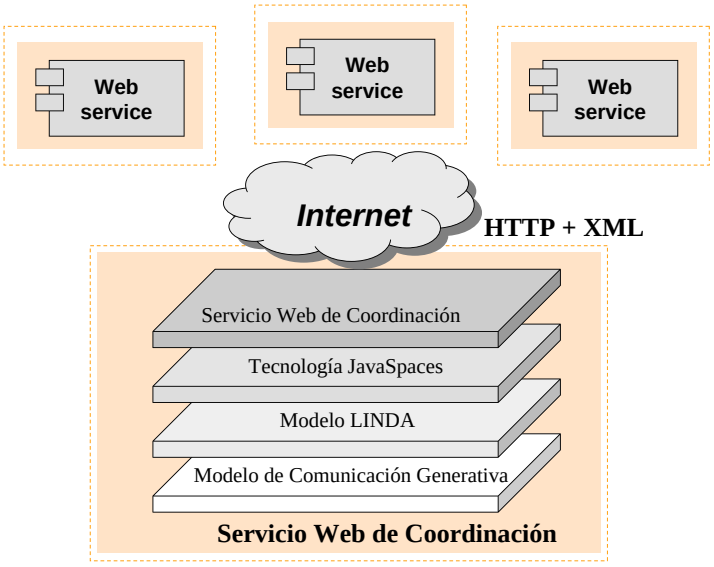

Figura 2.1: Representación conceptual del servicio de coordinación

2.1. Componentes del Servicio de Coordinación

Como puede verse en la figura 2.2, el servicio de coordinación diseñado está compuesto por tres componentes software: el *Componente Espacio de Tuplas XML*, el *Componente de Coordinación Java* y el *Componente de Coordinación HTTP*.

Sus roles y responsabilidades dentro del servicio de coordinación son los siguientes:

Componente Espacio de Tuplas XML. Es un espacio de interacción desarrollado sobre *JavaSpaces*. Es un repositorio de mensajes XML encapsulados como objetos Java. Permite que los procesos puedan comunicarse entre sí intercambiando mensajes XML a través del espacio de interacción en vez de comunicarse directamente. Su interfaz proporciona a los procesos el mismo conjunto de operaciones que *JavaSpaces*, pero mediante mensajes XML. Los procesos pueden escribir nuevos mensajes XML en el espacio, recuperar mensajes XML del mismo, o ser notificados cuando otros procesos escriben determinados mensajes en él. Además, no es necesario que los mensajes tengan una estructura previamente definida, sino que se van a poder definir nuevos formatos XML en tiempo de ejecución.

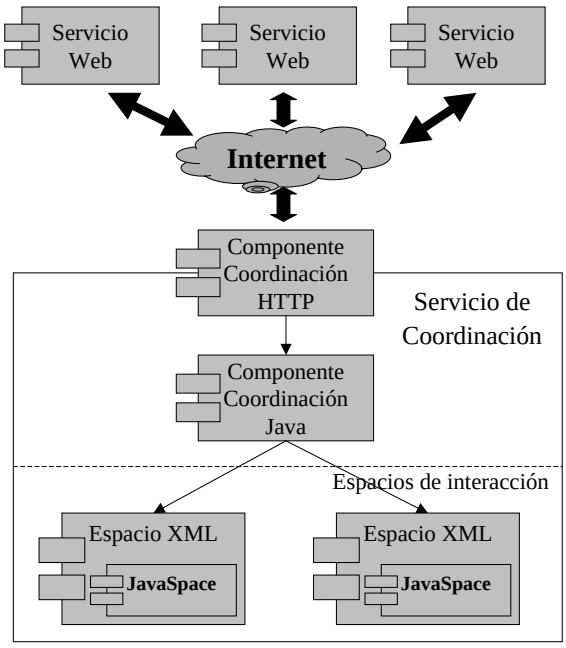

Figura 2.2: Diseño de alto nivel del servicio de coordinación

Componente de Coordinación Java. Este componente, desarrollado en lenguaje Java, es el núcleo del servicio de coordinación. Proporciona una colección de operaciones de coordinación divididas en dos interfaces diferentes: *Interfaz de Coordinación Básica* (BCI) y *Interfaz de Coordinación Reactiva* (RCI). La BCI ofrece un conjunto reducido de operaciones de coordinación y sincronización entre procesos basado en el modelo Linda. Estas operaciones permiten un estilo de programación en el que los procesos que invocan una operación pueden quedarse bloqueados hasta que se complete la operación. La RCI proporciona operaciones que permiten un estilo reactivo, de forma que un proceso puede manifestar su interés en generar un tipo específico de eventos, publicar estos eventos y suscribir su interés para recibir eventos de un tipo específico.

Componente de Coordinación HTTP. Proporciona el mismo conjunto de operaciones que el Componente de Coordinación Java a través de sus dos interfaces: *Interfaz de Coordinación Básica HTTP* (HBCI) y *Interfaz de Coordinación Reactiva HTTP* (HRCI). Estas interfaces ocultan los detalles de la plataforma sobre la que se está ejecutando el servicio y permiten la invocación de las operaciones de coordinación mediante el protocolo HTTP. También se han construido otros componentes funcionalmente similares, pero que utilizan otros protocolos de comunicación estándar como SOAP o SMTP.

2.1.1. Componente Espacio de Tuplas XML

Es un espacio de interacción de tuplas codificadas en formato XML. Su interfaz proporciona un conjunto de operaciones para escribir y recuperar tuplas XML del espacio, según el modelo de comunicación generativa, así como para suscribir el interés en ser notificado de la escritura de nuevas tuplas, permitiendo un modelo de programación reactivo.

Este componente no es una implementación de un repositorio de documentos XML creado desde cero, porque ése no era el objetivo del proyecto en ese momento, sino que se implementó como un servidor RMI (*Remote Method Invocation*) sobre *JavaSpaces* para proporcionar un espacio compartido en el que múltiples procesos pudieran interactuar concurrentemente a través de la red (figura 2.3).

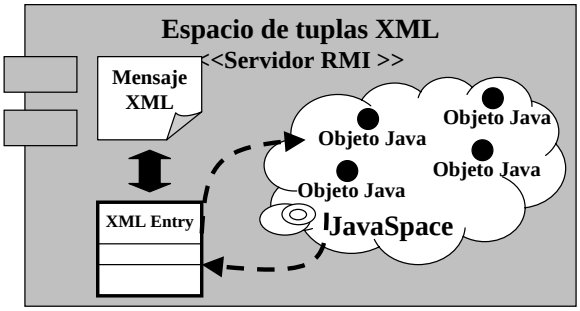

Figura 2.3: Componente Espacio de tuplas XML

La interfaz de este componente ofrece operaciones de alto nivel que pretenden mantener la semántica de las operaciones respectivas de *JavaSpaces*, pero su sintaxis ha sido adaptada para trabajar con tuplas XML. Define tres operaciones básicas: writeXML, readXML y takeXML, que son la versión con tuplas XML de las operaciones definidas en *JavaSpaces*: write, read y take.

La operación writeXML coloca una copia de la tupla XML en el espacio de tuplas encapsulado. Si se invoca varias veces, entonces se insertarán múltiples copias en el espacio. La operación de escritura tiene dos parámetros: un objeto de tipo `String`, que representa la tupla XML como una secuencia de caracteres; y un valor de tipo `entero` que especifica el tiempo máximo en milisegundos (*lease time*) que la tupla debe permanecer en el espacio.

El concepto de *leasing* no es propio del modelo básico Linda, pero sí aparece en *JavaSpaces* como una propiedad de *Jini* (véase [Arn99]). El parámetro *leasing time* permite especificar el tiempo máximo que una tupla puede permanecer en el espacio, después del cual la tupla será eliminada por el propio servicio. Su objetivo es una gestión más eficiente del espacio, permitiendo realizar periódicamente procesos de recolección de basura. En nuestra modelización formal no lo vamos a tener en cuenta ya que supondremos que solo se va a liberar el espacio de aquellas tuplas de las que se tiene la certeza de que ya no van a ser utilizadas por otros procesos y, por tanto, su eliminación no va a afectar

al comportamiento global del sistema. En la práctica, supondremos que los procesos invocan la operación writeXML asignando al parámetro *lease time* el valor *FOREVER*, que significa que el objeto se almacenará en el espacio indefinidamente (o hasta que la invocación de una operación takeXML lo elimine). Otra aproximación alternativa consistiría en modelar el uso del parámetro *lease time* añadiendo en el sistema distribuido un proceso que periódicamente consuma tuplas que han superado su *lease time*. Este proceso se comportaría como un proceso concurrente más, que retiraría periódicamente las tuplas obsoletas.

Por otra parte, conviene aclarar que el hecho de representar la tupla XML por medio de un `String` no va a causar ningún problema en este entorno. En ocasiones, cuando un `String` que representa un documento XML debe ser interpretado por un servidor en el contexto de un servicio Web, pueden aparecer problemas en el tratamiento de algunos de los caracteres especiales de marcado. Por ello, fue definido el tipo de dato *DOM*, para representar con fiabilidad los árboles XML en el marco de los servicios web [Jac06]. Sin embargo, en esta componente (y en las restantes), es seguro representar las tuplas XML como `String`, puesto que son utilizadas para un procesamiento por medio de programas Java, en los que los símbolos de marcado son tratados adecuadamente.

Siguiendo con el resto de operaciones de este componente, las tuplas pueden ser leídas o extraídas del espacio de interacción mediante los operadores readXML o takeXML. Ambas operaciones tienen dos parámetros: un objeto de tipo `String` que llamamos plantilla (*template*), que es una tupla XML cuyas etiquetas y valores se emparejan con las tuplas XML existentes en el espacio, y un valor entero, que representa un tiempo de espera máximo (*timeout*) para leer la tupla. La operación readXML devuelve una copia de alguna tupla XML del espacio que se corresponda con la plantilla. Si no existe ninguna tupla, el proceso se bloquea hasta que otra operación writeXML escriba una tupla XML que se corresponda con la plantilla especificada. Si el tiempo de espera expira sin haber obtenido la tupla, la operación se aborta y el proceso invocante se desbloquea. La operación takeXML es similar pero, además de devolver una copia de la tupla, la elimina del espacio. Consideraciones similares a las realizadas para el parámetro *lease time* nos permiten ignorar, en el estudio formal, el concepto de *timeout*.

La utilización de *JavaSpaces* en la construcción de este componente evitó tener que implementar todos aquellos aspectos relacionados con el almacenamiento y acceso concurrente a los datos, pero hizo aparecer otras dificultades en el desarrollo del servicio, derivadas de las decisiones de diseño de *JavaSpaces*. Por ejemplo, *JavaSpaces* trabaja únicamente con objetos Java. Si varios procesos cooperantes quieren interaccionar directamente con *JavaSpaces* es necesario que conozcan previamente la especificación de los objetos Java que van a ser intercambiados, puesto que las clases a las que pertenecen estos objetos han de ser definidas antes de la fase de compilación. Esto obligaría a los procesos cooperantes a conocer en tiempo de compilación, si no se toman las adecuadas precauciones, la estructura de los mensajes que se van a intercambiar.

Sin embargo, uno de los objetivos en la construcción del *Componente Espacio de Tuplas XML* era proporcionar a los procesos un modo de trabajo más flexible que *JavaS-*

paces. Mediante el uso de las operaciones writeXML, readXML y takeXML, en lugar de las correspondientes operaciones de *JavaSpaces* write, read y take, los procesos van a poder intercambiar mensajes XML (en vez de objetos Java), lo que permite que los procesos puedan estar escritos en cualquier lenguaje de programación). Además, el formato de estos mensajes va a poder ser definido en tiempo de ejecución. Pero, para que las tuplas XML puedan ser almacenadas en *JavaSpaces*, es necesario que sean transformadas a objetos Java cuando vayan a ser escritas en el repositorio y viceversa cuando vayan a ser recuperadas.

2.1.2. Componente de Coordinación Java

Como ya se ha comentado anteriormente, este componente es el corazón del servicio de coordinación. Está implementado en Java y está compuesto por dos interfaces. La *Interfaz de Coordinación Básica* (BCI) proporciona las mismas operaciones que el modelo básico de Linda, mientras que la *Interfaz de Coordinación Reactiva* (RCI) proporciona operaciones que permiten un estilo de programación basado en eventos (figura 2.4).

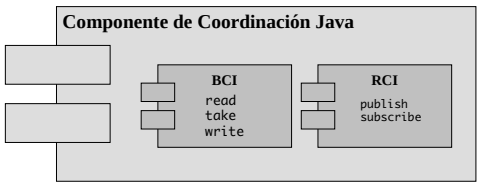

Figura 2.4: Componente de Coordinación Java

Cuando un proceso invoca una operación de coordinación, básica o reactiva, se crea un representante (*proxy*, agente o proceso interno) del proceso invocador, que lo representa dentro de este componente. Dependiendo de la operación invocada por el proceso externo, el representante estará especializado en comunicación, sincronización o comportamiento reactivo. De esta manera, los procesos externos delegan sus tareas de coordinación a sus respectivos procesos internos. Dentro del *Componente de Coordinación Java*, los procesos internos cooperan entre sí a través de uno o más espacios de tuplas XML. Los procesos internos se comunican intercambiando mensajes XML, definen condiciones de sincronización mediante plantillas basadas en XML y suscriben y notifican eventos codificados como mensajes XML.

Además de coordinarse con otros procesos, cada proceso interno debe ser capaz de informar sobre su estado interno y de proporcionar datos al proceso externo que representa. Esta conexión entre ambos se establece cuando se crea el proceso interno y permanece hasta que es destruido. La técnica usada para conectar cada proceso externo con su representante depende del entorno de ejecución de este componente.

La interfaz BCI declara tres operaciones básicas: write, read y take. La lista de parámetros formales es la misma para las tres operaciones: un objeto de tipo `Client`, un objeto de tipo `String` y un valor numérico. El primer parámetro contiene información

sobre el proceso externo que invoca la operación y encapsula un mecanismo de flujo de bytes para comunicarse con él. El segundo encapsula el mensaje XML. Finalmente, el tercer parámetro define el *lease time* o el *timeout*, dependiendo de si se trata de una operación de escritura o de lectura.

Cuando un proceso externo invoca alguna de las operaciones de la BCI, un servicio interno de este componente, llamado *Remote Proxy Support Service* (RPSS), crea el proceso interno especializado que gestiona su propia actividad una vez que ha sido creado. Según la operación invocada se pueden crear tres clases diferentes de procesos internos: *writer*, *reader* y *taker*. Ellos son los que cooperan mediante el *Componente Espacio de Tuplas XML*, en representación de sus respectivos procesos externos.

La RCI exporta un modelo reactivo basado en eventos mediante las operaciones advertise, unadvertise, publish, subscribe y `unsubscribe`. Análogamente, existe un servicio interno, en este caso denominado *Event Notification Service* (ENS), encargado de crear los procesos internos especializados cuando es invocada alguna de las operaciones de esta interfaz. No vamos a dar más detalles de esta interfaz porque en la modelización formal del servicio únicamente hemos tenido en cuenta las operaciones de la interfaz básica que siguen el modelo Linda.

2.1.3. Componente de Coordinación HTTP

Está compuesto por dos interfaces, *Interfaz de Coordinación Básica HTTP* (HBCI) y *Interfaz de Coordinación Reactiva HTTP* (HRCI) que encapsulan a las correspondientes interfaces, básica y reactiva, del *Componente de Coordinación Java*, para que el servicio pueda ser usado por otras aplicaciones web independientemente de la plataforma hardware y software en la que están ejecutando y del lenguaje de programación en el que están escritas. Esta aproximación es la base del modelo basado en servicios Web [FT02].

Este componente, además de ser una interfaz vía web del *Componente de Coordinación Java*, también tiene la responsabilidad de crear y de mantener abiertas conexiones HTTP para comunicar cada aplicación web que requiere un servicio de coordinación con su respectivo representante dentro del componente (mediante la técnica HTTP *streaming*), que consiste en mantener conexiones HTTP abiertas a través de las cuales se envían sucesivos paquetes de datos.

El núcleo de este componente ha sido implementado como un *servlet*, un programa Java que reside y se ejecuta en un servidor web, en este caso concreto en el servidor *Apache* (`http://www.apache.org`). Cuando una aplicación web envía una petición HTTP al servidor web, invocando alguna operación de coordinación de las interfaces HBCI o HRCI, el servidor la direcciona mediante un contenedor de *servlets*, en este caso el contenedor *Tomcat* (`http://tomcat.apache.org`), hacia el *servlet* apropiado que la procesa, y le proporciona al *servlet* un flujo de salida basado en caracteres que le permite enviar datos de texto a la aplicación web invocante.

2.2. El procedimiento de correspondencia en dos pasos

En el modelo Linda, para recuperar una tupla de un espacio de tuplas es necesario crear una plantilla que especifique en cada campo el valor que queremos que tenga la tupla en ese campo. En aquellos campos de la plantilla en los que no se especifique ningún valor concreto se admite que la tupla pueda tener cualquier valor. Recordar que el modelo básico de Linda contempla únicamente las tuplas como secuencias de valores. Tal y como hemos explicado en los apartados anteriores, el Servicio de Coordinación que estamos describiendo necesita que la comunicación se realice a través de estructuras más complejas (incluso más dinámicas que las clases, en las que se basa *JavaSpaces*). Si el Servicio de Coordinación tuviese que apoyarse en documentos XML cualesquiera, la gestión sería complicada y difícilmente podría relacionarse con un modelo tan simple (y, por tanto, con tan buenas propiedades) como es Linda. Felizmente, las aplicaciones industriales del Servicio de Coordinación (véase el último apartado de este capítulo) nos permiten delimitar el tipo de documentos XML a través de los que se realizará la coordinación. Concretamente, los árboles XML necesarios para el Servicio de Coordinación tienen las siguientes características:

- son *planos* (o *no anidados*); es decir, entre una etiqueta y su cierre no vuelve a haber un subárbol XML, sino que aparece un único terminal de un tipo básico (la unicidad excluye el caso multivaluado), y
- cada etiqueta aparece una única vez en cada árbol (excepto por su correspondiente cierre, obviamente).

Con esta clase de documentos XML tan sencillos, los conceptos de "esquema", sean asociados a DTDs o a XML Schema [MS06] coinciden con la idea intuitiva de "estructura" o "formato" de un mensaje XML. Abstrayendo de los detalles técnicos, llamaremos *tuplas XML* a secuencias ordenadas de pares *atributo/valor*, de la forma:

$$((atr_1, val_1) \ldots (atr_n, val_n))$$

donde cada atr_i representa el nombre (o etiqueta) de un atributo y cada val_i su valor asociado.

Análogamente diremos que su *esquema* es la tupla de atributos:

$$(atr_1, \ldots, atr_n)$$

Las secuencias serán ordenadas (en el sentido de que dos tuplas con los mismos pares, pero en distintas posiciones de la tupla serán consideradas distintas), por cercanía al modelo Linda (y también porque eso facilitará la gestión *serializada* de las mismas

en Java). También supondremos que los atributos no aparecen repetidos en una misma tupla.

Si permitimos que alguno de los valores de una tupla no aparezca, obtenemos la noción de *plantilla XML*, concepto similar al que aparece en Linda (los valores nulos representan los comodines o *wildcards* de Linda), pero en el que todos los atributos deben aparecer.

Adelantándonos a la descripción de la implementación del emparejamiento en el Servicio Web de Coordinación, podemos dar una primera idea intuitiva señalando que una tupla XML se corresponderá con una plantilla XML dada si se cumplen dos condiciones:

- la tupla y la plantilla tienen el mismo esquema XML (es decir, aparecen los mismos atributos en el mismo orden), y
- si la plantilla tiene un valor en alguno de sus campos, entonces la tupla tiene que tener los mismos valores en sus campos correspondientes; aquellos campos de la plantilla que no tengan ningún valor se comportan como comodines, permitiendo que la tupla tenga cualquier valor.

Pese a lo informal de la anterior explicación, destaquemos que el concepto de *valor nulo*, será representado de distintas maneras, dependiendo del contexto. Así, en un documento XML será identificado con la ausencia de valor (es decir, con la situación en la que XML admite la abreviación `<etiqueta/>`). Mientras que en una clase Java, un valor nulo será representado por `null`.

Ya hemos comentado, al explicar el *Componente Espacio de Tuplas XML*, que la decisión de implementar el espacio de tuplas XML sobre *JavaSpaces* exige que las tuplas sean transformadas a objetos Java cuando vayan a ser almacenadas en el repositorio, y viceversa cuando sean recuperadas. La clase Java que se definió para codificar cualquier tupla XML se llama `XMLSimpleEntry` (ver figura 2.5). Esta clase tiene dos campos estructurados, `tag-name` y `tag-value`, para almacenar, de forma ordenada, los nombres de los atributos (comúnmente llamados etiquetas o *tags*) y los valores de cada uno de los pares de la tupla XML respectivamente. Así, la etiqueta del primer par de la tupla XML se almacena en la primera componente del campo `tag-name` y su correspondiente valor en la primera componente del campo `tag-value`, y así sucesivamente con el restos de pares atributo/valor. Tanto las etiquetas como los valores se almacenan como cadenas de caracteres. Los campos `position` y `nameChannel` serán explicados más adelante, en este mismo capítulo.

Entre los métodos definidos en la clase `XMLSimpleEntry`, podemos destacar el método constructor con un parámetro de entrada de tipo `String`, que representa la tupla XML plana que se quiere codificar en objeto Java. Cuando se crea un nuevo objeto de la clase, el constructor evalúa la tupla XML y almacena los sucesivos pares atributo/valor en los dos correspondientes campos `tag-name` y `tag-value` del nuevo objeto.

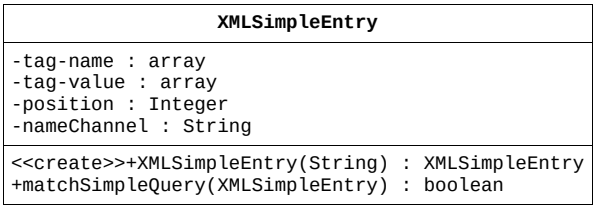

Figura 2.5: La clase XMLSimpleEntry

También tiene definido un método llamado matchSimpleQuery que implementa una regla de correspondencia simple entre el propio objeto que invoca al método y otro objeto de la misma clase pasado como parámetro. Este segundo objeto puede tener algunas de las componentes del campo `tag-value` con valor `null`. El método devuelve el valor verdad si se cumplen estas dos condiciones:

- Las tuplas XML representadas por ambos objetos tienen el mismo esquema XML, es decir, el campo `tag-name` de ambos objetos es el mismo (mismos atributos, en el mismo orden).
- Ambos objetos tienen los mismos valores (igualdad como cadenas de caracteres) en aquellas componentes del campo `tag-value` del objeto pasado como parámetro que sean cadenas distintas de `null`. Estas componentes nulas son así interpretadas como comodines.

El mecanismo de búsqueda asociativa de *JavaSpaces* no es adecuado para trabajar con objetos que contengan campos estructurados. Los objetos se emparejan comparando el contenido completo de cada uno de sus campos, y no permite buscar objetos especificando parcialmente valores en algunas de las componentes de un campo estructurado. Esto es debido a que los objetos se almacenan en el espacio utilizando la *serialización* de objetos Java, es decir, se guarda como una cadena de bytes que codifica todos sus campos y valores. En el caso de un campo estructurado, su valor es una única cadena resultante de la serialización de todas sus componentes, lo que imposibilita aplicar el operador de igualdad entre componentes individuales.

Precisamente, como los objetos de la clase `XMLSimpleEntry` almacenan los valores de las tuplas XML que representan en un campo estructurado, se hizo necesario construir un procedimiento de correspondencia en dos fases. En el primer paso se obtenía un objeto que tuviera el mismo esquema XML que la plantilla, haciendo uso del mecanismo de búsqueda de *JavaSpaces*, y posteriormente se comprobaba si también había una correspondencia de valores, invocando al método `matchSimpleQuery` explicado anteriormente. Una descripción más detallada del procedimiento es la siguiente:

1. En el primer paso se hace uso del mecanismo de correspondencia de *JavaSpaces*. En este paso sólo se considera el campo `tag-name` y no se tienen en cuenta sus valores correspondientes. Para ello, se crea una copia de la plantilla original en la que se

sustituyen todos los valores del campo `tag-value` por valores nulos. Con este nuevo objeto plantilla se ejecuta la operación de lectura read proporcionada por la interfaz de *JavaSpaces*, la cuál devolverá un objeto del repositorio que representará una tupla XML con el mismo esquema XML que la plantilla original.

2. En el segundo paso se invoca al método `matchSimpleQuery` del objeto recuperado, usando la plantilla original como parámetro real. Este método comprueba que cada componente no nula del campo `tag-value` de la plantilla original tenga el mismo valor que la correspondiente componente en el campo `tag-value` del objeto recuperado. Si el método devuelve el valor verdad significa que el objeto recuperado se corresponde con la plantilla. En caso contrario, significa que el objeto recuperado tiene el mismo esquema XML que la plantilla pero no se corresponden sus valores. En este caso, habría que volver al primer paso para seleccionar otro objeto con el mismo esquema XML y volver a aplicar este segundo paso.

Sin embargo, como puede comprobarse fácilmente, este procedimiento en dos pasos no garantizaba la recuperación de un tupla válida del espacio, aun conociendo su existencia. En el segundo paso, si el método matchSimpleQuery devuelve el valor `false` entonces se invoca nuevamente a la operación read de *JavaSpaces* con la misma copia del objeto plantilla. Debido al principio oportunista del modelo de comunicación generativa, en la que se basan *JavaSpaces* y Linda, es posible que esta operación de lectura recupere del espacio el mismo objeto `XMLSimpleEntry`, u otro objeto que tampoco satisfaga la segunda condición. Por tanto, es posible entrar en un bucle infinito que continuamente recupere en el primer paso objetos que posteriormente no pasen el segundo paso, aun sabiendo con certeza que existan objetos en el espacio que se correspondan con la plantilla original.

Lo deseable es que cada operación read, dentro del mismo procedimiento de lectura de una tupla XML, recupere un objeto `XMLSimpleEntry` diferente. Para conseguir este comportamiento ideal se necesitaba un patrón de comunicación más avanzado que permitiera acceder a un conjunto de objetos `XMLSimpleEntry` de forma aleatoria y sin repetir ninguno de ellos. Esta reflexión llevó al concepto de canal.

2.3. El concepto de canal

Como adaptación a la semántica de Linda, se modificó el mecanismo de búsqueda de forma que, en el primer paso, un proceso pueda acceder a todos los objetos almacenados que codifican tuplas con un mismo esquema XML. Para ello se dividió el espacio de tuplas en particiones disjuntas. Cada partición, denominada *canal*, contiene todos los objetos que codifican tuplas XML que comparten el mismo esquema. Por lo tanto, cada canal representa una estructura XML distinta almacenada en el repositorio (figura 2.6). Así, cuando se invoca una operación de lectura, sólo se busca entre las tuplas del canal correspondiente al esquema XML de la plantilla de búsqueda.

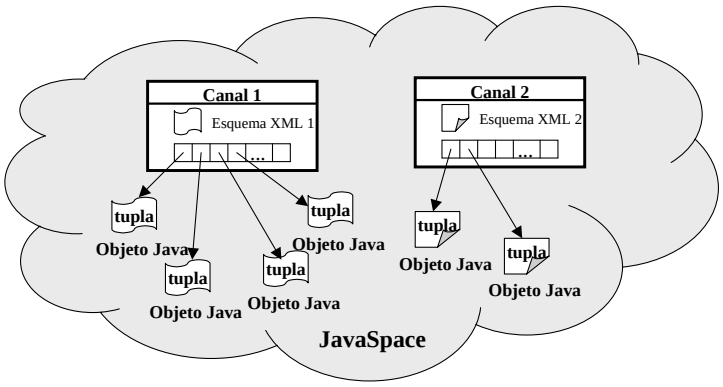

Figura 2.6: Organización del espacio de tuplas en canales

Los canales se implementan mediante la clase `RandomChannel` (figura 2.7). Cada objeto de la clase `RandomChannel` representa un canal de información distinto, y estará compuesto por todos los objetos `XMLSimpleEntry` del espacio que representan tuplas XML que comparten el mismo esquema XML. Los objetos `XMLSimpleEntry` se organizan en el espacio en canales dependiendo del esquema de la tupla XML que codifican. Por tanto, cada canal creado tiene su correspondiente objeto `RandomChannel`, que también se almacena en *JavaSpaces*, compartiendo el espacio con los objetos `XMLSimpleEntry`, para que sea accesible por cualquier proceso que use el canal. Cualquier proceso puede recoger este objeto, modificar su contenido, y escribirlo otra vez en el espacio para hacer los cambios permanentes. Además, el servicio de coordinación mantiene un registro de los canales existentes. Este registro está implementado en la clase `ChannelRegister`.

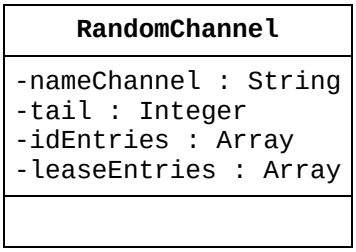

Figura 2.7: La clase RandomChannel

La clase `RandomChannel` tiene dos atributos básicos, el atributo `nameChannel` de tipo `String` para identificar el canal y el atributo `tail` de tipo `Integer` para marcar la siguiente posición en el canal. El atributo `nameChannel` contendrá la cadena de caracteres que determina el esquema XML de las tuplas que se van a almacenar en el canal (a pesar de que puede ser una cadena muy larga, será un identificador único del canal, puesto que será distinto para cada canal). Para gestionar los objetos `XMLSimpleEntry` del canal, se definen en la clase `RandomChannel` otros dos atributos estructurados, `idEntries` y `leaseEntries`. Cada componente del campo `idEntries` representa una identidad de un objeto `XMLSimpleEntry`, y la correspondiente componente del campo `leaseEntries` su tiempo de permanencia. El canal necesita mantener este tiempo de permanencia del objeto para eliminarlo automáticamente cuando su tiempo haya finalizado. Por tanto,

los objetos pueden ser eliminados del canal porque un proceso lector lo ha recuperado o porque su tiempo de vida ha expirado.

El canal gestiona información sobre los objetos `XMLSimpleEntry` asociados a él que están almacenados en el espacio en cada momento. Los objetos se organizan en el canal como una lista ordenada y se identifican mediante su posición en el canal. Cuando se añade un objeto en el espacio se le asigna la siguiente posición en su correspondiente canal y se almacena la identidad del nuevo objeto insertado en el campo `idEntries` del canal. Si el objeto insertado codifica una tupla cuyo esquema no se corresponde con ninguno de los esquemas ya almacenados en el espacio, entonces se crea un nuevo canal (creando un nuevo objeto `RandomChannel`) y se añade al registro `ChannelRegister` como paso previo a escribir la tupla. Por otra parte, cuando se recupera un objeto `XMLSimpleEntry` del espacio hay que borrar su información asociada al canal y actualizar el campo `tail` en el objeto `RandomChannel`.

Además, para que cada objeto almacenado en el espacio sepa a qué canal pertenece, hubo que extender la clase `XMLSimpleEntry`, añadiendo el atributo `position` de tipo `Integer`, para almacenar la posición del objeto en el canal, y el atributo `nameChannel` de tipo `String`, para identificar el canal donde está el objeto.

2.3.1. Operaciones básicas sobre el espacio de tuplas XML con canales

Como se acaba de explicar en el apartado anterior, en el espacio de interacción de *JavaSpaces* van a convivir dos tipos de objetos. Por una parte, los objetos `XMLSimpleEntry`, que representan a las tuplas XML que se van a intercambiar los procesos, y por otra, los objetos `RandomChannel`. Existirá un objeto `RandomChannel` por cada esquema XML (o canal) que esté representado en el espacio, de forma que cada objeto `RandomChannel` define un canal distinto y controla el acceso a todos los objetos `XMLSimpleEntry` pertenecientes al mismo. Además la clase `ChannelRegister` mantiene un registro de los canales existentes.

Cuando se ejecuta una operación de escritura writeXML sobre el espacio de tuplas, la primera acción que realiza el sistema es una consulta al registro `ChannelRegister` para saber si ya existe el canal o si es necesario crearlo. A continuación se extrae del espacio el objeto `RandomChannel` que gestiona al canal correspondiente, mediante una llamada a la operación take de *JavaSpaces*. Obsérvese que, en la anterior descripción, se está trabajando con dos espacios de tuplas *distintos*: una operación de escritura (writeXML sobre el espacio de tuplas XML) requiere una operación de lectura (take) sobre un dominio *JavaSpaces*. Esta convivencia de dos espacios de tuplas distintos será reflejada de forma ajustada en el modelo formal que presentaremos en el capítulo 3.

Puesto que la operación de *JavaSpaces* take es una operación bloqueante, el objeto `RandomChannel` actúa como un semáforo para los procesos que quieren acceder al canal. Su presencia en el espacio significa que el semáforo está abierto y que se puede acceder

a las tuplas del canal. Su ausencia significa que el semáforo está cerrado y los procesos invocantes se bloquearán hasta que nuevamente sea insertado en el espacio. De esta forma se garantiza que mientras se completa la operación ningún otro proceso va a tener acceso al canal y, por tanto, no va a poder añadir o eliminar objetos. La última acción consistirá en la inserción nuevamente del objeto `RandomChannel`, actualizado con la información del nuevo objeto `XMLSimpleEntry` que ha sido almacenado.

En la implementación de la operación readXML, usando canales, el procedimiento de correspondencia en dos pasos se realiza ahora de la siguiente manera. Cada vez que se invoca una operación de lectura, se obtiene una copia, mediante una llamada a la operación read de *JavaSpaces*, del objeto `RandomChannel` que controla el canal correspondiente al esquema XML de la plantilla especificada en la operación. Esta copia es utilizada por la búsqueda para: i) garantizar la posibilidad de recuperar todas las tuplas almacenadas; y ii) marcar aquellas que ya han sido comprobadas en el segundo paso y no se corresponden con la plantilla de búsqueda. De este manera se evita que se repita la comprobación de la misma tupla. Además, si existe una tupla que se corresponda con la plantilla, se garantiza que en algún momento se encontrará. El acceso a las tuplas indexadas por el canal se realiza de forma aleatoria para garantizar que todas las tuplas tengan la misma oportunidad de ser recuperadas y para simular el indeterminismo definido en el modelo Linda.

Mientras se realiza este procedimiento de lectura se permite la ejecución concurrente de otras operaciones sobre el canal original, tanto de lectura como incluso de escritura o de eliminación. Pero podría ocurrir que, de forma concurrente, otra operación takeXML eliminara una tupla del canal antes de ser examinada. Para abordar este caso, en cada iteración, la tupla se recupera invocando a la operación readIfExist de *JavaSpaces*. Si existe, se lleva una copia de la tupla para validarla. Si no existe, la marca en el canal como no válida y continúa el procedimiento en busca de otra tupla que sea válida, de entre las existentes en el canal. Si una vez recorrido todo el canal no se ha encontrado ninguna tupla válida, la operación readXML se bloquea a la espera de que algún proceso inserte una nueva tupla en el canal.

La posibilidad de acceder al canal por parte de otros procesos, mientras un proceso está realizando una operación de lectura sobre el mismo, podría hacer pensar en un primer momento que la operación readXML no cumple la semántica de Linda porque podría ocurrir que una tupla válida que inicialmente esté en la copia del objeto `RandomChannel` sea eliminada por otro proceso antes de que termine la operación de lectura. Sin embargo, hay que tener en cuenta que la única forma de comprobar el estado del espacio de tuplas es mediante el resultado de las operaciones de lectura. Como la operación de lectura no termina hasta que se invoca a la operación readIfExist para comprobar su existencia y validación, es en ese preciso instante en el que se conoce el estado del espacio de tuplas, y por con tanto es también el instante en el que se ejecuta la operación de lectura.

En la implementación del bloqueo de la operación de lectura se hace uso de la operación notify de *JavaSpaces* para ser notificados siempre que un nuevo objeto

`XMLSimpleEntry` se registra en el canal. Conocida la posición de este objeto en el canal, se recupera del espacio una copia del mismo para comprobar si se corresponde con la plantilla de la operación de lectura. Si es así, se devuelve el objeto y se desbloquea la operación.

La operación takeXML es muy similar a la operación readXML. La principal diferencia es que, una vez encontrado un objeto que se corresponde con la plantilla, éste es eliminado del canal. Además, hay que recuperar el objeto `RandomChannel` para borrar la información relativa al objeto eliminado y volverlo a escribir en el espacio.

2.3.1.1. Descripción algorítmica de las operaciones de lectura y escritura

Recapitulando la anterior descripción, cuando se invoca la operación readXML para leer una tupla XML del espacio, comprobando que concuerde con una plantilla XML, se ejecuta la siguiente secuencia:

1. Codificar la plantilla XML, pasada como parámetro, a un objeto `XMLSimpleEntry`, haciendo una copia, pero en la que se asigna el valor `null` a su campo `tag-value`.
2. Localizar el correspondiente canal, dado el esquema XML de la plantilla. Obtener el nombre del canal.
3. Tomar, haciendo una copia, el objeto `RandomChannel` que representa al canal. La copia va a ser modificada localmente para completar la operación de lectura.
4. Mientras el campo `idEntries` no esté vacío, seleccionar aleatoriamente el identificador de un objeto `XMLSimpleEntry` de entre los objetos del canal.
   a) Obtener una copia del objeto `XMLSimpleEntry` identificado.
   b) Invocar al método `matchSimpleQuery` del objeto recuperado pasándole la plantilla original como parámetro. Si devuelve el valor `true`, entonces el objeto se corresponde con la plantilla y la operación de lectura termina devolviendo este objeto. En otro caso, se borra, de la copia local del campo `idEntries`, el identificador del objeto con el fin de evitar volverlo a leer. Estas dos acciones se repiten hasta encontrar en el canal un objeto que sea válido.
5. Si no se ha encontrado ningún objeto en el canal que se corresponda con la plantilla original, entonces la operación de lectura se bloquea hasta que un nuevo objeto sea almacenado en el canal, momento en el que se comprueba si este nuevo objeto se corresponde con la plantilla.

La descripción para la operación takeXML es similar, salvo que en caso de emparejamiento con éxito el elemento devuelto es eliminado de la memoria compartida.

Cuando un proceso externo invoca la operación writeXML se realiza la siguiente secuencia de acciones:

1. Codificar la tupla XML, pasada como parámetro, a un objeto `XMLSimpleEntry`.
2. Consultar el registro `ChannelRegister` para determinar si ya existe el canal o si es necesario crearlo.
3. Si no existe el canal, crear el objeto `RandomChannel` correspondiente, formando su nombre a partir de las etiquetas XML, y registrarlo en el objeto `ChannelRegister`.
4. Tomar del espacio el objeto `RandomChannel` correspondiente al canal, dado el esquema XML de la tupla, mediante una operación take.
5. Obtener el número de la siguiente posición en el canal.
6. Asignar el nombre del canal y el número de posición al objeto `XMLSimpleEntry`.
7. Añadir a la información del canal la identidad de nuevo objeto y su tiempo de existencia.
8. Escribir los objetos `XMLSimpleEntry` y `RandomChannel` en el espacio, mediante sendas operaciones write.

2.4. Aplicación industrial: Servicios basados en la localización

Actualmente, los sistemas de posicionamiento son ampliamente conocidos por el público general y su uso se está convirtiendo en un hecho cotidiano. Entre ellos, el más popular y extendido es el GPS (*Global Positioning System*). Hoy en día ya no sorprende a nadie conducir un coche equipado con un dispositivo GPS que va indicando la ruta a seguir para llegar hasta su destino, ni cruzarse en un parque con un corredor que lleva en la muñeca un reloj pulsómetro GPS que, además de mostrarle las pulsaciones, registra el recorrido realizado para poder visualizarlo posteriormente en un servidor de mapas (i.e. *Google Maps*).

La tecnología utilizada por estos sistemas de posicionamiento (satélites, dispositivos móviles, PDAs, teléfonos móviles, etc.) permite determinar y seguir la posición de un objeto, persona o vehículo con bastante precisión. Esta capacidad de posicionamiento ha facilitado la aparición de los denominados *Servicios Basados en la Localización* (SBL). Algunos ejemplos de aplicaciones en el contexto de los SBL son:

- Seguimiento y localización de vehículos.
- Búsqueda de rutas más apropiadas.
- Asignación de recursos (autobuses, ambulancias, quitanieves, empleados, etc.) a servicios.

- Información en tiempo real sobre la situación del tráfico.
- Información turística localizada.
- Información contextual: gasolinera más próxima, servicios cercanos (cajeros, restaurantes, farmacias, etc).

Para el desarrollo de los sistemas basados en la localización es necesaria la integración de múltiples tecnologías: métodos y aparatos para determinar la posición, comunicaciones inalámbricas, dispositivos móviles, sistemas e infraestructuras de información geográfica, bases de datos espaciales, sistemas distribuidos, etc., y todo ello con Internet como *middleware* subyacente. El concepto clave para lograr esta integración es la *interoperabilidad* entre sistemas heterogéneos, es decir, la habilidad de éstos para intercambiar información y usarla de forma transparente. Podemos distinguir dos niveles de interoperabilidad:

Interoperabilidad sintáctica Capacidad para comunicarse e intercambiar datos. Para alcanzarla son necesarios formatos de datos y protocolos comunes.

Interoperabilidad semántica Capacidad para intercambiar información, de forma que la información transferida contenga el significado suficiente para que el sistema que la recibe pueda interpretarla automáticamente y produzca los resultados esperados por los usuarios finales. Para lograrla son necesarios, además de la interoperabilidad sintáctica, lenguajes de representación de conocimiento u *ontologías* comunes (o bien técnicas que asocien automáticamente las definiciones de un sistema con las definiciones usadas por el otro, como la que se describe en el apartado 4.1).

Una de las propuestas para lograr esta integración es la arquitectura de servicios Web. En esta arquitectura el sistema está compuesto por una colección de servicios cuya funcionalidad es accesible a través de la red y cuyas interacciones se resuelven mediante una serie de estándares dependientes del contexto basados en XML. En el contexto de los SBL podemos destacar, entre otros, los propuestos por el consorcio OpenGIS (OGC, `http://www.opengis.org`) en el dominio de los Sistemas de Información Geográfica (SIG) o los definidos por el *Location Interoperability Forum* (LIF), ahora integrado en la *Open Mobile Alliance* (OMA), (`http://www.openmobilealliance.org/`) en el dominio de los servicios de localización. Estos estándares complementan a los estándares de propósito general y a la infraestructura que proporciona Internet.

Sin embargo, la arquitectura de servicios Web carece de un modelo de interacción que soporte los requisitos de comunicación y coordinación necesarios en el contexto de los SBL, contexto muy dinámico y cooperativo. La solución propuesta por Álvarez en [Álv03] (ver figura 2.8) consiste en la integración de dos *frameworks* de servicios Web, el *framework* basado en OpenGIS (parte izquierda de la figura), y el *framework* basado en LIF (parte derecha), mediante un servicio Web de coordinación. El *framework* basado

en el trabajo de OpenGIS consta de una colección de servicios estándar SIG capaces de almacenar, consultar, visualizar y analizar diferentes tipos de datos geográficos (por ejemplo, WFS (*Web Feature Service*) servidor de datos geográficos [OGC01b]; WMS (*Web Map Service*) servidor de mapas básico [OGC03]; el *Servicio Geocoder* [OGC01a]; o el *Servicio Gazetteer* [OGC00]). El *framework* basado en las especificaciones de LIF está formado por un conjunto de servicios de localización que permiten interactuar con los dispositivos móviles remotos a través de redes inalámbricas.

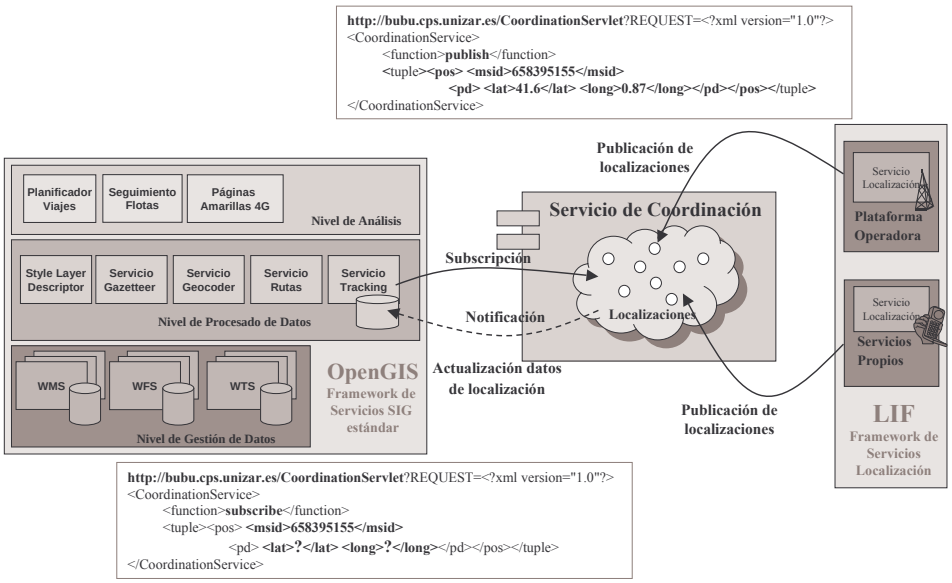

Figura 2.8: Patrón de integración de servicios basada en un servicio de coordinación

La solución de integración propuesta por Álvarez está dirigida por los datos y se basa en tres entidades (figura 2.8): 1) el servicio *tracking*, que es un servidor de geodatos para el almacenamiento y consulta de la información sobre la última localización conocida de los distintos dispositivos móviles; 2) los distintos servicios de localización del *framework* inspirado en LIF; y como mediador 3) un servicio de coordinación capaz de actualizar las localizaciones almacenadas en el servicio *tracking* a partir de los datos recibidos por los servicios de localización del *framework* LIF. Este servicio de coordinación sirve como repositorio de los datos de localización publicados por los servicios de localización, y notifica de una nueva localización a aquellos servicios del *framework* OpenGIS que estén interesados en recibir localizaciones actualizadas de los dispositivos móviles. Estas tres entidades cooperan a través de Internet de forma desacoplada y utilizando los distintos estándares de propósito general y de dominio involucrados en la integración.

El siguiente código muestra un ejemplo simplificado de invocación de HTTP para publicar una localización codificada en XML en el servicio de coordinación:

```
http://bubu.cps.unizar.es/
   CoordinationServlet?REQUEST=<?xml version"1.0"?>
     <CoordinationService>
```

```
   <function>out</function>
   <tuple> <id>658395155</id> <lat>41.6</lat> <long>0.87
   </long> </tuple>
</CoordinationService>
```

La publicación de la localización generará la siguiente tupla:

((id 658395155) (lat 41.6) (long 0.87))

Nótese que la estructura XML es "simple" (no anidada, plana), pero que sirve sobradamente a los efectos de coordinación.

Este patrón de integración ha sido aplicado con éxito en en el desarrollo de diversos sistemas (fruto de proyectos en los que Pedro Álvarez ha participado): Seguimiento de quitanieves; ayuda a la toma de decisiones para la asignación de técnicos de campo a órdenes de servicios en un sistema de empresa CRM; asignación de ambulancias a emergencias; y seguimiento de autobuses urbanos e información al usuario. A continuación describimos brevemente los dos primeros casos.

En el sistema de seguimiento de quitanieves, el objetivo era el seguimiento de las máquinas quitanieves que trabajan en el departamento francés de *Rhone-Alpes*. Debido a las características orográficas del terreno, existían tres zonas de cobertura de radio sin enlaces entre ellas. Las máquinas quitanieves podían moverse libremente por el departamento, por lo cual, en un momento dado, podían estar trabajando en cualquiera de las tres zonas. Como no podía realizarse un seguimiento en tiempo real, dado que no era posible recibir continuamente localizaciones vía radio cuando las quitanieves cambiaban de zona, se propuso una solución en la que el proceso de adquisición de los datos de localización se distribuía entre las tres zonas.

Como se muestra en la figura 2.9, el sistema consta de:

- Un servicio de localización en cada una de las zonas de cobertura de radio. Cada servicio puede comunicarse vía *trunking* únicamente con aquellas quitanieves que se encuentran dentro de su zona de cobertura.
- Un centro de control.
- Un *framework* de servicios Web diseñados a partir de los estándares de OpenGIS y LIF.
- Un servicio de coordinación que garantiza el trabajo cooperativo.

Cuando el sistema está activo, los dispositivos GPS instalados en las quitanieves envían periódicamente su localización al servicio de localización disponible en su correspondiente zona de cobertura, el cual las publica inmediatamente vía HTTP en el servicio de coordinación. La localización publicada estará codificada en XML, facilitando la interoperabilidad entre servicios.

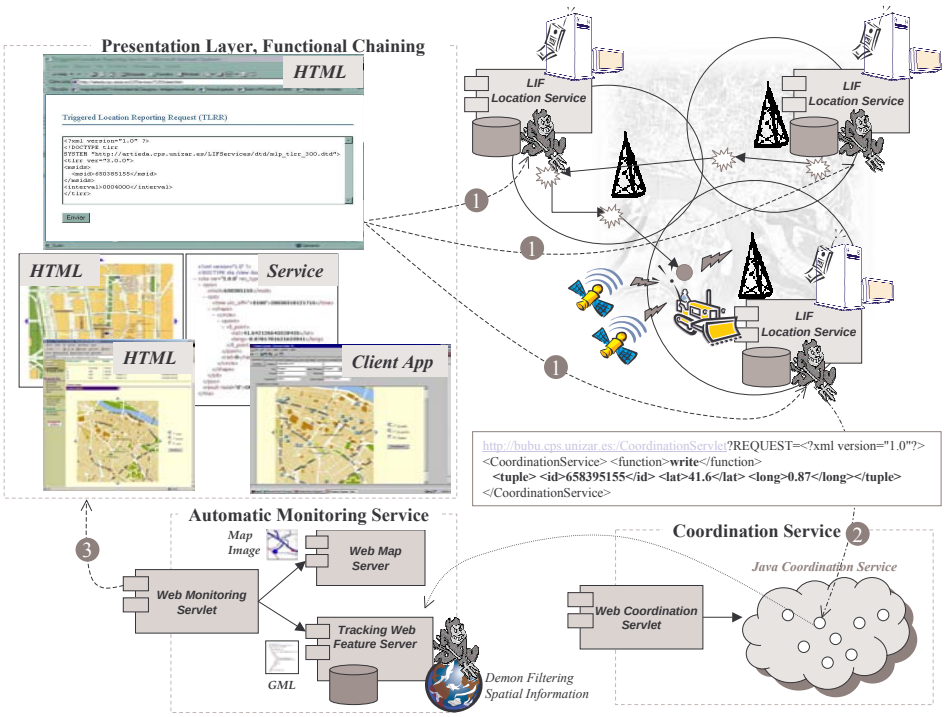

Figura 2.9: Sistema para el seguimiento de quitanieves

El servicio de *tracking*, que forma parte del *framework* de servicios Web basados en OpenGIS, está suscrito al servicio de coordinación para ser notificado de la publicación de nuevas localizaciones. De esta manera el servicio de *tracking* mantiene actualizada la localización de aquellos dispositivos móviles en cuyo seguimiento se está interesado. El intercambio de información entre los servicios de localización y el servicio de *tracking* mediante un servicio de coordinación posibilita que el servicio de *tracking* no necesite tener ningún conocimiento previo sobre los servicios de localizaciones. Sólo necesita conocer cómo suscribirse al servicio de coordinación. Una vez subscrito, las localizaciones son notificadas independientemente de cuál es el proveedor más apropiado o disponible en cada momento.

El segundo caso es la ayuda en la toma de decisiones para la asignación de técnicos de campo a órdenes de servicios en un sistema de empresa CRM. Los CRM (*Customer Relationship Management*) son sistemas informáticos de apoyo a la gestión de las relaciones con los clientes. Estos sistemas permiten a las empresas recopilar mayor información sobre sus clientes con el objetivo de ofrecerles servicios más adecuados a sus necesidades. La estrategia de negocio de las empresas que utilizan estos sistemas está enfocada a la satisfacción del cliente como medio para alcanzar su fidelidad. El uso de información geográfica y la integración con la localización de los clientes y de los empleados móviles de la empresa permite dar un servicio más eficiente y personal.

El objetivo de este proyecto era integrar servicios de localización en el CRM *Van-*

tive [1]. Mediante una solución cliente/servidor, este CRM automatiza e integra actividades de venta, marketing, centros de llamadas, transacciones, inventario y calidad. La integración de servicios de localización en *Vantive* permite funcionalidades como: localizar en "tiempo real" los recursos móviles de la empresa (por ejemplo, los técnicos de campo); almacenar y consultar geodatos relacionados con la actividad (direcciones de clientes, proveedores, almacenes, rutas que deben realizar los técnicos, etc.); o visualizar sobre mapas digitalizados los geodatos estáticos y dinámicos.

En concreto, el problema era integrar los servicios de localización con aquellas componentes de *Vantive* dedicadas a la gestión de las peticiones de servicio solicitadas por los clientes, que son *Vantive Field Service* y *Vantive Web Field Service*. La primera componente gestiona las peticiones de servicio de los clientes y planifica la asignación de personal para su realización en los correspondientes domicilios, mientras que la segunda permite a los técnicos de la empresa consultar sus órdenes de servicio y acceder al soporte necesario para su realización, todo ello a través de comunicaciones móviles y un computador portátil (ver figura 2.10).

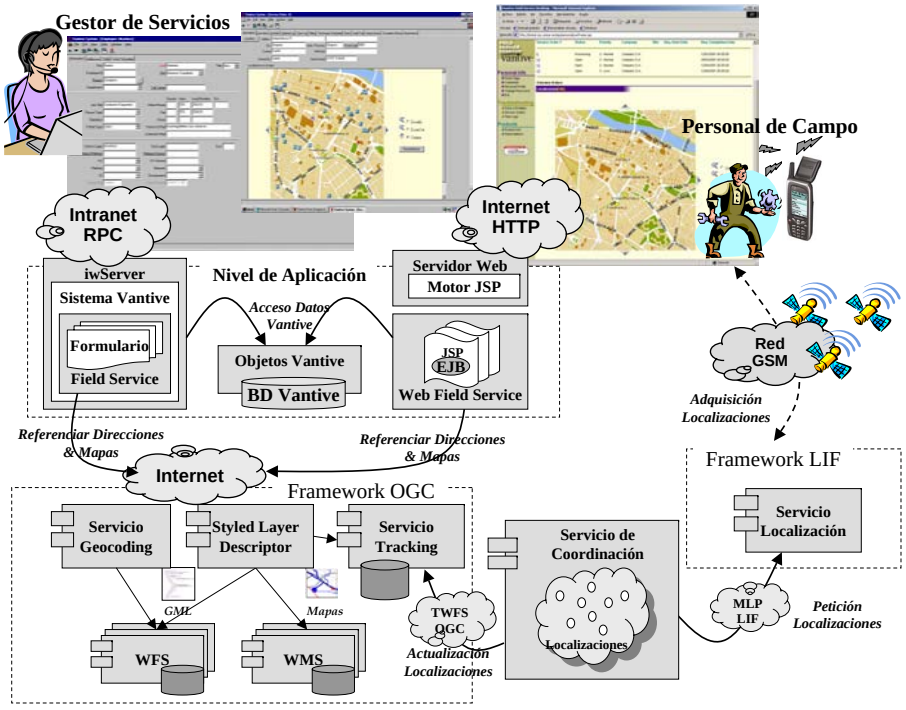

Figura 2.10: Integración de la funcionalidad SBL en Vantive

Por una parte, la herramienta *Field Service*, utilizada por los gestores de servicios para la creación, modificación y seguimiento de las órdenes de servicio, ha sido modificada para poder visualizar mapas con la localización exacta de la dirección donde deben realizarse servicios o con la con la localización de los técnicos, proveedores o almacenes de la empresa próximos.

[1] *Vantive* es propiedad de Oracle desde el 2005. En la actualidad, Oracle no da soporte del producto, aunque sigue siendo utilizado por un buen número de empresas

Por otra, la herramienta *Web Field Service*, utilizada por los técnicos de campo para acceder al sistema CRM a través de Internet, ha sido modificada para poder obtener de *Vantive* las direcciones de las órdenes de servicio asignadas al técnico conectado al sistema y visualizar un mapa con estas direcciones, su orden de ejecución y la localización de recursos de la empresa próximos.

En ambos casos de uso, las localizaciones son recibidas periódicamente por el servicio de localización configurado, e integradas en el *framework* de servicios SIG por medio de la acción del servicio de coordinación que actualiza las posiciones almacenadas en su servicio *tracking*, según el patrón de integración descrito anteriormente.

Capítulo 3

Un modelo formal del sistema

3.1. Un modelo abstracto para Linda

Parte de los resultados recogidos en este capítulo aparecieron reflejados en las publicaciones [ÁBM+03a], [MÁBR04] y [MÁBR07a].

Para la formalización del modelo básico de Linda hemos seguido el formalismo matemático utilizado por Viroli y Ricci en [VR02], que se basa en los sistemas de transición de estados de Plotkin [Plo81]. Aunque existen otras aproximaciones en la literatura, algunas de ellas inspiradas en el álgebra de procesos [BGZ98, BGZ00, BZ08], hemos optado por este formalismo por su simplicidad y porque, para nuestro problema concreto, estamos más interesados en las clases de datos que vamos a poder manejar en el medio de coordinación, mediante un mecanismo de correspondencia o *pattern-matching*, que en el comportamiento de los componentes activos (o procesos). Así, modelos formales como los de [BGZ98] abstraen en exceso las representaciones concretas de las tuplas en el espacio compartido, por lo que no son adecuados para representar las transformaciones complejas de estructuras de datos que hemos descrito en el apartado anterior. Dichas transformaciones deberían ir siendo anotadas en las comunicaciones del álgebra de procesos, lo que sería ciertamente farragoso (incluso desde el punto de vista de las notaciones). Una situación similar, en el marco de la orquestación de servicios Web, queda reflejada en [LZH08]. En dicho trabajo, un formalismo abstracto denominado *t-calculus* [LZH07] tuvo que ser concretado en otro llamado TCOS [LZPH07] en el que los datos pueden ser modelados más explícitamente. En nuestro caso, el modelo de Viroli y Ricci [VR02] admite una *navegación* más sencilla entre modelos más abstractos (que no fijan ningún modo de representación de tuplas) y otros más concretos (en los que ciertos aspectos de la representación son explicitados). Como veremos, esa característica es esencial para nuestra aproximación.

El *espacio de tuplas*, que es el medio de coordinación basado en *tuplas* definido en Linda, se representa por medio de un sistema de transición (que abstrae a una componente software) que interactúa con las entidades coordinadas recibiendo *eventos de*

entrada y enviando *eventos de salida*.

Como hemos visto en los capítulos anteriores, la memoria *asociativa* en la que Linda se basa hace que una misma tupla pueda aparecer repetidas ocasiones en un mismo espacio de tuplas (es decir, en un espacio de tuplas pueden aparecer tuplas *distintas* con la propiedad de que son de la misma longitud y tienen las mismas componentes en el mismo orden; tuplas con esas características son *indistinguibles* desde el punto de vista de los mecanismos de consulta e interacción de Linda). Por ello, el formalismo matemático de la *Teoría de Conjuntos* no es adecuado para especificar Linda (pues, en un conjunto, cada elemento puede aparecer, a lo más, una única vez). Así, es conveniente trabajar con *multiconjuntos*: colecciones de elementos en los que un elemento puede aparecer con cardinalidad superior a 1. Dado un dominio de elementos X, denotaremos $\overline{X}$ a la familia de todos los multiconjuntos formados a partir de elementos de X y con $\overline{x}$ a cualquier multiconjunto de dicha familia. Para hacer explícito que un elemento x aparece (al menos una vez) en un multiconjunto escribiremos dicho multiconjunto como $x\ \overline{x}$. El predicado de pertenencia de un elemento x a un multiconjunto $\overline{x}$ será indicado con la notación habitual para conjuntos: $x \quad \overline{x}$. Asimismo, eliminar un elemento de un multiconjunto será representado, como es habitual, por: $\overline{x}\ x$ (pero, atención, eso no significa que x ya no pertenezca al multiconjunto resultante, como sucede con la misma notación en conjuntos; se elimina una sola ocurrencia del elemento x, no necesariamente todas).

Con cierta frecuencia, un dominio de datos D es ampliado con un elemento especial que no pertenecía previamente a D, denotado usualmente por $\quad_D$ (o simplemente $\quad$, si no ha lugar a confusión). El conjunto así resultante lo denotaremos $D_\perp = D \qquad_D$. Cuando un símbolo es escrito con un acento circunflejo, como en $\widehat{d}$, queremos denotar que el elemento puede estar en $D_\perp$ (es decir, que el elemento puede estar en D o ser igual a $\quad_D$).

Por último, para expresar la semántica de predicados utilizamos la siguiente notación:

$$\frac{pred_1 \ldots pred_n}{pred}$$

que hay que interpretar del siguiente modo: de la *conjunción* de todas las propiedades $pred_1 \ldots pred_n$ (que aparecen en la notación simplemente concatenadas, sin comas adicionales) se obtiene como consecuencia *pred*. Esa notación también admite una interpretación en términos de procesos: de las entradas $pred_1 \ldots pred_n$ se obtiene la salida *pred*. O desde una perspectiva más dinámica también se puede comprender que los eventos $pred_1 \ldots pred_n$ disparan el evento *pred*.

Con estos convenios notacionales podemos describir los principales elementos de nuestro marco abstracto de trabajo que son:

- un **conjunto de tuplas** T, cuyos elementos denotamos con la variable t. Si denotamos $\overline{T}$ al conjunto de multiconjuntos sobre T, entonces el contenido del espacio de tuplas en un instante dado es un multiconjunto de tuplas $\overline{t}$ perteneciente a $\overline{T}$;

- un **conjunto de tuplas plantilla** $Templ$ cuyos elementos denotaremos con la variable *templ*;
- un **predicado de *correspondencia***, o *emparejamiento*, entre una plantilla y una tupla $mtc(templ, t)$;
- y un **predicado de selección** $\mu(templ, \overline{t}, \widehat{t})$, entre una plantilla, un multiconjunto de tuplas y una tupla $\widehat{t}$. El predicado μ debe cumplir las siguientes *reglas de correspondencia*:

$$\frac{mtc(templ, t)}{\mu(templ, t\ \overline{t}, t)} \qquad \frac{\nexists\ t \quad \overline{t}\ :\ mtc(templ, t)}{\mu(templ, \overline{t}, \quad _T)} \tag{3.1}$$

  La tupla $\widehat{t}$ está definida sobre $T_\perp = T \quad _T$ y representa a cualquier tupla perteneciente a $\overline{t}$ que se corresponda con la plantilla *templ*, o bien, toma el valor $_T$ (o simplemente por simplicidad) para denotar una especie de valor excepción cuando no existe ninguna tupla en $\overline{t}$ que se corresponda con la plantilla *templ*.

  La primera regla de correspondencia modeliza la relación entre los predicados *mtc* y μ, recogiendo el hecho de que cualquier tupla perteneciente a $\overline{t}$ que se corresponda con la plantilla *templ* es válida para que se satisfaga el predicado μ. La segunda regla modeliza la situación de la no existencia de una tupla en el multiconjunto $\overline{t}$ que se corresponda con la plantilla, asumiendo $\widehat{t}$ el valor .

Representaremos al estado del espacio de tuplas, en un instante dado, mediante el par $\overline{t}, \overline{w}$, donde $\overline{t}$ es un multiconjunto de tuplas y $\overline{w}$ es un multiconjunto de *peticiones pendientes*. Una petición pendiente será un evento de entrada *ie* perteneciente a $IE \subseteq W$, donde W es el conjunto de primitivas tales como la operación de lectura rd, la operación de lectura y eliminación in, o la operación de escritura out, las cuales posiblemente están esperando a ser atendidas por el espacio de tuplas.

Los eventos de salida *oe* toman valores de OE, donde OE es un conjunto de mensajes de notificación, con la sintaxis $\underline{o}v$ representando un mensaje v para la entidad o.

La semántica del espacio de tuplas quedará definida por el par S, E , donde $S \subseteq W \times \overline{T}$ es un predicado de satisfacción para las peticiones y E es una función de evaluación.

La interpretación de la pertenencia, $w, \overline{t} \quad S$, es que que w es *satisfacible* en $\overline{t}$; es decir, que la petición w puede ser atendida (o evaluada) bajo el contenido del espacio de tuplas en curso $\overline{t}$. También escribiremos, en este caso, $S(w, \overline{t})$, en forma de predicado.

La función de evaluación E se define como una aplicación:

$$E : W \quad 2^{(\overline{T} \times \overline{W}) \times OE \times (\overline{T} \times \overline{W})}$$

Así, $\overline{t}, \overline{w}, \widehat{oe}, \overline{t}', \overline{w}' \quad E(w)$ debe ser interpretado como que la evaluación de la petición pendiente w provoca que el espacio de tuplas pase del estado $\overline{t}, \overline{w}$ al estado $\overline{t}', \overline{w}'$ y produce $\widehat{oe}$ (es decir, el evento de salida *oe* o nada cuando su valor sea).

3.1.1. Dinámica del espacio de tuplas

Tras precisar el marco notacional y terminológico, procedemos en este apartado a definir el espacio de transición que corresponde al modelo Linda. En otras palabras, vamos a definir el predicado S y la función de evaluación E. Para ello, en primer lugar, introducimos el comportamiento dinámico del sistema de transición (aquí S y E continúan siendo genéricos).

La dinámica del espacio de tuplas se define mediante el siguiente sistema de transiciones etiquetado:

$$\begin{array}{rcl} {\scriptstyle I} & \subseteq & (\overline{T} \times \overline{W}) \times IE \times (\overline{T} \times \overline{W}) \\ {\scriptstyle O} & \subseteq & (\overline{T} \times \overline{W}) \times OE \times (\overline{T} \times \overline{W}) \\ {\scriptstyle I} & \subseteq & (\overline{T} \times \overline{W}) \times IE \times \overline{OE} \times (\overline{T} \times \overline{W}) \end{array}$$

Este sistema distingue tres transiciones: la primera, para describir la acción de los eventos de entrada; la segunda, para los eventos de salida; y la tercera, para describir al espacio de coordinación como un componente reactivo, es decir, que las transiciones de entrada y salida no son eventos disjuntos, sino que a la recepción de un evento de entrada le corresponde el envío de uno, o varios, eventos de salida.

- La transición $\bar{t}, \overline{w} \stackrel{ie}{-}_I \bar{t}', \overline{w}'$ representa el paso del estado $\bar{t}, \overline{w}$ al estado $\bar{t}', \overline{w}'$ después de recibir el evento de entrada ie. Su semántica queda definida por la regla:

$$\frac{\nexists\, w \quad \overline{w} : S(w, \bar{t})}{\bar{t}, \overline{w} \stackrel{ie}{-}_I \bar{t}, ie\, \overline{w}}$$

  La regla dice que que si al llegar un evento de entrada ie ninguna petición pendiente es satisfacible, entonces ie se añade al conjunto de peticiones pendientes.

- La transición $\bar{t}, \overline{w} \stackrel{oe}{-}_O \bar{t}', \overline{w}'$ representa el paso del estado $\bar{t}, \overline{w}$ al estado $\bar{t}', \overline{w}'$ tras el envío del evento de salida oe. Se interpreta mediante la regla:

$$\frac{S(w, \bar{t}) \quad \bar{t}, w\, \overline{w}, oe, \bar{t}', \overline{w}' \quad E(w)}{\bar{t}, w\, \overline{w} \stackrel{oe}{-}_O \bar{t}', \overline{w}}$$

  Esta regla modeliza el hecho de que si una petición pendiente w es satisfacible bajo el contenido de $\bar{t}$, entonces se produce un cambio de estado del espacio de tuplas, se envía un evento de salida oe y se elimina w del conjunto de peticiones pendientes.

- Por último, la transición $\bar{t}, \overline{w} \stackrel{ie \triangleright [\overline{oe}]}{\longrightarrow} \bar{t}', \overline{w}'$ representa el paso del estado $\bar{t}, \overline{w}$ al estado $\bar{t}', \overline{w}'$ cuando el espacio de tuplas recibe el evento de entrada ie y envía el correspondiente multiconjunto de eventos de salida $\overline{oe}$. La semántica de esta transición queda definida mediante la siguiente regla:

$$\frac{\begin{array}{c}\overline{t}_0, \overline{w}_0 \ \stackrel{ie}{-}_I \ \overline{t}_1, \overline{w}_1 \\ \overline{t}_i, \overline{w}_i \ \stackrel{oe_i}{-}_O \ \overline{t}_{i+1}, \overline{w}_{i+1} \quad 0 < i < n \\ \nexists\, w \quad \overline{w}_n : S(w, \overline{t}_n)\end{array}}{\overline{t}_0, \overline{w}_0 \ \xrightarrow{ie \triangleright [oe_1|...|oe_n]} \ \overline{t}_n, ie\ \overline{w}_n}$$

Esta regla recoge la interpretación del espacio de coordinación como un componente que acepta inicialmente un evento de entrada y que envía todos los eventos de salida correspondientes a las peticiones pendientes que son satisfacibles (sin distinguir en qué orden son procesadas).

Si nos centramos en el modelo Linda, considerando como eventos de entrada las siguientes cinco primitivas: $rdp(templ)^o$, $in(templ)^o$, $rdp(templ)^o$, $inp(templ)^o$ y $out(t)^o$ (donde la etiqueta o se refiere al proceso que envía la petición) entonces podremos describir el comportamiento de Linda particularizando el predicado de satisfacibilidad S y la función de evaluación E a estas cinco primitivas.

En este caso, el predicado S se define como la mínima relación que satisface las siguientes reglas:

$$S(rdp(templ)^o, \overline{t})$$

$$S(inp(templ)^o, \overline{t})$$

$$\frac{mtc(templ, t)}{S(rd(templ)^o, t\ \overline{t})}$$

$$\frac{mtc(templ, t)}{S(in(templ)^o, t\ \overline{t})}$$

$$S(out(t)^o, \overline{t})$$

En resumen, estas reglas expresan que las peticiones pendientes correspondientes a los eventos de entrada *inp*, *rdp* y *out* se satisfacen siempre, mientras que *in* y *rd* se atienden solamente cuando haya, al menos, una tupla en el espacio que se corresponda con la plantilla (lo que hace de *in* y *rd* operaciones *bloqueantes*, respecto a la dinámica del sistema de transición).

La función de evaluación E, que especifica la semántica de las operaciones, se define mediante una regla para cada petición pendiente correspondiente a cada una de las operaciones permitidas:

- La operación *rdp* elige una tupla del espacio que se corresponde con la plantilla *templ* y la envía al proceso invocante o. En caso de que no exista tal tupla, se envía el mensaje $\underline{o}$.

$$\frac{\mu(templ, \overline{t}, \widehat{t})}{\overline{t}, \overline{w}, \underline{o}\widehat{t}, \overline{t}, \overline{w} \qquad E(rdp(templ)^o)}$$

- La operación *inp* se comporta de forma análoga a la operación anterior, pero además la tupla es eliminada del espacio.

$$\frac{\mu(templ, \overline{t}, \widehat{t})}{\overline{t}, \overline{w}, \underline{o}\widehat{t}, \overline{t}\ \ \widehat{t}, \overline{w} \qquad E(inp(templ)^o)}$$

- La operación *rd* es similar a la operación *rdp*, pero siempre devuelve una tupla válida porque no se evalúa hasta que no haya una tupla en el espacio que se corresponda con la plantilla.

$$\frac{mtc(templ, t)}{t\ \overline{t}, \overline{w}, \underline{o}t, t\ \overline{t}, \overline{w} \qquad E(rd(templ)^o)}$$

- La operación *in* tiene respecto a *inp* un comportamiento análogo al que tiene la operación *rd* con respecto a la operación *rdp*.

$$\frac{mtc(templ, t)}{t\ \overline{t}, \overline{w}, \underline{o}t, \overline{t}, \overline{w} \qquad E(in(templ)^o)}$$

- Finalmente, la operación *out* simplemente añade una tupla al espacio, sin enviar ningún mensaje a ningún proceso.

$$\overline{t}, \overline{w}, \quad , t\ \overline{t}, \overline{w} \qquad E(out(t)^o)$$

Es importante entender, en la anterior descripción, que la dinámica del modelo queda determinada al fijar los predicados *mtc* y μ. Así, puede haber tantos modelos Linda como definiciones se den a esos dos predicados, *siempre que se satisfagan las condiciones semánticas* definidas en (3.1). En el resto de este capítulo nos concentraremos en estos aspectos *estáticos* del modelo. Las cuestiones dinámicas volverán a ser importantes en el capítulo 5, donde el modelo Linda será generalizado.

3.2. El modelo *core*. Un modelo inspirado por *JavaSpaces*

Una vez presentado el marco abstracto para el modelo Linda, en este apartado daremos una definición de las tuplas y las plantillas como secuencias finitas y ordenadas de datos, cada uno sobre su propio dominio. Además, ahora las tuplas plantilla admitirán variables libres. Esta definición pretende ser una formalización, bastante simplificada, de *JavaSpaces*. Las tuplas representarán a los objetos Java de clases que implementen la interfaz Entry (ver 1.2, en el capítulo 1). Después, definiremos los predicados de correspondencia y de selección que formalizan, también de manera simplificada, el procedimiento de correspondencia de objetos de *JavaSpaces*. Algunas de estas definiciones

han sido inspiradas por el artículo [BGLZ04]. Denominaremos a este modelo formal **modelo core**.

Sea $A = (A_1, \ldots, A_m)$ una lista de atributos con dominios $D_i = dom(A_i)$, para $i = 1, \ldots, m$. Utilizamos la notación $D = dom(A) = \bigcup_{i=1}^{m} dom(A_i) = \bigcup_{i=1}^{m} D_i$ para indicar el conjunto de todos los posibles valores que puede tomar el conjunto de atributos.

Definición 3.1 (**Tupla_core**)**.** Una tupla_core $\mathbf{t_{core}}$ sobre A es una aplicación parcial e inyectiva:

$$\mathbf{t_{core}} : \quad X \subseteq A - \quad dom(X) \tag{3.2}$$

tal que:

$$A_i \quad X \quad t_{core}(A_i) = d_i \text{ con } d_i \quad D_i.$$

Por tanto, una *tupla_core* es una secuencia finita y ordenada de datos simples de la forma $t_{core} = (d_{i_1}, \ldots, d_{i_n})$ donde cada $d_{i_j} \quad D_{i_j}$, con $j = 1, \ldots, n$. Por comodidad, escribiremos las tuplas con índices simples: $(d_1, \ldots, d_n)$.

Definición 3.2 (**Plantilla_core**)**.** Una plantilla_core $\mathbf{templ_{core}}$ sobre A es una función parcial e inyectiva:

$$\mathbf{templ_{core}} : \quad X \subseteq A - \quad dom_?(X) \tag{3.3}$$

tal que:

$$A_i \quad X \quad templ_{core}(A_i) = td_i \text{ con } td_i \quad dom_?(A_i) \text{ donde } dom_?(A_i) = dom(A_i) \quad ?a_i$$

El símbolo $?x_i$ representa el nombre de una variable libre asociada al atributo A_i. Su papel en la plantilla es equivalente al del valor `null` en *JavaSpaces*, es decir, significa que la tupla que se empareje con la plantilla puede tener cualquier valor en ese campo. Un vez realizado el emparejamiento, la variable x_i almacenará una copia del valor que tiene la tupla en el campo A_i.

Por tanto, una *plantilla_core* es una secuencia ordenada y finita de datos o variables libres de la forma $templ_{core} = (td_1, \ldots, td_n)$ con $td_i \quad dom_?(A_i)$.

T_{core} será el conjunto de todas las posibles tuplas definidas sobre A, y $Templ_{core}$ será el conjunto de todas las posibles plantillas en el modelo Linda *core*. Por definición, es obvio que $T_{core} \subseteq Templ_{core}$, i.e. en particular una tupla es una plantilla sin variables. De aquí en adelante, únicamente utilizaremos el adjetivo *core* cuando sea necesario para distinguir a qué modelo nos estamos refiriendo.

Ejemplo 3.3. ***Algunos casos de tuplas y plantillas*** **core**

Tupla_core

La siguiente tupla representa un vuelo que parte de Madrid a las 10:30 y llega a Londres a las 12:30.

(vuelo, Madrid, Londres, 10:30, 12:30)

Plantilla_core sin variables

La misma tupla anterior puede ser también interpretada como una plantilla sin variables. *Bajo esa perspectiva, esa plantilla permitiría comprobar la existencia de un vuelo de Madrid a Londres que parta a las 10:30 y llegue a las 12:30.*

Plantilla_core con variables

Con esta plantilla podemos comprobar si existe algún vuelo entre Madrid y Londres. En caso de existir un vuelo, la variable x contendrá la hora de salida y la variable y la hora de llegada a Londres.

(vuelo, Madrid, Londres, ?x, ?y)

Definición 3.4 (**Predicado de correspondencia_core**)**.** Sea $t_{core} = (d_1, \ldots, d_n)$ T_{core} una tupla, y $templ_{core} = (td_1, \ldots, td_{n'})$ $Templ_{core}$ una plantilla. Decimos que t_{core} se **corresponde o empareja** con $templ_{core}$, y lo denotamos con el predicado $\mathbf{mtc_{core}}(\mathbf{templ_{core}}, \mathbf{t_{core}})$, si se cumplen las siguientes condiciones:

1. La tupla t_{core} y la plantilla $templ_{core}$ tienen la misma aridad, es decir $n = n'$, y están definidas sobre el mismo subconjunto de atributos de A.
2. Cada campo de $templ_{core}$ cuyo contenido sea un dato, y no una variable, es igual que el correspondiente campo de t_{core}, es decir $td_i = d_i$ o $td_i = ?x_i, \quad i\ 1 \leq i \leq n$.

El predicado mtc_{core} recibe el nombre de **predicado de correspondencia_core**.

Definición 3.5 (**Predicado de elección_core**)**.** Dado un predicado de correspondencia $mtc_{core}(templ_{core}, t_{core})$ definido sobre $Templ_{core} \times T_{core}$, llamamos **predicado de elección_core**, denotado por $\mu_{\mathbf{core}}(\mathbf{templ_{core}}, \overline{\mathbf{t}}_{\mathbf{core}}, \widehat{\mathbf{t}}_{\mathbf{core}})$, a *cualquier* predicado que satisfaga las reglas de correspondencia de Linda definidas en (3.1).

$$\frac{mtc_{core}(templ_{core}, t_{core})}{\mu_{core}(templ_{core}, t_{core}\ \overline{t}_{core}, t_{core})} \qquad \frac{\nexists\ t \quad \overline{t}_{core}\ :\ mtc_{core}(templ_{core}, t_{core})}{\mu_{core}(templ_{core}, \overline{t}_{core}, \quad)}$$

Es decir, en caso de existir alguna, $\widehat{t}_{core}$ es una tupla cualquiera del multiconjunto $\overline{t}_{core}$ que se corresponde con la plantilla $templ_{core}$, o bien es un valor nulo (indicado por el símbolo) si $\overline{t}_{core}$ no contiene dicha tupla.

Las definiciones 3.4 y 3.5 formalizan el procedimiento de correspondencia de *JavaSpaces*. Obsérvese que, en realidad, lo único que hemos definido con precisión es el predicado de emparejamiento, puesto que es el descrito (informalmente) en la definición de *JavaSpaces*. Como predicado de selección elegimos cualquiera que, en relación con el de emparejamiento, respete la semántica de Linda. Esta decisión significa que estamos admitiendo, sin demostración, que *JavaSpaces* satisface realmente la semántica de Linda, como los implementadores del producto afirman. Esta suposición actúa como *axioma* en nuestro acercamiento al problema de determinar si también el servicio de coordinación WCS respeta la semántica de Linda.

3.3. El modelo *estructurado*. Una extensión estructurada del modelo *core*

En este apartado, con el fin de proporcionar un modelo formal más cercano al servicio de coordinación descrito en capítulo anterior, trabajaremos con una versión donde las tuplas y las plantillas serán secuencias de pares *atributo/valor*.

Análogamente al modelo *core*, partimos de un conjunto de atributos $A = A_1, \ldots, A_m$ con dominios $D_i = dom(A_i)$, $i = 1, \ldots, m$ y $D = dom(A) = \bigcup_{i=1}^{m} dom(A_i) = \bigcup_{i=1}^{m} D_i$ el conjunto de todos los posibles valores que pueden tomar los atributos.

Además, ahora exigiremos que $A \setminus D = \varnothing$ para mantener la distinción entre atributos y valores.

Definición 3.6 (**Tupla_estructurada**)**.** Una tupla_estructurada $\mathbf{t_{str}}$ sobre A es una aplicación parcial e inyectiva:

$$\mathbf{t_{str}} : \quad X \subseteq A - \quad X \times dom(X) \tag{3.4}$$

tal que:

$$a_i \quad X\ t_{str}(a_i) = (a_i, v_i) \text{ con } v_i \quad dom(a_i) \text{ y } a_i = a_j \text{ si } i = j, \quad i, j = 1, \ldots, n.$$

Por tanto, una tupla_estructurada es un conjunto finito de pares *atributo/valor* de la forma $t_{str} = ((a_1, v_1) \ldots (a_n, v_n))$, sin atributos repetidos.

Definición 3.7 (**Plantilla_estructurada**)**.** Una plantilla_estructurada $\mathbf{templ_{str}}$ sobre A es una aplicación parcial e inyectiva:

$$\mathbf{templ_{str}} : \quad X \subseteq A - \quad X \times dom_?(X) \tag{3.5}$$

tal que: $A_i \quad X\ templ_{str}(A_i) = (A_i, tv_i)$ con $tv_i \quad dom_?(A_i)$, donde $dom_?(A_i) = dom(A_i) \quad ?x_i$. El símbolo $?x_i$ tiene el mismo papel que en el modelo *core*, representa un comodín en la plantilla y almacenará una copia del valor en el campo A_i contenido en la tupla que se corresponda con la plantilla.

Ejemplo 3.8. ***Ejemplos de tuplas y plantillas estructuradas***

Tupla_estructurada

Una tupla estructurada contiene, en cada campo, el nombre del atributo y su valor asociado. Se puede interpretar como un mensaje XML plano. Esta es una tupla estructurada que representa el vuelo de Madrid a Londres que sale a las 10:30 y llega a las 12:30:

((tipo vuelo) (origen Madrid) (destino Londres) (salida 10:30) (llegada 12:30))

Plantilla_estructurada sin variables

Análogamente a lo que se comentó para el modelo core*, una tupla como la anterior es también considerada una plantilla estructurada sin variables, con todos sus valores constantes.*

Plantilla estructurada con variables

Igual que en el modelo core*, las variables se introducen con el símbolo ? y tienen el mismo significado. En este ejemplo recogen la hora de salida y llegada del vuelo.*

((tipo vuelo) (origen Madrid) (destino Londres) (salida ?x) (llegada ?y))

Definición 3.9 (Predicado de correspondencia_estructurado). Sea $t_{str} = ((a_1, v_1) \dots (a_n, v_n))$ una tupla sobre $X \subseteq A$ y sea $templ_{str} = ((ta_1, tv_1) \dots (ta_{n'}, tv_{n'}))$ una plantilla sobre $X' \subseteq A$.

Se dice que la tupla t_{str} se **corresponde o empareja** con $templ_{str}$, y se indica mediante el predicado $\mathbf{mtc_{str}}(\mathbf{templ_{str}}, \mathbf{t_{str}})$, si se cumplen las siguientes condiciones:

1. Tienen la misma aridad, es decir $n = n'$.
2. Están definidas sobre el mismo conjunto de atributos, es decir, $ta_i = a_i$, $1 \leq i \leq n$.
3. En aquellos atributos en los que la plantilla no tenga una variable libre, la tupla y la plantilla tienen el mismo valor. Es decir, $tv_i = v_i$, o $tv_i = ?x_i$, $1 \leq i \leq n$.

Obsérvese que, dado que en este modelo cada campo de la tupla y de la plantilla contienen, además del valor, el nombre del atributo, cada campo queda determinado unívocamente dentro de la tupla por un nombre de atributo (pues, como ya indicamos, no aparecen atributos repetidos ni en tuplas ni en plantillas). Así, podríamos haber desvinculado el proceso de emparejamiento del orden en el que aparecen los campos (o, dicho de otro modo, podríamos haber considerado las tuplas como *conjuntos* de pares *atributo/valor*, en lugar de listas). Sin embargo, en la definición de correspondencia anterior hemos mantenido la condición de que los atributos deben aparecer en la misma posición, porque como explicamos en el capítulo anterior, esta propiedad es exigida en el proceso de emparejamiento de las tuplas XML en el servicio de coordinación WCS.

En este instante, es natural preguntarse por la relación existente entre el modelo *estructurado* y el modelo *core*. Dada una tupla/plantilla_estructurada podríamos obtener una tupla/plantilla_core no considerando la distinción entre atributos y valores (esto es, "aplanando" la estructura y obteniendo una tupla de longitud $2n$ en la que los atributos también son considerados datos). Sin embargo, esta operación no se corresponde con ningún proceso realizado en el servicio de coordinación WCS (aparte de que tendría la característica "patológica" de que los datos en posiciones impares no podrían ser instanciados). Una segunda idea consiste en "olvidar" los atributos y obtener una tupla/plantilla_core, en la que solo los valores aparecen. La tercera posibilidad sería

quedarnos únicamente con los atributos y obtener una tupla que podríamos denominar *tupla_esquema*. Las siguientes definiciones recogen en el marco formal estas dos últimas ideas.

Definición 3.10. La **tupla_esquema** de una tupla_estructurada, o de una plantilla_estructurada, es una tupla_core t_{core} en A definida por la aplicación:

$$\textbf{esquema}: \quad T_{str} \quad Templ_{str} - \quad T_{core_A} \tag{3.6}$$

tal que:

$$esquema(((a_1, v_1) \dots (a_n, v_n)) = (a_1, \dots, a_n))$$

Es decir, dada una tupla_estructurada t_{str}, o una plantilla_estructurada $templ_{str}$, la aplicación *esquema* devuelve una tupla_core t_{core} que contiene únicamente los nombres de los atributos de la tupla_estructurada. Dicha tupla tiene como dominio de datos el conjunto de atributos A, por lo que hemos denotado T_{core_A} al conjunto de todas estas posibles tuplas. Esta tupla t_{core} representa la estructura de la tupla_estructurada, y permitirá descomponer el espacio de tuplas_estructuradas en particiones disjuntas. Cada partición contendrá todas aquellas tuplas_estructuradas que compartan la estructura común. La idea que subyace a esta definición, por supuesto, es la de *esquema XML*, pues en los casos en los que son empleadas la tuplas XML en el servicio de coordinación (véase el capítulo anterior), la tupla esquema contiene la misma información que el correspondiente esquema XML.

Definición 3.11. El $\overline{\textbf{esquema}}$ de un multiconjunto $\bar{t}_{str}$ es otro multiconjunto de tuplas_core en A definido por la aplicación:

$$\overline{\textbf{esquema}}: \quad \overline{T}_{str} - \quad \overline{T}_{core_A} \tag{3.7}$$

tal que:

$$\overline{esquema}(\bar{t}_{str}) = \quad esquema(t_{str}) \quad t_{str} \quad \bar{t}_{str}$$

Definición 3.12. La **tupla_valor** de una tupla_estructurada es una tupla_core en D definida por la aplicación:

$$\textbf{valor}: \quad T_{str} - \quad T_{core_D} \tag{3.8}$$

tal que:

$$valor(((a_1, v_1) \dots (a_n, v_n))) = (v_1, \dots, v_n)$$

Por tanto, dada una tupla_estructurada t_{str}, la aplicación *valor* devuelve una tupla_core t_{core} que contiene únicamente los valores asociados a cada atributo.

Definición 3.13. La **plantilla_valor** de una plantilla_estructurada es una plantilla_core en $dom_?(D)$ definida por la aplicación:

$$\textbf{valor}: \quad Templ_{str} - \quad T_{core_{dom_?(D)}} \tag{3.9}$$

tal que:

$$valor((ta_1, tv_1) \dots (ta_n, tv_n)) = (tv_1, \dots, tv_n)$$

Es decir, dada una plantilla_estructurada $templ_{str}$, la aplicación *valor* devuelve una plantilla_core $templ_{core}$ que contiene únicamente los valores o variables libres asociados a cada atributo.

El siguiente teorema muestra la relación existente entre el predicado de correspondencia en el modelo *estructurado* y el predicado de correspondencia en el modelo *core*.

Teorema 3.14. *Se cumple la siguiente relación entre* mtc_{core} *y* mtc_{str}*:*

$$mtc_{str}(templ_{str}, t_{str}) = mtc_{core}(esquema(templ_{str}), esquema(t_{str})) \\ mtc_{core}(valor(templ_{str}), valor(t_{str}))$$

Demostración. Sea $t_{str} = ((a_1, v_1) \dots (a_n, v_n))$ T_{str} una tupla_estructurada, y sea $templ_{str} = ((ta_1, tv_1) \dots (ta_{n'}, tv_{n'}))$ $Templ_{str}$ una plantilla_estructurada.

Por la definición 3.9 se cumple que:

1. tienen la misma aridad, $n = n'$.
2. están definidos sobre el mismo conjunto de atributos, $ta_i = a_i$, $1 \leq i \leq n$.
3. tienen el mismo valor asociado a cada atributo o en el campo correspondiente de la plantilla hay una variable libre: $tv_i = v_i$ or $tvi = ?x_i$, $1 \leq i \leq n$.

Por las condiciones 1 y 2, y la definición 3.4

$$mtc_{core}(esquema(templ_{str}), esquema(t_{str}))$$

y por las condiciones 1 y 3, y la definición 3.4

$$mtc_{core}(valor(templ_{str}), valor(t_{str}))$$

■

A partir de las definiciones 3.11 y 3.12, podemos concluir que todo multiconjunto de tuplas_estructuradas tendrá asociados dos multiconjuntos de tuplas_core. Uno formado por tuplas_esquema y otro por tuplas_valor.

En la siguiente definición presentamos, en nuestro marco de trabajo, el concepto de *canal*. Este concepto apareció en la implementación de WCS como una herramienta para simular el comportamiento de Linda, a través de la recuperación de múltiples tuplas (con el mismo esquema) mediante *JavaSpaces*. Aquí damos una interpretación de los canales como herramienta para demostrar que la semántica de Linda es preservada cuando el servicio WCS es representado en nuestro modelo formal.

Definición 3.15. Dados un multiconjunto de tuplas_estructuradas $\overline{t}_{str}$ y una plantilla_estructurada $templ_{str}$, definimos el $\overline{\mathbf{canal}}_{str}$ de $templ_{str}$ en el multiconjunto $\overline{t}_{str}$ como la aplicación:

$$\overline{\mathbf{canal}}_{\mathbf{str}} : \quad Templ_{str} \times \overline{T}_{str} - \quad \overline{T}_{core_D} \tag{3.10}$$

tal que:

$$\overline{canal}_{str}(templ_{str}, \bar{t}_{str}) = \\ valor(t_{str}) \quad \overline{T}_{core_D}\ mtc_{core}(esquema(templ_{str}), esquema(t_{str}))$$

A partir de la definición anterior, podemos concluir que el *canal* de una plantilla_estructurada, en un multiconjunto de tuplas, es el submulticonjunto de tuplas_valor que comparten la misma estructura de atributos que la plantilla.

Llamaremos indistintamente canal al multiconjunto de tuplas_estructuradas o a su correspondiente multiconjunto de tuplas_valor en el modelo *core* (esta segunda noción fue la adoptada en nuestro artículo [MÁBR07a]), puesto que, conocido el esquema de la plantilla, un multiconjunto puede obtenerse de forma biunívoca a partir del otro.

Ahora estamos en condiciones de dar el paso inverso, es decir, pasar del modelo *core* al modelo *estructurado*. Se trata de construir un tupla_estructurada a partir de una tupla_esquema y una tupla_valor dadas. La siguiente definición muestra cómo se construye dicha tupla.

Definición 3.16. La operación **construir_tupla**, que permite crear una tupla_estructurada, es una aplicación *parcial*:

$$\mathbf{construir\_tupla_{str}}: \quad T_{core_A} \times T_{core_D} - \quad T_{str} \tag{3.11}$$

tal que:

Dadas una tupla_esquema $a = (a_1, \ldots, a_n)$ y una tupla_valor $v = (v_1, \ldots, v_m)$ tal que $n = m$ y $v_i \quad D_i = dom(A_i) \quad i = 1, \ldots, n$, entonces:

$$construir\_tupla_{str}(a, v) = ((a_1, v_1) \ldots (a_n, v_n))$$

Si a o v son la tupla nula ( ) entonces la imagen de la aplicación es también una tupla nula ( ).

Por tanto, a partir de una tupla_esquema y una tupla_valor "compatibles", mediante la operación *construir_tupla*, podemos crear una tupla_estructurada compuesta por una secuencia de pares *atributo/valor*. La aplicación es parcial porque no siempre se va a poder construir una tupla_estructurada a partir de dos tuplas_core cualesquiera. La "compatibilidad" significa que los valores de la tupla_valor tienen que pertenecer a los correspondientes dominios de los atributos de la tupla_esquema respectivos.

Definición 3.17 (Predicado de elección_estructurado). Definimos en el modelo *estructurado* el predicado μ_{str}, como el predicado $\mu_{str}(templ_{str}, \bar{t}_{str}, \widehat{t}_{str})$, donde $templ_{str}$ es una plantilla_estructurada, $\bar{t}_{str}$ es un multiconjunto de tuplas_estructuradas y la tupla $\widehat{t}_{str}$ se define como:

$$\widehat{t}_{str} = construir\_tupla_{str}(\widehat{t.esquema}, \widehat{t.valor})$$

donde la tupla $t.\widehat{esquema}$ satisface el predicado:

$$\mu_{core}(esquema(templ_{str}), \overline{esquema}(\overline{t}_{str}), t.\widehat{esquema})$$

y la tupla $\widehat{t.valor}$ satisface el predicado:

$$\mu_{core}(valor(templ_{str}), \overline{canal}(templ_{str}, \overline{t}_{str}), \widehat{t.valor})$$

Es decir, $\widehat{t_{str}}$ se reconstruye a partir de una tupla_esquema y una tupla_valor, de forma que estas tuplas satisfacen el predicado de elección en el modelo *core*. La tupla_esquema $\widehat{t.esquema}$ satisface el predicado μ_{core} en un multiconjunto de tuplas_esquemas y la tupla_valor $\widehat{t.valor}$ satisface el predicado μ_{core} en el canal de la plantilla_estructurada.

Esta definición es, en cierto sentido, la construcción central de todo el modelo, pues en ella se refleja el modo de selección complejo que fue implementado en el servicio WCS. En la definición se observa cómo conviven varios espacios de tuplas y cómo, para completar una selección en el modelo *estructurado* es necesario consultar, a través del canal, en las tuplas de atributos, y allí, respecto a las tuplas de valores. Esto se corresponde fielmente con el proceso descrito en el capítulo anterior, en el que, para resolver una consulta en el *Componente Espacio de Tuplas XML* se hacía necesario realizar una operación `take` sobre *JavaSpaces* para obtener un objeto `RandomChannel`.

Para que μ_{str} sea efectivamente un predicado de elección Linda en el modelo *estructurado* debemos demostrar que, junto con el predicado mtc_{str} (definición 3.9), satisface las reglas de correspondencia de Linda, definidas en (3.1). En el teorema 3.20 demostraremos que se satisfacen dichas reglas. Previamente se plantean dos proposiciones que facilitan la posterior demostración del teorema.

Proposición 3.18. *Sea* t_{str} T_{str} *una tupla_estructurada y sea* $\overline{t}_{str}$ $\overline{T}_{str}$ *una multiconjunto de tuplas_estructuradas, entonces* $esquema(t_{str})$ $\overline{esquema}(\overline{t}_{str}) = \overline{esquema}(t_{str}\ \overline{t}_{str})$

Demostración. Se deduce directamente de la definición de *esquema* y $\overline{esquema}$. ■

Proposición 3.19. *Sean* t_{str} T_{str} *una tupla_estructurada,* $\overline{t}_{str}$ $\overline{T}_{str}$ *un multiconjunto de tuplas_estructurada y* $templ_{str}$ $Templ_{str}$ *una plantilla_estructurada. Si* $mtc_{core}(esquema(templ_{str}), esquema(t_{str}))$ *entonces* $valor(t_{str})$ $\overline{canal}(templ_{str}, \overline{t}_{str}) = \overline{canal}(templ_{str}, t_{str}\ \overline{t}_{str})$

Demostración. Dado que $mtc_{core}(esquema(templ_{str}), esquema(t_{str}))$, por la definición 3.15, $valor(t_{str})$ $\overline{canal}(templ_{str}, \overline{t}_{str})$ entonces,

$$valor(t_{str})\ \overline{canal}(templ_{str}, \overline{t}_{str}) = \overline{canal}(templ_{str}, t_{str}\ \overline{t}_{str})$$

■

Teorema 3.20. *Los predicados definidos previamente,* μ_{str} *y* mtc_{str}*, cumplen las reglas de correspondencia de Linda:*

$$(i) \quad \frac{mtc_{str}(templ_{str}, t_{str})}{\mu_{str}(templ_{str}, t_{str}\ \overline{t}_{str}, t_{str})}$$

$$(ii)\quad \frac{\nexists\ t_{str}\quad \bar{t}_{str}\ :\ mtc_{str}(templ_{str}, t_{str})}{\mu_{str}(templ_{str}, \bar{t}_{str},\quad _T)}$$

Demostración. (i) Sea $t_{str}\quad T_{str}$ una tupla_estructurada tal que $mtc_{str}(templ_{str}, t_{str})$. Entonces, por el teorema 3.14:

$$mtc_{core}(esquema(templ_{str}), esquema(t_{str})) \tag{3.12}$$

$$mtc_{core}(valor(templ_{str}), valor(t_{str})) \tag{3.13}$$

por la definición 3.4 y (3.12)

$$\mu_{core}(esquema(templ_{str}), esquema(t_{str})\ \overline{esquema}(\bar{t}_{str}), esquema(t_{str})) \tag{3.14}$$

por la proposición 3.18 y (3.14)

$$\mu_{core}(esquema(templ_{str}), \overline{esquema}(t_{str}\ \bar{t}_{str}), esquema(t_{str})) \tag{3.15}$$

por la definición 3.4 y (3.13)

$$\mu_{core}(valor(templ_{str}), valor(t_{str})\ \overline{canal}(templ_{str}, \bar{t}_{str}), valor(t_{str})) \tag{3.16}$$

por la proposición 3.19 y (3.16)

$$\mu_{core}(valor(templ_{str}), \overline{canal}(templ_{str}, t_{str}\ \bar{t}_{str}), valor(t_{str})) \tag{3.17}$$

Ahora, podemos construir la tupla_estructurada

$$t_{str} = construir\_tupla_{str}(esquema(t_{str}), valor(t_{str}))$$

entonces, por la definición 3.17, (3.15) y (3.17)

$$\mu_{str}(templ_{str}, t_{str}\ \bar{t}_{str}, t_{str})$$

(ii) Si $\nexists\ t_{str}\quad \bar{t}_{str}$ tal que $mtc_{str}(templ_{str}, t_{str})$ entonces, por el teorema 3.14, podría haber dos posibilidades:

La primera opción es que no haya una correspondencia de esquemas. Es decir,

$$\nexists\ t.esquema\quad \overline{esquema}(\bar{t}_{str}) :\quad mtc_{core}(esquema(templ_{str}), t.esquema)$$

por la definición 3.5

$$\mu_{core}(esquema(templ_{str}), \overline{esquema}(\bar{t}_{str}),\quad _T)$$

y entonces, por la definición 3.16

$$\mu_{str}(templ_{str}, \bar{t}_{str},\quad _T)$$

La segunda opción es que haya una correspondencia de esquemas pero no de valores en el canal de la plantilla $templ_{str}$. Es decir,

$$t_{str} \quad \bar{t}_{str} : \quad mtc_{core}(esquema(templ_{str}), esquema(t_{str}))$$

pero,

$$\nexists \ t_{str} \quad \overline{canal}(templ_{str}, \bar{t}_{str}) : \quad mtc_{core}(valor(templ_{str}), valor(t_{str}))$$

por la definición 3.5

$$\mu_{core}(valor(templ_{str}), \overline{canal}(templ_{str}, \bar{t}_{str}), \quad _T)$$

y entonces, por la definición 3.17 y por la definición 3.16

$$\mu_{str}(templ_{str}, \bar{t}_{str}, \quad _T)$$

■

Como conclusión de este teorema, podemos afirmar que el predicado μ_{str}, definido en la definición 3.17 es un **Predicado de elección Linda** para el modelo *estructurado*.

3.4. Conclusiones del estudio formal

Como acabamos de ver, nuestro principal objetivo en este capítulo ha sido alcanzado: el modelo *estructurado* respeta la semántica de Linda. Puesto que dicho modelo ha sido abstraído de un sistema software *real* de coordinación de servicios Web, dicha propiedad se transfiere, en cierto modo, hasta el sistema real objeto de estudio.

Ahora bien, dado que las propiedades han sido demostradas *en el modelo formal*, y no en el sistema que lo inspira, no podemos considerar que los teoremas de la sección anterior sean una prueba de la *corrección* del Servicio de Coordinación (en el sentido de que respete la semántica de Linda). Ese problema es mucho más difícil, puesto que depende, entre otros factores, de la implementación realizada en Java, y, por tanto, indirectamente, de la semántica del lenguaje de programación Java. Abordar la verificación formal completa de un sistema Java de la complejidad y tamaño del Servicio de Coordinación es algo que está fuera de las posibilidades de la tecnología actual.

Así pues, debemos contentarnos con probar propiedades en el modelo, y ajustar el modelo para que refleje fielmente las características del sistema real. De este modo, aumentamos la confianza en el sistema, pues se ha demostrado la corrección de un modelo de una versión "idealizada" del mismo. Esto asegura, al menos, que no aparecen *errores conceptuales* en las ideas que han guiado la implementación del Servicio Web de Coordinación.

Otra consecuencia, quizás más importante, es que el modelo formal nos permite un análisis más "puro" del sistema bajo estudio (en el sentido de que el estudio está menos

enturbiado por detalles concretos de implementación, relacionados con características de bajo nivel). Esto facilita la propuesta de generalizaciones y mejoras del sistema.

En nuestro caso concreto, una característica que se hace explícita en el modelo formal es que el Servicio de Coordinación ha extendido el modo de emparejar tuplas para realizar operaciones de lectura. En lugar de la igualdad literal entre constantes (es decir, la igualdad dentro del tipo de dato al que la constante pertenece), se ha realizado un proceso de emparejamiento en "dos pasos": primero el relativo a la estructura de la tupla (abstracción del esquema XML en la aplicación real), y después el habitual dentro del modelo Linda. Esta constatación permite generalizar fácilmente la teoría: basta reemplazar el mecanismo de emparejamiento de Linda por cualquier función de *matching*.

Pero dicha generalización teórica, que podría conllevar un estudio paralelo al realizado por otros autores en la formalización de la comunicación generativa, nos ha parecido menos interesante que sus aplicaciones prácticas. Como ilustración de esta idea, en el siguiente capítulo desarrollamos dos propuestas diferentes de *matching* complejo. En la primera, extraída directamente de los campos en los que el Servicio de Coordinación ha sido utilizado realmente (esto es, del ámbito de los *Sistemas Basados en la Localización*), introducimos un *matching semántico*. Para plasmar la validez de esta idea, más allá de la simple propuesta teórica, nos hemos visto obligados a realizar una investigación que está más relacionada con el procesamiento del lenguaje natural que con la ingeniería del software.

En la segunda parte del capítulo realizamos una generalización de corte más teórico: extender el emparejamiento para que las consultas puedan ser más complejas, basadas en restricciones. En este apartado, nos quedamos tan solo con el estudio teórico. Su interés en la práctica se explotará después, en el capítulo 5.

Consideramos que estas aportaciones ponen en valor una de las apuestas de nuestro tipo de investigación: que los métodos formales no sólo tienen interés en sí mismos (como propuestas matemáticas) o por los objetivos *directos* para los que fueron creados (demostrar la corrección de los diseños, en nuestro caso), sino también porque de su uso pueden extraerse importantes consecuencias *prácticas*.

Capítulo 4

Dos aplicaciones del emparejamiento complejo

4.1. Primer ejemplo de aplicación: *matching semántico*

4.1.1. Presentación del problema

Como hemos explicado al terminar el capítulo anterior, una de las consecuencias de nuestro análisis formal fue el apercibimiento de que la función de correspondencia o emparejamiento de Linda puede ir mucho más allá de la simple comparación literal de datos. Una variante puede consistir en emparejar no solo datos *iguales* si no admitir también la correspondencia entre datos *aproximados*, respecto a alguna noción de similaridad. Dado que las primeras aplicaciones industriales del Servicio de Coordinación se desarrollaron en el ámbito de los sistemas basados en la localización (como explicamos en el apartado 2.4), tuvimos una ocasión inmejorable de poner en valor nuestras ideas acerca del emparejamiento *complejo*. Esto es así porque en los Sistemas Basados en la Localización, y más en general en los sistemas basados en datos geográficos, las fuentes de información disponibles son heterogéneas y, por tanto, es poco frecuente que los mismos conceptos sean referidos por distintas organizaciones con exactamente el mismo término. Por poner solo un ejemplo simple, si se trata de cruzar información de callejeros preparados por distintas instituciones, es posible que un mismo objeto sea marcado en uno de ellos a través de término "calle", mientras que en el otro es recuperado a través de "vía". Si el Servicio de Coordinación tiene que integrar datos de las dos organizaciones del ejemplo, tal vez sería conveniente, si ninguna tupla se corresponde con una plantilla que contiene el término "vía", que intentase también un emparejamiento con tuplas que tengan la cadena de caracteres "calle" en la posición correspondiente. (Otro ejemplo evidente de aplicación de estas técnicas aparece en el caso de integración de sistemas de información multilingües; pero este caso no ha sido tratado en la práctica todavía).

Resumiendo la anterior discusión, podemos enunciar nuestro problema como un intento de mejorar la interoperabilidad entre los servicios cooperantes dotando al espacio de tuplas de un mecanismo de correspondencia que permita recuperar tuplas acorde a patrones de búsqueda más complejos, que incluyan *conceptos*, y no sólo *términos sintácticos* (cadenas de caracteres, en definitiva). Versiones preliminares de este trabajo dieron lugar a las publicaciones [MAB+01], [MBG+02] y [ÁBM+03a].

Ejemplos de otras propuestas relacionadas con el nuestro trabajo son *RDFSpaces* y *Triple-Spaces*. El paradigma *Triple-Space Computing* [Fen04] utiliza un espacio de tuplas para modelar la interacción de servicios Web. En este caso, el espacio de tuplas contiene tripletas RDF, es decir, tuplas de la forma <`sujeto, predicado, objeto`> (`http://www.w3.org/TR/rdf-concepts`). Estas tripletas contienen los datos que se intercambian las aplicaciones y describen información semántica que puede ser directamente procesada por la máquina. Además, si los `objetos` son `sujetos` en otras tripletas, se tiene también una estructura de tripletas que captura la información semántica en forma de grafo. Por otra parte, *RDFSpaces* [TBN05] es la continuación de un trabajo previo del mismo grupo denominado *XMLSpaces* [TG01]. *XMLSpaces* es una extensión de *TSpaces* (implementación de Linda por parte de IBM) con el objetivo de dar soporte a la gestión de documentos XML como tuplas del espacio.

La principal dificultad para realizar una búsqueda basada en conceptos es el problema de la *desambiguación semántica*. En el siguiente apartado se describe un ejemplo de aplicación, en el ámbito de los sistemas de información geográfica, donde se aplica un método heurístico para la desambiguación de términos obtenidos de un tesauro, utilizando la base de conocimiento léxica *WordNet* ([Mil90], `http://wordnet.princeton.edu`). La importancia de este ejemplo radica en que el método de desambiguación utilizado puede ser incorporado en el procedimiento de correspondencia en dos pasos, lo que nos permitiría obtener tuplas a partir de conceptos.

La idea básica consiste en aprovechar los recursos léxicos disponibles (tesauros, diccionarios, bases de conocimiento léxicas, ...) y las técnicas de tratamiento del lenguaje natural para mejorar la organización y recuperación de la información. El problema de desambiguación se aborda mediante un método heurístico basado en la estructura jerárquica de *WordNet*. Dado un término ambiguo, se utiliza un sistema de votos que pretende determinar el significado "más cercano" a los significados de las palabras tomadas como contexto, en este caso una rama completa de un tesauro. Los votos llevan asociados unos pesos que se han ido ajustando empíricamente para corregir los desvíos detectados. Este método se ha incorporado en una aplicación para la gestión y administración de tesauros dentro del proyecto ISIGIS, destinado al desarrollo de la tecnología necesaria para la puesta en marcha de la Infraestructura Española de Información Geográfica (véase [BBG+01], [NLN+01]).

4.1.2. Recuperación semántica de información

La mayoría de las herramientas de búsqueda de información como, por ejemplo, los motores de búsqueda de Internet actuales, recuperan documentos que contienen los términos que el usuario ha introducido en la consulta. Google también utiliza los enlaces que apuntan a una página Web para priorizar dicha página.

En algunos casos, los resultados de la búsqueda no son los esperados, bien porque se obtienen páginas Web no relacionadas con lo que buscamos, bien porque el motor ha asignado una baja prioridad a las páginas relevantes para la consulta. Hay, al menos, tres situaciones que pueden producir estos errores [SF05]:

Polisemia Realizamos una consulta en el buscador introduciendo un término y encontramos páginas Web que contienen el término pero con un significado distinto.

Sinonimia Realizamos una consulta en el buscador introduciendo un término y no encontramos páginas Web relacionadas con aquello que nos interesa porque tales páginas Web contienen un sinónimo del término, pero no el propio término.

Multilingüismo Realizamos una consulta en el buscador introduciendo un término en un idioma y no encontramos páginas Web relacionadas con lo que nos interesa porque están en otro idioma.

En todos los casos, el problema reside en que mediante una "búsqueda sintáctica", o basada en términos, no podemos identificar de forma precisa aquello en lo que estamos interesados. El concepto de Web semántica intenta resolver estos problemas incluyendo informaciones adicionales que describan el contenido, el significado y la relación de los datos en la Web. El formalismo más ampliamente utilizado en el contexto de la Inteligencia Artificial para especificar estas conceptualizaciones son las *ontologías*. Una ontología es una descripción, similar a una especificación formal de un programa, de los conceptos y relaciones que pueden existir en un dominio de interés y se construye para ser compartida por un grupo de personas, organizaciones o agentes [Gru93].

Un problema adicional en la construcción y uso de ontologías es el que se conoce como *correspondencia o alineamiento entre ontologías* (*ontology mapping*), que trata de resolver el problema de detectar cuándo conceptos definidos en distintas ontologías son equivalentes o existe alguna relación entre ellos.

Este problema de correspondencia también se plante entre otras fuentes de conocimiento léxico. Así, los *tesauros* tienen un organización jerárquica de términos que son utilizados en un ámbito concreto del conocimiento. Los *diccionarios* electrónicos mantienen una estructura similar a los diccionarios clásicos, pero permiten la consulta por software. Cuando los significados no son almacenados explícitamente, pero se mantiene una organización más estructurada respecto a las relaciones entre los distintos términos (relaciones basadas en propiedades gramaticales como la sinonimia, homonimia, etc.) hablamos de *bases de conocimientos léxicas* (siendo la ya citada *WordNet* un

ejemplo paradigmático). Por último, todas estas fuentes de información pueden ser utilizadas para extraer *listas controladas* de términos, que son empleadas por los programas de ayuda a la catalogación y a la captura de datos, para evitar errores de tecleo y romper ambigüedades. Obviamente (y para añadir mayor complejidad) estas listas controladas también pueden ser definidas de modo *ad-hoc* por cada organización, o incluso para cada aplicación informática. Una referencia reciente en la que ver en la práctica estos distintos conceptos (cuyas fronteras no están siempre claras tanto en la literatura como en sus aplicaciones prácticas) es [BH08].

4.1.3. Enriquecimiento de tesauros con *WordNet*. Una aproximación heurística

La información geográfica, o *geodatos*, es información que describe fenómenos asociados directa o indirectamente con una localización (y posiblemente un tiempo y una orientación) relativa a la superficie de la Tierra. Esta clase de datos digitales está siendo recogida desde hace más de 35 años. La velocidad de esta recolección de geodatos se incrementa rápidamente con avances en tecnologías tales como los sistemas de imágenes de satélite de alta resolución, los sistemas de posicionamiento global (GPS, Glonass), los sistemas de bases de datos, las nuevas tecnologías de software aplicables al geoprocesamiento y con el creciente número de personas y organizaciones que están recogiendo y utilizando geodatos. A todo esto hay que añadir que alrededor de un 80 % de las bases de datos utilizadas en la administración contienen referencias geográficas (direcciones postales, coordenadas cartográficas o distribución por municipios, sectores, barrios, secciones censales, etc.), por lo que puede hacerse un tratamiento de estos datos relacionado con la localización. En diferentes actividades se requiere la adquisición de este tipo de información, y su combinación con otras, haciendo uso de Sistemas de Información Geográfica (SIG). Para conseguir esto es necesario, por una parte, saber si existe esa información geográfica y dónde conseguirla y, luego, disponer de mecanismos para adquirirla.

Los geodatos se describen mediante un conjunto de informaciones denominadas *metadatos geoespaciales* y se almacenan en un *Catálogo*, termino acuñado por el OpenGIS Consortium ([OGC99]) para describir un conjunto de servicios de interfaz que soportan organización, búsqueda y acceso a información geoespacial. Los servicios del Catálogo ayudan a usuarios y aplicaciones software a encontrar información que existe en un entorno de computación distribuido.

La calidad de un catálogo de información geográfica depende esencialmente de los metadatos por los que están indexadas las entradas de geodatos. Una parte importante de los metadatos es la sección de palabras clave, que permite la clasificación de los geodatos mediante un conjunto de términos que representan conceptos o categorías en uno o más dominios. Dicha clasificación hace más fácil la búsqueda de información a partir de las palabras clave. Sin embargo, la clasificación con palabras clave es una labor subjetiva. Con objeto de reducir la subjetividad de las clasificaciones se utilizan, en la

medida de lo posible, listas de palabras clave predefinidas o *listas controladas*. Estas listas definen un conjunto de términos representando conceptos y categorías dentro de uno o varios dominios temáticos que permiten clasificar y caracterizar datos provenientes de diferentes fuentes con términos relacionados. El uso de estas palabras clave predefinidas facilita la asociación entre un vocabulario seleccionado y una gran colección de datos geográficos. El problema está en las diferencias semánticas que existen entre los tesauros utilizados por las diversas comunidades interesadas por compartir mapas e información espacial.

4.1.4. Uso de tesauros para la clasificación de metadatos y para búsquedas avanzadas

Debido a la subjetividad de la tarea de clasificación, incluso dentro de una misma organización o dominio, es posible encontrar clasificaciones distintas de los mismos datos. Por esta razón, conviene realizar la implementación de las listas controladas mediante un tesauro, puesto que, además de los términos, aporta un conocimiento sobre las relaciones entre ellos. Dado que el proyecto ISIGIS supone el desarrollo de una infraestructura nacional de información geográfica, se debe tener en cuenta además que cuando se clasifica la información mediante palabras clave de un tesauro, esta clasificación está realizada desde el punto de vista de la organización que diseña dicho tesauro. Sin embargo, el catálogo debe permitir compartir información entre diversas organizaciones que utilizan diferentes términos, e incluso idiomas, para referirse a los mismos conceptos. Por lo tanto, no basta con las relaciones de los términos dentro de un tesauro, sino que es necesario interrelacionar distintos tesauros para poder clasificar los metadatos en función de cada uno de ellos.

Además, esta imprecisión o subjetividad de la tarea de clasificación queda patente por el hecho de que aunque los SIG son potentes herramientas cuantitativas de razonamiento geoespacial, las personas razonan sobre el espacio de forma cualitativa. La información verbal sobre localizaciones puede conllevar ciertos aspectos de imprecisión, aunque las personas son capaces de deducir información a partir de dichas descripciones (véase [DF99]). Otro problema añadido es la utilización de términos ingleses en muchos de los tesauros, que son difícilmente traducibles para usuarios de diferentes culturas e idiomas.

Una vez presentada la importancia de los tesauros, veamos cómo utilizarlos. Un tesauro (norma ISO 2788) es un conjunto de términos seleccionados del lenguaje natural representando el vocabulario de un cierto campo. Uno de sus usos más común es la recuperación de información, aunque también se usa para el indexado y almacenamiento de datos. Un tesauro multilingüe (norma ISO 5964) puede ser un tesauro normal que contiene para cada término traducciones a otros idiomas, o bien puede ser un conjunto de tesauros que almacenan el mismo subconjunto del lenguaje natural para cada uno de los idiomas. Los tesauros pueden definir cuatro tipo de relaciones entre los términos: *sinonimia* (mismo significado), *términos relacionados* (tienen significados cercanos pero no son sinónimos), *relaciones jerárquicas* (entre términos más genéricos y términos más

específicos) y *traducción* (a distintos idiomas).

En este caso se ha utilizado como ejemplo prototipo el tesauro GEMET *"GEneral Multilingual Environmental Thesaurus"*, propuesto por el *European Topic Centre on Catalogue of Data Sources* de la Agencia Europea de Medioambiente (accesible en `http://www.eionet.europa.eu/gemet`). GEMET es un tesauro de ámbito medioambiental. Actualmente consta de 6000 términos organizados en 109 ramas y traducido a 22 idiomas.

Para dotar a un sistema de un mecanismo de consulta inteligente se suele hacer uso de *ontologías* o *bases de conocimiento léxicas*, como *WordNet* [Mil90] y *EuroWordNet* [GVPC98], creadas desde un punto de vista global, de forma que el usuario formule sus consultas y el sistema tenga la responsabilidad de gestionar la recuperación basándose en los conceptos o temas involucrados en la petición. Estas bases de conocimiento, a diferencia de un diccionario, organizan las palabras en conjuntos de sinónimos e incluyen diferentes relaciones como *parte-de* o *es-un* (en términos lingüísticos: *meronimia*, *hiperonimia*). Un concepto importante dentro de *WordNet* es el de *synset*: un conjunto de sinónimos identificados mediante un número. Dicho concepto se utiliza en nuestra aproximación, así como en otras debidas a otros autores. Un ejemplo de aplicación de *WordNet* aparece en [Vos01], donde se usa para construir un árbol de términos que sirva de interfaz para la recuperación de información en una colección de documentos sobre un dominio específico. El principal problema que se encuentra es "comprender" el significado de los términos que aparecen en los documentos, es decir, el problema de la desambiguación semántica.

4.1.5. El problema de la extracción de conceptos o desambiguación semántica

La desambiguación semántica (en inglés, *word sense disambiguation*) es, tal vez, el mayor problema abierto, a nivel léxico, del procesamiento de lenguaje natural [RY97], y tiene aplicación en tareas tales como la traducción automática, el reconocimiento de voz o la recuperación de información. Una palabra que cambia de significado según el contexto en el que se utiliza se dice que es *polisémica*. Desambiguar una palabra polisémica consiste en determinar cuál de sus significados es el más apropiado en un contexto particular, caracterizado como un conjunto de palabras relacionadas con la palabra ambigua.

Las diferentes técnicas que tratan de resolver el problema se distinguen según el material de aprendizaje disponible. Los métodos de entrenamiento supervisados usan información obtenida tras un entrenamiento en un corpus que, previamente, ha sido semánticamente desambiguado. Los métodos de entrenamiento sin supervisar son aquellos que usan información obtenida de textos sin ningún tipo de anotación semántica. También existen métodos que utilizan las definiciones proporcionadas por diccionarios electrónicos, bases de conocimiento léxicas o tesauros.

En los métodos de desambiguación supervisados cada ocurrencia de una palabra

en el corpus está anotada con su significado más apropiado al contexto. Basándose en esta información, el objetivo de estos métodos es construir un clasificador que clasifique correctamente los nuevos casos. Un ejemplo de este tipo es el clasificador Bayesiano que aparece en [GCY92]. Se basa en calcular la probabilidad de cada uno de los significados s_i de una palabra ambigua a dado un contexto C, probabilidad que denotamos por $P(s_i\ C)$, y elegir el significado más probable. El valor $P(s_i\ C)$ se calcula mediante el Teorema de Bayes:

$$P(s_i\ C) = \frac{P(C\ s_i)P(s_i)}{P(C)}$$

Si solo queremos maximizar esta cantidad, podemos omitir el denominador, y si asumimos la hipótesis de independencia entre las palabras del contexto, que claramente no es cierta en la realidad, podemos factorizar su cálculo:

$$P(C\ s_i) = \prod_{w \in C} P(w\ s_i)$$

y estimar los valores de $P(w, s_i)$ y $P(s_i)$ mediante los estimadores de máxima verosimilitud que se obtienen del corpus de entrenamiento:

$$P(w, s_i) = \frac{N(w, s_i)}{N(s_i)}$$

$$P(s_i) = \frac{N(s_i)}{N(a)}$$

donde $N(w, s_i)$ es el número de ocurrencias de w en un contexto del significado s_i, $N(s_i)$ es el número de ocurrencias de s_i y $N(a)$ es el número total de ocurrencias de la palabra ambigua a.

Los métodos sin supervisar intentan distinguir entre los significados de una palabra polisémica sin la ayuda de ejemplos desambiguados. Aunque los métodos supervisados obtienen buenos resultados, todos ellos requieren textos anotados manualmente para entrenar los algoritmos. Para muchos dominios no existen estos textos y su creación resulta costosa. En estas situaciones, los métodos sin supervisar intentan distinguir entre los significados de una palabra basándose solamente en las características que pueden ser extraídas automáticamente de un texto sin anotar.

Estrictamente hablando, la desambiguación completamente sin supervisar no es posible, si lo que pretendemos es etiquetar las ocurrencias de una palabra como perteneciente a un significado o a otro, puesto que se necesita alguna caracterización previa de los significados. Lo que sí se puede realizar de forma automática es una discriminación de diferentes clases en las que la ocurrencia de la palabra ambigua tiene unas características distintas, e interpretar estas clases como posibles usos o significados de la palabra.

Los métodos basados en diccionarios se caracterizan porque se apoyan en las definiciones que aparecen en diccionarios electrónicos o en las agrupaciones de palabras de los tesauros. En algunas ocasiones se combinan con técnicas de entrenamiento supervisadas o sin supervisar. Un primer intento aparece en [Les86], donde se parte de la idea de que las definiciones de las palabras que aparecen en un diccionario son buenos indicadores de los significados que describen. Dada una palabra ambigua y un conjunto de palabras contexto, calcula, para cada significado de la palabra ambigua, el número de palabras del contexto que aparecen en la glosa que define el significado en el diccionario. Elige el significado que tenga más palabras comunes.

En [Yar92] se describe un método en el que se utilizan las categorías del tesauro de Roget [Cha77]. Este tesauro, en su versión de 1977, define 1042 categorías. Cada categoría consiste en un conjunto de palabras relacionadas y sirve como aproximación para definir una clase conceptual diferente. Dada una palabra polisémica, el problema de desambiguación consiste en seleccionar la categoría del tesauro más verosímil, tomando como contexto C las cien palabras que la rodean en el texto en el que aparece.

Primero se extrae, para cada una de las palabras de una categoría, las 100 palabras que rodean cada ocurrencia de la palabra en un corpus anotado, en este caso de la *Grolier's encyclopedia* (accesible en `http://ea.grolier.com/`), creando el conjunto de palabras que aparecen típicamente en el contexto colectivo de la categoría. A continuación identifica las palabras que aparecen significativamente con más frecuencia en el contexto de la categoría que en otros puntos del corpus (*salient words*) mediante el cociente entre el número de ocurrencias de la palabra w, dada la categoría $RCat$, y el número de ocurrencias de la palabra en el corpus completo:

$$\frac{P(w\ RCat)}{P(w)}.$$

Cuando alguna de estas palabras aparece en el contexto de una palabra ambigua, entonces hay evidencia de que la palabra pertenece a la categoría indicada. Si aparecen varias de estas palabras, usando el teorema de Bayes, se suman sus pesos, sobre todas las palabras del contexto y se determina la categoría cuya suma es mayor; es decir se elige la categoría $RCat$ que maximiza el siguiente valor:

$$\sum_{w \in C} log \frac{P(w\ RCat)P(RCat)}{P(w)}$$

(Como se trata de maximizar una función, se toma el logaritmo para maximizar un sumatorio, en lugar del producto que aparecería al considerar, en el Teorema de Bayes, la hipótesis de independencia.)

[Res95] presenta un método para desambiguar grupos de nombres relacionados. En lugar de resolver la polisemia de una palabra tomando como contexto un conjunto de palabras cercanas en el texto, parte de un grupo de palabras obtenidas previamente en un

tesauro o por algún algoritmo de agrupación. El método se basa en la noción de *similaridad semántica* utilizando la relación jerárquica *es_un* de *WordNet*. Está inspirado en la observación de que cuanto más similares son dos palabras polisémicas, más informativo será el concepto más específico c que domina a ambas y nos proporcionará información sobre cuál es el significado más relevante de cada palabra. Dadas dos palabras w_1 y w_2 define su similaridad semántica como:

$$sim(w_1, w_2) = \max_{c \quad \text{subsumers}(w_1,w_2)} [-logP(c)],$$

donde $subsumers(w_1, w_2)$ es el conjunto de *synsets* de *WordNet* que son ancestros comunes de w_1 y w_2 para alguno de los significados de cada palabra. El concepto c que maximiza la expresión anterior es el concepto más informativo que domina a w_1 y w_2. Las probabilidades las estima de un corpus calculando el cociente entre el número de palabras que tienen algún significado dominado por el concepto c y el número total de palabras observadas.

En [AR96] se presenta un método que utiliza la estructura jerárquica de *WordNet* y la noción de *distancia conceptual* entre conceptos, noción que se define como la longitud del camino más corto entre los conceptos en una red semántica. El método se evalúa automáticamente sobre *SemCor* (accesible en `http://www.cs.unt.edu/~rada/downloads.html`), una versión anotada del *Corpus Brown* (disponible en `http://icame.uib.no/cd/`). La distancia conceptual intenta proporcionar una medida para determinar la cercanía de los significados entre dos palabras, tomando como referencia una estructura de red jerárquica. Según [AR96], la medida de la distancia conceptual debe tener en cuenta aspectos como:

- la longitud del camino más corto entre los conceptos;
- la profundidad en la jerarquía: a mayor profundidad los conceptos deben estar más cercanos;
- la densidad de los conceptos en la jerarquía: conceptos en una parte densa están relativamente más cercanos que en una zona dispersa;
- ser independiente del número de conceptos que está midiendo.

En su trabajo, [AR96] utiliza una fórmula de *densidad conceptual* para comparar áreas de subjerarquías. Dado un concepto c en la cima de una subjerarquía, y dado *nhyp* (número medio de hipónimos por nodo), define la *densidad conceptual* del concepto c, cuando su subjerarquía contiene m marcas (significados de las palabras a desambiguar), como el cociente entre el área esperada para una subjerarquía que contenga m marcas (significados) y el área real. Tras simplificar la altura de la jerarquía, se obtiene la siguiente fórmula, en la que $descendientes_c$ denota el número de descendientes de c en la jerarquía (aunque añade el factor 0,20 por razones empíricas):

$$CD(c,m) = \frac{\sum_{i=0}^{m-1} nhyp^{i^{0,20}}}{descendientes_c}$$

Dada una palabra para desambiguar, el algoritmo extrae, en primer lugar, un conjunto de palabras contexto y obtiene de *WordNet* sus significados y sus hiperónimos. A continuación calcula la densidad conceptual de cada concepto en *WordNet* según los significados que contiene en su subjerarquía. Por último se selecciona el concepto con mayor densidad y se toman los significados dominados por él como los significados correctos de las palabras a desambiguar.

[DPR99] parte de una taxonomía de significados en español obtenidos automáticamente de un diccionario monolingüe con el objetivo de realizar un alineamiento con *WordNet*, es decir, asignar a cada significado en español un significado (o *synset*) de *WordNet*. Para ello utiliza un algoritmo de satisfacción de restricciones (*relaxation labeling*). Por otra parte, en [FAGV01] se evalúa el papel de la distancia conceptual en la desambiguación semántica mediante la generalización de la medida de [AR96], mejorando el algoritmo original y llegando a la conclusión de la necesidad de la combinación de diversas fuentes de conocimiento para una correcta desambiguación.

Por último, otra forma de abordar el problema es la que aparece en [WOB98], donde se explora la posibilidad de formular el problema de desambiguación mediante una red bayesiana ([Pea88, CGH96]) extraída de la estructura de *WordNet*. La idea es utilizar la relación léxica entre conceptos representada en la red semántica de *WordNet* para construir una red basada en la dependencia causal entre hipérnimos e hipónimos. Se describen dos opciones para la representación léxica: un nodo por palabra o un nodo por significado y otras dos opciones para la representación de las relaciones léxicas: un enlace en la dirección hiperónimo hipónimo o en la dirección hipónimo hiperónimo. Se propone construir las tablas de probabilidad condicionada asociada a cada nodo con distribuciones uniformes o mediante aprendizaje en un corpus. Dada una palabra ambigua y un conjunto de palabras contexto, el algoritmo de desambiguación consistiría en construir la red bayesiana que abarque todas las palabras con sus significados según la estructura jerárquica de *WordNet*. Las palabras contexto formarían la evidencia para propagar los valores probabilísticos por la red y elegir el significado de la palabra ambigua con mayor valor.

Para terminar este apartado, comentar que en las competiciones SENSEVAL (cuyos resultados pueden consultarse en `http://www.senseval.org/`) se evalúan periódicamente un gran número de programas de computador que realizan automáticamente la tarea de desambiguación.

4.1.6. Descripción y evaluación de nuestro método

El método de desambiguación que presentamos en este apartado entra dentro de la clasificación de métodos no supervisados basado en la estructura jerárquica de *WordNet* y es comparable con los que se describen en [AR96] y en [Res95]. Para ilustrar el método, partimos de los términos del tesauro GEMET. El objetivo es analizar los términos del tesauro y, para cada palabra del mismo, determinar el significado *WordNet* "más cercano" a los significados del resto de términos que forman una rama completa en GEMET.

Cada rama de GEMET se corresponde con un árbol cuya raíz es un término principal que no está incluido en ningún otro término. Por ejemplo, la primera rama es la correspondiente al término `accident` con todos sus descendientes (ver figura 4.1).

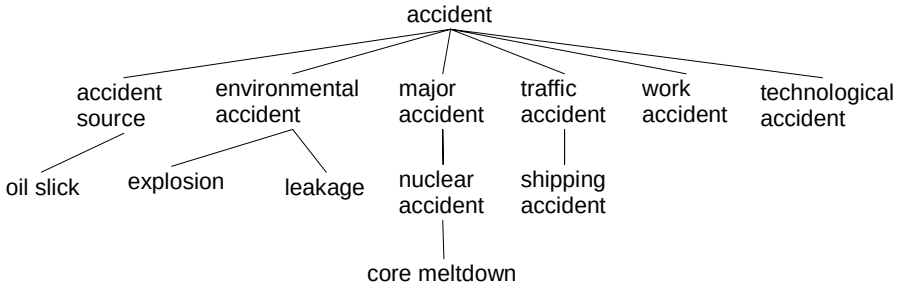

Figura 4.1: La rama `accident` del tesauro GEMET

Cada término del tesauro se puede representar mediante el camino que se recorre desde la raíz de la rama a la que pertenece hasta el propio término. El método de desambiguación recorre las ramas siguiendo una estrategia de primero en profundidad. Para cada término, se considera que el resto de términos de su misma rama constituyen su contexto y se extrae de *WordNet* todos sus posibles significados. En el caso de un término compuesto (formado por más de una palabra) y que no esté incluido en *WordNet*, se extrae el significado de cada una de las palabras que componen el término. Un significado o concepto en *WordNet* se representa mediante un *synset* (recuérdese: un conjunto de sinónimos identificados mediante un número). Como los *synsets* en *Wordnet* mantienen una estructura jerárquica, es posible obtener un camino de *synsets* para cada significado extraído. Por ejemplo, el término `accident` tiene dos significados en *WordNet*, correspondientes a los siguientes caminos:

```
synset_1: [5443572] accident
   path: event happening trouble misfortune mishap accident
synset_2: [5443380] accident, fortuity, chance, event
   path: event happening accident
```

En lugar de modelizar la creencia del significado "más cercano" mediante cálculo de probabilidades, como se hace en [GCY92], se optó por un sistema de votos que tuviera

en cuenta la aportación de las palabras del contexto, sin tener que asumir la hipótesis de independencia entre ellas, utilizada con frecuencia en los clasificadores bayesianos para simplificar la complejidad de las operaciones. El método también se basa en la estructura jerárquica de *WordNet* con la idea de que "dos términos serán más cercanos si coinciden con más subjerarquías (si tienen más hiperónimos comunes)". Dado un término de GEMET, se consideran como significados candidatos todos aquellos *synsets* de *WordNet* que contienen al término. Para cada *synset* candidato se obtiene su camino de *synsets* en la estructura jerárquica de *WordNet*, y el sistema de votación compara este camino con el resto de caminos del término y con los caminos de otros términos de la misma rama de GEMET (que actúan como contexto). Para cada par de caminos, el sistema cuenta el número de *synsets* comunes (hiperónimos comunes), obteniendo un valor acumulado para el camino inicial. Cada camino vota a todos los otros caminos, y es votado por todos los caminos de todos los términos de su contexto. Una vez obtenido los resultados para todos los caminos de un término, el camino con mayor votación es elegido como el significado desambiguado del término.

Para realizar las pruebas del método de desambiguación propuesto hemos tomado los términos del tesauro GEMET y los hemos buscado en *WordNet*. Analizando la complejidad del tesauro hemos observado que 5070 de sus 5542 términos son términos compuestos por varias palabras; que el 15 % de las palabras no se han encontrado en *WordNet* (en este primer análisis no se ha tenido en cuenta si eran adjetivos y conjeturamos que la mayoría de los términos estaban formados por un adjetivo y varios nombres); y tan sólo el 24 % de los términos son polisémicos.

Con estos datos se deduce que la probabilidad de asociar aleatoriamente, es decir sin utilizar ninguna información, el significado *WordNet* más adecuado a una palabra GEMET es 0.217. Por tanto nuestro método de desambiguación debe mejorar significativamente este valor acercándose lo máximo posible a 1.

Aplicando nuestro método a una rama del tesauro GEMET, en concreto a la rama `administration` (en la que únicamente el 10 % de los términos son monosémicos y sólo un 2.8 % de las palabras no se encuentran en *WordNet*) hemos obtenido un nivel de acierto al desambiguar de 0.77. A la vista de este resultado y analizando con mayor detenimiento el método de votación, se observan tres anomalías que influyen en la valoración final:

1. Los conceptos de *WordNet* que están en niveles más inferiores tienen rutas más largas y, por tanto, coincidirán en más subjerarquías y serán más votados. Se puede solucionar dividiendo el valor de la votación por la profundidad del concepto en la jerarquía de *WordNet*.

2. No conviene ponderar igual todas las palabras del contexto porque el método de votación permite que significados erróneos sean votados por significados correctos de términos muy alejados en el tesauro. Para ello, se da más importancia a las coincidencias con palabras más cercanas en la estructura de GEMET dividiendo el valor de la votación por la distancia (en el sentido de la teoría de grafos) entre

los términos en el tesauro GEMET.

3. Las palabras del contexto más polisémicas votan más veces porque votan con cada uno de sus significados, la mayoría de ellos erróneos. Se soluciona dividiendo el valor de la votación por el número de polisemias.

En cierto sentido, estos tres criterios de ponderación están relacionados con los aspectos que Aguirre y Rigau usan en [AR96] para definir la *distancia conceptual*, tienen en cuenta la longitud del camino entre conceptos en *WordNet*, así como la profundidad y la densidad de los conceptos en la jerarquía. Al igual que el método de Aguirre y Rigau, nuestro método no necesita un corpus de entrenamiento para estimar probabilidades, necesario por ejemplo en [Res95] para estimar la similaridad semántica, sino que aprovecha el hecho de que GEMET es un tesauro con una estructura jerárquica para valorar de forma distinta la información que proporcionan las palabras del contexto según su posición en el tesauro.

Teniendo en cuenta estos tres criterios se obtiene un acierto del 81 % en la rama `administration` y se supera el 90 % de acierto en el significados de palabras de otras ramas con términos menos polisémicos.

Como aplicación de este método, se ha construido una herramienta útil y sencilla para importar y exportar tesauros de distintas organizaciones, que puede modificar tanto la estructura jerárquica como el resto de la información contenida en los tesauros. Durante el proceso de importación se realiza la asignación automática de conceptos y la expansión de claves que permiten la búsqueda de información geoespacial utilizando palabras clave de distintos tesauros. Su utilización en el proyecto ISIGIS, que tiene por objetivo desarrollar una infraestructura nacional de información, ha demostrado la viabilidad y utilidad de la estrategia planteada para la desambiguación semántica o asignación automática de conceptos. El método de expansión de claves clasifica, siempre que es posible (ya que los tesauros tratan ámbitos restringidos), cada metadato para el nuevo tesauro, añadiendo también los sinónimos en las clasificaciones para que las búsquedas sean más completas.

En particular, la anterior tecnología define una noción clara de "similaridad" entre geodatos extraídos de fuentes heterogéneas. Así, para aplicar con eficiencia el servicio de coordinación WCS en el contexto de los Sistemas Basados en la Localización, es posible reemplazar el *matching literal* de Linda, por el *matching semántico* basado en la alineación por medio de la distancia conceptual entre términos que aparecen en listas controladas o en tesauros.

4.2. Segundo ejemplo de aplicación: *matching con restricciones*

En este apartado retomamos de nuevo nuestro modelo formal, que dejamos al final del anterior capítulo. Esto es así pues en esta ocasión, el *matching complejo* no está basado en una aplicación pragmática de tecnología, sino en un enriquecimiento de la capacidad de consulta dentro del modelo estructurado.

Una vez que disponemos de tuplas estructuradas, podemos extender la potencia expresiva de la coordinación entre procesos por medio de una generalización en la sintaxis de las plantillas. Dicha generalización consiste en dotar a las plantillas de un lenguaje de consulta que permita definir restricciones más complejas que la simple concordancia de valor o el uso de una variable libre. De esta forma se podrán obtener tuplas del espacio de tuplas que se ajusten a criterios más complejos. En esta variante del modelo, se mantiene la misma estructura para las tuplas, pero en las plantillas cada atributo lleva asociada una restricción sobre su valor.

Manteniendo las notaciones del apartado 3.3, supondremos que existe una relación de orden total en cada dominio D_i, de forma que se podrán aplicar los operadores relacionales $(=, =, <, >, \ldots)$ entre valores de un mismo dominio. En aquellos dominios no numéricos en los que no haya una relación de orden natural, siempre se podrá utilizar el orden lexicográfico. La existencia de una relación de orden es necesaria para poder incorporar criterios de búsqueda de tuplas que sean compuestos (por ejemplo, buscar una tupla que el atributo `hora-salida` tenga un valor comprendido entre las 9 y las 11 horas).

Introducimos en la siguiente definición nuestra noción de *restricción*.

Definición 4.1 (**Restricción coherente**)**.** Dado un conjunto de atributos $A = A_1, \ldots, A_n$, decimos que la restricción r_i, asociada a un atributo A_i, es **coherente** respecto al conjunto de atributos A si:

1. r_i es una expresión que respeta la sintaxis BNF fijada en el cuadro 4.1.
2. Las únicas variables libres que aparecen en la restricción son los nombres de dichos atributos y, además la restricción contiene, como mínimo, a la variable libre correspondiente al atributo A_i.

Definición 4.2 (**Plantilla_estructurada con restricciones**)**.** Sea $A = A_1, \ldots, A_m$ una lista de atributos con dominios de valores $D_i = dom(A_i)$ y dominios de restricciones coherentes $R_i = rest(A_i)$, $i = 1, \ldots, m$. Utilizaremos la notación $R = rest(A) = \bigcup_{i=1}^{m} rest(A_i) = \bigcup_{i=1}^{m} R_i$ para indicar el conjunto de todas las posibles restricciones coherentes con respecto al conjunto de atributos A.

Una plantilla_estructurada con restricciones $\mathbf{templ_{rest}}$ sobre A es una aplicación parcial e inyectiva:

$$\mathbf{templ_{rest}} : \quad X \subseteq A - \quad X \times rest(X) \tag{4.1}$$

array	::=	una secuencia de caracteres
integer	::=	un número entero válido
float	::=	un número real válido
número	::=	*integer* *float*
constante	::=	*número* *array*
variable	::=	*array*
restricción	::=	*restricción_simple* *restricción_compuesta*
restricción_simple	::=	*constante* *constante* ? *variable*
restricción_compuesta	::=	? *variable* &(*condición*)
condición	::=	*condición_simple* *condición_compuesta*
condición_compuesta	::=	*condición* & *condición* *condición* *condición*
condición_simple	::=	*átomo* *átomo*
átomo	::=	(*predicado* *término* $^+$)
término	::=	*constante* ? *variable* (*función* *término* $^+$)
predicado	::=	$<$ $>$ $\leq$ $\geq$ $=$ $=$ *array*
función	::=	$+$ $-$ $*$ $/$ *array*

Cuadro 4.1: Sintaxis BNF para las restricciones

tal que A_i X $templ_{rest}(A_i) = (A_i, r_i)$, donde r_i es una restricción *coherente* respecto al conjunto de atributos de referencia A.

Por tanto, una plantilla_estructurada con restricciones es un conjunto finito de pares *atributo/restricción* de la forma $templ_{rest} = ((a_1, r_1) \ldots (a_m, r_m))$, donde cada r_i es una restricción coherente.

Nota. La terminología BNF utilizada para describir la sintaxis de las plantillas_estructuradas con restricciones es bastante común en los manuales de referencia de computación.

- Las palabras o caracteres "planos", particularmente los paréntesis, se escriben exactamente como aparecen. Las secuencias de palabras encerradas entre " ", tal como *atributo* , representan una entidad individual, denominados símbolos no terminales.

- Un símbolo no terminal seguido por un $*$, representa cero o más ocurrencias de este elemento. Un símbolo no terminal seguido por un + representa una o más apariciones del elemento. Un $*$ o un + como nombre de función representan el producto y la suma respectivamente. En otro caso se representan a sí mismos.

- La barra vertical indica una opción entre múltiples elementos. Los espacios en blanco se utilizan como delimitadores entre elementos.

- El símbolo ::= se utiliza para definir símbolos no terminales, indicando cómo pueden ser reemplazados. Por ejemplo:

::=

- Un array es cualquier secuencia de caracteres que comienza por un carácter (ASCII imprimible) y le siguen 0 o más caracteres (ASCII imprimibles). El array termina con un delimitador, que no forma parte del mismo. Los caracteres que actúan como delimitadores son: espacio en blanco, dobles comillas, paréntesis, &, , y ?.

- Una función es una porción de código ejecutable identificado por un nombre y que devuelve un valor. Para llamar a una función usaremos la notación prefija. Las funciones permiten obtener el valor de un atributo ejecutando una porción de código. La llamada a la función comienza con abrir paréntesis, seguido del nombre de la función y, a continuación, los términos separados por espacios en blanco. Termina con cerrar paréntesis.

- Un predicado es una función que devuelve el valor *verdad* o *falso*. Los predicados permiten restringir el valor de un campo basándose en la veracidad de una expresión booleana dada.

Es obvio que esta definición de plantilla_estructurada con restricciones es más general que la definición dada en el modelo *estructurado*, puesto que toda plantilla_estructurada se puede representar como un plantilla_estructurada con restricciones. La definición de plantilla_valor sigue siendo válida, teniendo en cuenta que ahora la función valor devuelve una tupla_core cuyos campos son las restricciones de la plantilla_estructurada origen. Los conceptos de esquema y canal siguen siendo igualmente válidos.

Ejemplo 4.3. ***Ejemplos de tuplas y plantillas con restricciones***

Tupla en el modelo con restricciones

Las tuplas en el modelo con restricciones tienen el mismo formato que las tuplas_estructuradas.

((tipo vuelo) (origen Madrid) (destino Londres) (salida 10:30) (llegada 12:30))

Plantilla sin variables en el modelo con restricciones

La anterior tupla también puede ser considerada una plantilla sin variables. Además, en el modelo con restricciones, pueden aparecer otras plantillas sin variables, ya que mediante el símbolo podemos negar un valor. Por ejemplo, la siguiente plantilla representa cualquier vuelo que no tenga destino Londres.

((tipo vuelo) (origen Madrid) (destino Londres) (salida 10:30) (llegada 12:30))

Plantilla con variables en el modelo con restricciones, sin restricciones

Si las variables no tienen asociada ninguna restricción se mantiene la sintaxis del modelo estructurado. En este caso no se puede utilizar cualquier nombre para las variables, sino que cada variable se llamará igual que "su" atributo respectivo (precedido por el símbolo ?).

((tipo vuelo) (origen Madrid) (destino Londres) (salida ?salida) (llegada ?llegada)

Plantilla con variables en el modelo con restricciones, con restricciones

Si la variable tiene asociada alguna restricción, ésta se introduce con el símbolo & seguido de un predicado. En la condición debe aparecer dicha variable, pero también pueden aparecer otras variables que contengan valores de otros campos de la tupla.

El predicado < se utiliza en este ejemplo para localizar vuelos de Madrid a Londres, que lleguen a su destino antes de las 12 horas.

((tipo vuelo) (origen Madrid) (destino Londres) (salida ?salida)
(llegada ?llegada&(< ?llegada 12:00)))

Plantilla con conectores

En este ejemplo se hace uso del conector para seleccionar vuelos que partan de Madrid con destino Londres o Dublín.

((tipo vuelo) (origen Madrid) (destino ?destino&((= ?destino Londres) (= ?destino Dublín))
(salida ?salida) (llegada ?llegada))

Con el conector &, combinado con el operador =, podemos expresar la búsqueda de cualquier vuelo, con origen Madrid, que no tenga como destino París ni Roma.

((tipo vuelo) (origen Madrid) (destino ?destino&((= París)&(= Roma))
(salida ?salida) (llegada (?llegada))

Esta plantilla selecciona vuelos cuya hora de salida sea entre las 9 y las 12 horas.

((tipo vuelo) (origen Madrid) (destino Londres) (salida ?salida&((> ?salida 9:00) &
(< ?salida 12:00))) (llegada ?llegada))

Plantilla con funciones en el modelo con restricciones

La siguiente plantilla utiliza la función + para expresar que se quieren localizar vuelos cuya hora de llegada sea 2 horas posterior a la hora de salida. En este ejemplo la restricción asociada al atributo `llegada` *contiene una variable de otro campo, en este caso la variable* `?salida`*, cuyo valor se obtiene de la restricción anterior.*

((tipo vuelo) (origen Madrid) (destino Londres) (salida ?salida))
(llegada ?llegada&(= (+ ?salida 2)))

Ahora parece razonable pensar que para que una plantilla y una tupla se correspondan sea necesario que todas y cada una de las restricciones de la plantilla sean satisfechas por la tupla. La siguiente extensión de la definición del predicado de correspondencia incluye esta consideración.

Definición 4.4 (Predicado de correspondencia_estructurado con restricciones). Sea t_{rest} una tupla sobre $X \subseteq A$, $X = A_1, \ldots, A_n$ y sea $templ_{rest} = ((a_1, r_1) \ldots (a_m, r_m))$ una plantilla sobre $X' \subseteq A$, $X' = A'_1, \ldots, A'_m$, donde $r_1, \ldots, r_m$ son restricciones asociadas a los dominios $A'_1, \ldots, A'_m$ respectivamente.

Se dice que la tupla t_{rest} se **corresponde o empareja** con $templ_{rest}$, y se indica mediante el predicado $\mathbf{mtc_{rest}}(\mathbf{templ_{rest}}, \mathbf{t_{rest}})$, si se cumplen las siguientes condiciones:

1. Tienen la misma aridad, es decir $m = n$.
2. Están definidas sobre el mismo conjunto de atributos, es decir $X = X'$.

3. Todas y cada una de las restricciones son satisfechas por la tupla.

Las dos primeras condiciones son las mismas que se exigen en el modelo *estructurado*. La tercera condición exige el cumplimiento de que cada una de las restricciones de la plantilla por parte de los valores particulares de la tupla. La siguiente definición formaliza esta condición.

Definición 4.5 (Restricción satisfecha por una tupla). Sea $t_{rest} = ((a_1, v_1) \ldots (a_n, v_n))$ una tupla sobre $X \subseteq A$, $X = A_1, \ldots, A_n$ y sea r_i una restricción asociada al dominio A_i.

Se dice que la restricción r_i **es satisfecha** por la tupla t_{rest}, si la función `restricción_satisfecha?`, aplicada al par (r_i, t_{rest}), devuelve el valor *verdad*.

La definición de la función `restricción_satisfecha?` será una definición recursiva sobre la definición BNF de la restricción.

Function `restricción_satisfecha?`(*r:restricción; t:tupla*):booleano

```
if r es ? variable then
    return verdad;
else if r es constante then
    return constante =v_i;
else if r es ? variable & condición then
    return evaluar_condición (condición,t);
end
```

Function `evaluar_condición`(*c:condición; t:tupla*):booleano

```
if c es condición_simple then
    if c es átomo then
        return evaluar_predicado (predicado(c),términos(c));
    else if c es átomo then
        return NOT evaluar_predicado (predicado(c),términos(c));
    end
else if c es condición_compuesta then
    if c es c1&c2 then
        return evaluar_condición (c1,t) AND evaluar_condición (c2,t);
    else if c es c1 c2 then
        return evaluar_condición (c1,t) OR evaluar_condición (c2,t);
    end
end
```

Definición 4.6 (Función evaluar_predicado). La función `evaluar_predicado` se aplica a un predicado con una lista de términos de la siguiente manera:

- Se resuelve la evaluación de cada uno de los términos.

- Se resuelve el predicado aplicando la semántica usual asociada a los operadores lógicos ($=$, $=$, $<$, $>$, $\leq$, $\geq$,...) devolviendo un valor booleano. En caso de tratarse de un predicado definido por el usuario, se aplicará el código asociado al predicado.

- Se supone que las funciones y operadores lógicos están bien definidos en el conjunto de atributos de la tupla.

Formalmente, la evaluación de un término será una función que, aplicada a un término, devuelva un valor del conjunto de valores del conjunto de atributos correspondientes a la tupla.

Function `evaluar_término`(*r:restricción; t:tupla*):booleano

if *término es constante* **then**
  return *constante* ;
else if *término es ? variable* **then**
  return valor de la tupla correspondiente a ese atributo;
else if *término es función* **then**
  return valor de aplicar la función a la evaluación de cada uno de sus argumentos;
end

Observación. *Como era de esperar, el predicado de correspondencia_estructurado con restricciones es una extensión del predicado de correspondencia estructurado. Por una parte, cualquier plantilla_estructurada puede expresarse como una plantilla con restricciones. Por otra, la definición de la función* `restricción_satisfecha?` *es una extensión del procedimiento de reconocimiento de patrones en el que se basa el mecanismo de correspondencia del modelo* estructurado*. Recordar que en el modelo* estructurado *sólo se exige la igualdad de valores en aquellos campos de la plantilla que explícitamente contengan algún valor y se permite cualquier valor en la tupla cuando en el correspondiente campo de la plantilla aparezca una variable.*

Las anteriores definiciones encadenadas nos han permitido definir un nuevo ejemplo de *matching complejo*, el *emparejamiento con restricciones*. Puesto que se trata, al igual que en el caso del *matching semántico*, de una particularización del modelo Linda, la definición del predicado de selección (al igual que la dinámica del sistema de transición) no debe ser modificada: bastará con adoptar como predicado de elección cualquiera que sea *compatible* (en el sentido de Linda) con el predicado de *matching*. Esta idea es explícitamente recogida en la última definición, que cierra este apartado y el capítulo.

Definición 4.7 (Predicado de elección_estructurado con restricciones). Dado un predicado de correspondencia $mtc_{rest}(templ_{rest}, t_{rest})$ definido sobre $Templ_{rest} \times T_{rest}$, llamamos **predicado de elección_estructurado con restricciones**, denotado por $\mu_{\mathbf{rest}}(\mathbf{templ_{rest}}, \overline{\mathbf{t}}_{\mathbf{rest}}, \widehat{\mathbf{t}}_{\mathbf{rest}})$, a cualquier predicado que satisfaga las reglas de correspondencia de Linda definidas en (3.1).

$$\frac{mtc_{rest}(templ_{rest}, t_{rest})}{\mu_{rest}(templ_{rest}, t_{rest}\ \overline{t}_{rest}, t_{rest})} \qquad \frac{\nexists\ t \quad \overline{t}_{rest}\ :\ mtc_{rest}(templ_{rest}, t_{rest})}{\mu_{rest}(templ_{rest}, \overline{t}_{rest}, \quad)}$$

Es decir, en caso de existir alguna, $\widehat{t}_{rest}$ es una tupla cualquiera del multiconjunto $\overline{t}_{rest}$ que se corresponde con la plantilla $templ_{rest}$, o bien es un valor nulo (indicado por el símbolo) si $\overline{t}_{rest}$ no contiene dicha tupla.

Capítulo 5

Extensión del modelo formal

En este capítulo extendemos el modelo formal para dotarlo de capacidad transaccional. Las ideas iniciales de este trabajo están publicadas en [MÁBR04], [MÁBR07a] y [MÁBR07b].

Con el objetivo de utilizar el modelo Linda en un entorno basado en la Web, en el capítulo 3 hemos trabajado con una versión de Linda donde las tuplas y las plantillas admitían descripciones como secuencias de pares *atributo/valor*, similar a la estructura de un documento XML simple. En la sección 4.2 hemos extendido la capacidad expresiva del modelo enriqueciendo la sintaxis de las plantillas. El uso de un lenguaje de consulta como el definido en el modelo *con restricciones* permite construir patrones de búsqueda más complejos que en el caso del modelo *estructurado*. Este lenguaje permite definir, además de restricciones que relacionen valores de los atributos en una misma plantilla, relaciones entre atributos de varias plantillas. De esta forma, estaríamos ante un nuevo modelo que incorporaría operaciones que involucran a más de una tupla. Este mecanismo de correspondencia múltiple proporciona una aproximación prometedora para que los servicios Web se puedan comunicar, coordinar y colaborar en la red de un modo transaccional.

5.1. Modelo con correspondencia múltiple

El plan para definir un nuevo modelo que extienda Linda es el siguiente. En primer lugar, definiremos las reglas semánticas que ligarán los predicados de correspondencia y de elección. A continuación, para trabajar con la misma granularidad que en capítulos anteriores, definiremos un predicado concreto de correspondencia. Para ello, retomaremos las notaciones del apartado 4.2, aunque con el fin de simplificar la notación y facilitar la lectura, omitiremos el sufijo *rest* en los nombres de tuplas y plantillas. Otra nueva característica será que supondremos que cada plantilla tiene un nombre, pues dicho nombre será necesario para identificar sus componentes cuando aparezcan en las restricciones de otras plantillas.

Una vez realizadas esas definiciones básicas, daremos la descripción dinámica del sistema. Para ello, volveremos a las técnicas y notaciones del apartado 3.1.1 (en el que se describió formalmente la dinámica de Linda, a través de un sistema de transición). Manteniendo las mismas definiciones para el sistema de transición, nos concentraremos aquí en definir el predicado de satisfacción S y la función de evaluación E. Esto terminará la definición formal de nuestro nuevo **modelo con correspondencia múltiple**. El resto del capítulo será dedicado a ilustrar la potencialidad del modelo y a discutir cuestiones de implementación y verificación.

Introducimos las definiciones en el caso de consulta múltiple de *dos* plantillas. El caso general, que es una simple generalización pero con notación más engorrosa, será brevemente explorado al final de este apartado.

Definición 5.1 (**Reglas de correspondencia múltiple. Caso de dos plantillas**). Sean t_1, t_2 dos tuplas definidas sobre $X_1 \subseteq A$, con $X_1 = \ A_{11}, \ldots, A_{1n}\ $ y $X_2 \subseteq A$, con $X_2 = \ A_{21}, \ldots, A_{2m}\ $ respectivamente, y sean $templ_1$, $templ_2$ dos plantillas definidas sobre los mismos dominios respectivos.

Dados un predicado de correspondencia $mtc_2(templ_1, templ_2, t_1, t_2)$ y un predicado de elección múltiple $\mu_2(templ_1, templ_2, \overline{t}, \widehat{t_1\ t_2})$, se definen las siguientes reglas de correspondencia:

$$\frac{mtc_2(templ_1, templ_2, t_1, t_2)}{\mu_2(templ_1, templ_2, t_1\ t_2\ \overline{t}, t_1\ t_2)}$$

$$\frac{\nexists\ t_1 \quad \overline{t} \quad or \quad \nexists\ t_2 \quad \overline{t}\ :\ mtc_2(templ_1, templ_2, t_1, t_2)}{\mu_2(templ_1, templ_2, \overline{t}, \ \overset{2}{T})}$$

En caso de existir, $\widehat{t_1\ t_2}$ serán dos tuplas del multiconjunto $\overline{t}$ que se corresponden con las plantillas $templ_1$ y $templ_2$, es decir, que satisfacen el predicado $mtc_2(templ_1, templ_2, t_1, t_2)$, o bien es un valor nulo (indicado por el símbolo $\overset{2}{T} = \ T, \quad T\ $) si $\overline{t}$ no contiene ambas tuplas.

La anterior definición recoge la semántica *estática* del nuevo modelo formal que estamos introduciendo. Particularizamos ahora el predicado de correspondencia, para volver al grado de precisión que hemos empleado en nuestros modelos *estructurado* y con *restricciones*. Las notaciones y convenios son iguales al caso con restricciones, salvo que, además, se permite que las restricciones de las plantillas contengan variables correspondientes al conjunto de atributos totales $X = X_1 \bigcup X_2$. Los nombres de las variables que se correspondan con el nombre de un atributo tendrán como prefijo el nombre de la plantilla (rompiendo de ese modo posibles ambigüedades). Asumimos, tal y como introdujimos en el caso con restricciones, que los predicados, funciones y la función de evaluación están bien definidos (son *coherentes*) en el conjunto total de atributos. Es decir, la evaluación de un término se hace con respecto a los valores correspondientes de los atributos de las tuplas involucradas.

Definición 5.2 (**Predicado de correspondencia múltiple. Caso de dos plantillas**). Sean t_1, t_2 dos tuplas y sean $templ_1$, $templ_2$ dos plantillas, como en la definición anterior. Se dice que las dos tuplas t_1 y t_2 se **corresponden o emparejan** con las dos plantillas $templ_1$ y $templ_2$, respectivamente, y se indica mediante el predicado $\mathbf{mtc_2(templ_1, templ_2, t_1, t_2)}$, si se cumplen las siguientes condiciones:

1. La plantilla $templ_1$ y la tupla t_1 tienen la misma longitud y están definidas sobre el mismo conjunto de atributos.
2. La plantilla $templ_2$ y la tupla t_2 tienen la misma longitud y están definidas sobre el mismo conjunto de atributos.
3. Todas las restricciones de las plantillas $templ_1$ y $templ_2$ son coherentes respecto al conjunto total de atributos $X = X_1 \bigcup X_2$.
4. Cada una de las restricciones de las plantillas $templ_1$ y $templ_2$ son satisfechas por los valores de las dos tuplas.

En las siguientes dos proposiciones, mtc denota el predicado de correspondencia con restricciones del apartado 4.2 y mtc_2 el predicado que acabamos de introducir, mientras que μ y μ_2 son cualesquiera predicados de elección satisfaciendo la semántica de los modelos Linda y nuestro modelo de correspondencia múltiple, respectivamente.

Proposición 5.3. *El predicado mtc es un caso particular del predicado* mtc_2. *Más concretamente,* $mtc_2(templ, templ, t, t)$ *si y sólo si* $mtc(templ, t)$.

Demostración.

—

Si $mtc_2(templ, templ, t, t)$ es verdad, por la definición 5.2 se cumple que:

- $templ$ y t tienen la misma longitud y están definidas sobre el mismo conjunto de atributos.
- Todas las restricciones de $templ$ son coherentes respecto al conjunto de atributos.
- Cada una de las restricciones de $templ$ son satisfechas por los valores de t.

Por tanto, por la definición 3.9 se cumple $mtc(templ, t)$.

—

Si se cumple $mtc(templ, t)$, por definición 3.9 se cumplen las condiciones de la definición 5.2 y, por tanto, se cumple $mtc_2(templ, templ, t, t)$. ■

Proposición 5.4. *Si se cumple* $\mu_2(templ, templ, \overline{t}, \widehat{t\ t})$ *entonces se cumple* $\mu(templ, \overline{t}, \widehat{t})$.

Demostración. Inmediato por la definición de μ y μ_2. ■

Observemos, sin embargo, que la implicación contraria no siempre es cierta. Para que se cumpla es necesario que existan dos ocurrencias de la tupla t en el multiconjunto $\overline{t}$. En caso de existir una única ocurrencia de t, se cumplirá:

$$\mu(templ, \overline{t}, \widehat{t})$$

pero

$$\mu_2(templ, templ, \overline{t}, \quad {}^{2}_{T})$$

Procedemos a continuación a definir la semántica de nuestro modelo con correspondencia múltiple. Para ello, como indicamos anteriormente, bastará con ir presentando, para cada operación que introduzcamos, las reglas de definición del predicado de satisfacibilidad S y de la función de evaluación E. Las enumeramos a continuación, haciendo breves comentarios sobre su significado en la práctica. Consideramos que no requieren mayor explicación debido a que, como no podía ser de otro modo, están diseñadas tomando como patrón la semántica del modelo Linda presentado en el capítulo 3.

5.1.1. La operación de lectura bloqueante múltiple $rd_2()$

Para S:

$$\frac{mtc_2(templ_1, templ_2, t_1, t_2)}{S(rd_2(templ_1, templ_2)^o, t_1\ t_2\ \overline{t})}$$

Para E:

$$\frac{mtc_2(templ_1, templ_2, t_1, t_2)}{t_1\ t_2\ \overline{t}, \overline{w}, \underline{o}t_1\ \underline{o}t_2, t_1\ t_2\ \overline{t}, \overline{w} \qquad E(rd_2(templ_1, templ_2)^o)}$$

Es decir, devuelve las tuplas t_1 y t_2 y no modifica el espacio de tuplas. Bloquea al proceso que la invoca hasta que existan ambas tuplas.

5.1.2. La operación de lectura no bloqueante múltiple $rdp_2()$

Para S:

$$S(rdp_2(templ_1, templ_2)^o, \overline{t})$$

Para E:

$$\frac{\mu_2(templ_1, templ_2, \overline{t}, \widehat{t_1\ t_2})}{\overline{t}, \overline{w}, \underline{o}\widehat{t_1\ t_2}, \overline{t}, \overline{w} \qquad E(rdp_2(templ_1, templ_2)^o)}$$

Es decir, devuelve $\widehat{t_1\ t_2}$ y no modifica el espacio de tuplas. No bloquea al proceso que la invoca. Si no existen las dos tuplas entonces devuelve el valor $\overset{2}{T}$.

5.1.3. La operación de extracción bloqueante múltiple $in_2()$

Para S:

$$\frac{mtc_2(templ_1, templ_2, t_1, t_2)}{S(in_2(templ_1, templ_2)^o, t_1\ t_2\ \overline{t})}$$

Para E:

$$\frac{mtc_2(templ_1, templ_2, t_1, t_2)}{t_1\ t_2\ \overline{t}, \overline{w}, \underline{o}t_1\ \underline{o}t_2, \overline{t}, \overline{w} \qquad E(in_2(templ_1, templ_2)^o)}$$

Es decir, devuelve las tuplas t_1 y t_2 y las elimina del espacio de tuplas. Bloquea al proceso que la invoca hasta que existan ambas tuplas.

5.1.4. La operación de extracción no bloqueante múltiple $inp_2()$

Para S:

$$S(inp_2(templ_1, templ_2)^o, \overline{t})$$

Para E:

$$\frac{\mu_2(templ_1, templ_2, \overline{t}, \widehat{t_1\ t_2})}{\overline{t}, \overline{w}, \underline{o}\widehat{t_1\ t_2}, \overline{t}\ \widehat{t_1\ t_2}, \overline{w} \qquad E(inp_2(templ_1, templ_2)^o)}$$

Es decir, devuelve $\widehat{t_1\ t_2}$ y elimina las tuplas del espacio de tuplas. No bloquea al proceso que la invoca. Si no existen las dos tuplas entonces devuelve el valor $\overset{2}{T}$.

5.1.5. La operación mixta bloqueante múltiple $rd.in()$

Para S:

$$\frac{mtc_2(templ_1, templ_2, t_1, t_2)}{S(rd.in(templ_1, templ_2)^o, t_1\ t_2\ \overline{t})}$$

Para E:

$$\frac{mtc_2(templ_1, templ_2, t_1, t_2)}{t_1\ t_2\ \overline{t}, \overline{w}, \underline{o}t_1\ \underline{o}t_2, t_1\ \overline{t}, \overline{w} \qquad E(rd.in(templ_1, templ_2)^o)}$$

Es decir, devuelve las tuplas t_1 y t_2 y elimina t_2 del espacio de tuplas. Bloquea al proceso que la invoca hasta que existan ambas tuplas.

5.1.6. La operación mixta no bloqueante múltiple *rdp.inp*()

Para S:

$$S(rdp.inp(templ_1, templ_2)^o, \overline{t})$$

Para E:

$$\frac{\mu_2(templ_1, templ_2, \overline{t}, \widehat{t_1\ t_2})}{\overline{t}, \overline{w}, \underline{o}\widehat{t_1\ t_2}, \overline{t}\ \widehat{t_2}, \overline{w} \qquad E(rdp.inp(templ_1, templ_2)^o)}$$

Es decir, devuelve $\widehat{t_1\ t_2}$ y elimina la tupla $\widehat{t_2}$ del espacio de tuplas. No bloquea al proceso que la invoca. Si no existen ambas tuplas entonces $\widehat{t_2}$ es el valor $_T$ y la operación devuelve el valor ^2_T.

5.1.7. Generalización al caso de más de dos tuplas

Como indicamos anteriormente, la generalización a múltiples tuplas es evidente. La notación es ligeramente más farragosa, pues hay que emplear una notación vectorial para definir secuencias de tuplas y plantillas, y hay que utilizar índices para representar convenientemente las reglas semánticas.

Sin detenernos en detalles notacionales, damos aquí las fórmulas de la *operación mixta bloqueante múltiple* $rd_{n_1}in_{n_2}()$.

Para S:

$$\frac{mtc_{n_1+n_2}(\overline{\overline{templ}}, \overline{t}\ )}{S(rd_{n_1}in_{n_2}(\overline{\overline{templ}})^o, t_1\ \ldots\ t_{n_1+n_2}\ \overline{t})}$$

Para E:

$$\frac{mtc_{n_1+n_2}(\overline{\overline{templ}}, \overline{t}\ )}{t_1\ \ldots\ t_{n_1+n_2}\ \overline{t}, \overline{w}, \underline{o}t_1\ \ldots\ \underline{o}t_{n_1+n_2}, t_1\ \ldots\ t_{n_1}\ \overline{t}, \overline{w} \qquad E(rd_{n_1}in_{n_2}(\overline{\overline{templ}})^o)}$$

Es decir, devuelve las tuplas $\overline{t} = t_1, \ldots, t_{n_1+n_2}$ y elimina las tuplas $t_{n_1+1}, \ldots, t_{n_1+n_2}$ del espacio de tuplas. Bloquea al proceso que la invoca hasta que existan todas las tuplas.

Obsérvese que los casos $n_1 = 0$ o $n_2 = 0$ cubren, respectivamente, las operaciones de extracción bloqueante múltiple o de lectura bloqueante múltiple. De modo análogo, en un único par de fórmulas podemos cubrir todas las operaciones no bloqueantes.

Concretamente, la *operación mixta no bloqueante múltiple* $rdp_{n_1}.inp_{n_2}()$ queda definida del siguiente modo.

Para S:

$$S(rdp_{n_1}.inp_{n_2}(\overline{templ})^o, \overline{t})$$

Para E:

$$\frac{\mu_{n_1+n_2}(\overline{templ}, \overline{t}, t_1 \ \widehat{\ldots \ t_{n_1+n_2}})}{\overline{t}, \overline{w}, \underline{o}t_1 \ \widehat{\ldots \ t_{n_1+n_2}}, \overline{t} \ t_{n_1+1}, \widehat{\ldots, t_{n_1+n_2}}, \overline{w} \qquad E(rdp_{n_1}.inp_{n_2}(\overline{templ})^o)}$$

Es decir, devuelve $t_1 \ \widehat{\ldots \ t_{n_1+n_2}}$ y elimina las tuplas $t_{n_1+1}, \ldots, t_{n_1+n_2}$ del espacio de tuplas. No bloquea al proceso que la invoca. Si no existen las tuplas entonces $t_1 \ \widehat{\ldots \ t_{n_1+n_2}}$ es el valor $\overset{n_1+n_2}{T}$ y la operación devuelve el valor $\overset{n_1+n_2}{T}$.

5.2. Aspectos transaccionales del modelo Linda

Una de las principales limitaciones del modelo básico de Linda, señaladas por varios autores (véase, por ejemplo, [AS92] y [BCGZ01]) es la ausencia de capacidad para realizar *transacciones*. El concepto de transacción, tomado prestado del ámbito de las Bases de Datos, se refiere a la necesidad de asegurar que una secuencia de operaciones se realice completamente o bien no se realice en absoluto. Cuando las operaciones tienen efectos laterales (como la modificación de una base de datos) el conseguir ese efecto tiene dificultades técnicas evidentes. Usualmente se solicita a una secuencia de operaciones, para ser considerada una transacción, que respete las propiedades denominadas ACID (la "I" proviene del inglés *isolation*):

Atomicidad O se ejecutan todas las operaciones involucradas en la transacción o no se ejecuta ninguna.

Consistencia La realización de una transacción debe dejar el sistema en un estado consistente.

Aislamiento El progreso de una transacción no debe afectar a la realización de otras transacciones en curso.

Durabilidad Una vez que se confirma una transacción sus efectos perdurarán y solo podrán ser modificados por la realización de otra transacción.

En relación con Linda, hay que tener presente que cada operación básica (ya sea de lectura o de escritura) puede ya ser vista como una operación compuesta (puesto que una tupla es un dato estructurado). Dependiendo de la implementación podría suceder que por un malfuncionamiento del sistema de tuplas o por un problema de concurrencia, una operación de, por ejemplo, lectura lea y retire solo parcialmente una tupla, dejando el sistema en un estado inconsistente. Digamos que este comportamiento transaccional *mínimo* es asumido, sin mención explícita, por el modelo básico Linda. En *JavaSpaces*, que ya es una implementación concreta, cada vez que se inicia una transacción se activa un gestor de transacciones particular que se responsabiliza de que se cumplan las propiedades ACID.

Sin embargo, cuando los autores señalan la falta de transacciones en Linda se refieren a la realización de una secuencia de operaciones *básicas*. Una primera idea, explotada en [AS92], consiste en emular en Linda el modo de trabajo en Bases de Datos, extendiendo Linda con mecanismos sintácticos que permiten: dar comienzo a una transacción, terminarla, confirmarla (*commit*) o deshacerla (*rollback*). Sin embargo, cuando Linda es utilizado para realizar trabajos de coordinación con soporte en redes (como Internet), esta aproximación se muestra insuficiente. En general, muchos autores han señalado [Pri08], [LZH08] que mantener las propiedades ACID en un contexto de redes heterogéneas y en las que la duración de una transacción puede extenderse (de forma impredecible) en el tiempo, es excesivamente riguroso. La idea consiste en asociar a cada transacción una "transacción inversa" (denominada *compensación*), puesto que deshacer los efectos ya realizados (como en el clásico *rollback*) puede ser directamente imposible en un contexto como Internet [LZH08].

Una vez abandonada la generalización directa del tratamiento de la transaccionalidad en Bases de Datos (como se propuso en [AS92]), la atención se ha vuelto hacia otros modelos de transaccionalidad en lo que ésta está implícitamente incluida en los constructores mismos del lenguaje de procesamiento (véase [HMPH08]). La adaptación de esta idea al modelo Linda es realmente sencilla y natural, como ha sido destacado por diversos autores, por ejemplo en [BCGZ01]: mientras que las operaciones definidas en el modelo básico Linda son primitivas que operan con una única tupla, la transaccionalidad puede ser conseguida si añadimos primitivas que operen con más de una tupla.

En esta línea de añadir operaciones al modelo que involucren atómicamente a un multiconjunto de tuplas, TSpaces [WMLF98] proporciona la operación multiwrite que permite introducir atómicamente en el espacio de tuplas un vector de tuplas.

Otro ejemplo es la operación $\mathsf{rew}(\mathsf{m}_1, \mathsf{m}_2)$, definida en [BCGZ01]. Esta operación, inspirada en el lenguaje Gamma ([BM93]), consume atómicamente un multiconjunto de tuplas m_1 y produce un multiconjunto m_2. Formalmente se define como:

Para S:

$$\frac{mtc^m(templ^{m_1}, m_1)}{S(rew(m_1, m_2)^o, m_1\ \bar{t})}$$

Para E:

$$\frac{mtc^m(templ^{m_1}, m_1)}{m_1\ \overline{t}, \overline{w}, \underline{o}m_1, \overline{t}\ m_2, \overline{w} \qquad E(rew(m_1, m_2)^o)}$$

Desde el punto de vista de dotar de capacidad transaccional al modelo Linda, que la producción del multiconjunto m_2 sea atómica no aporta grandes ventajas. Sin embargo, la atomicidad en la ejecución de la retirada de las tuplas del multiconjunto m_1 sí que es importante. En [Zav00] se incluye la operación múltiple $\mathsf{min}(\mathsf{t_1}, \ldots, \mathsf{t_n})$, que consume atómicamente un multiconjunto de tuplas del espacio de tuplas. En ese trabajo se prueba que no es posible implementar la operación min utilizando las primitivas básica de Linda, llegando a la conclusión de que esta operación incrementa estrictamente la expresividad del modelo de coordinación básico.

La situación es muy similar a la que encontramos en nuestra propuesta de modelo con correspondencia múltiple. Por una parte, hemos aumentado la capacidad expresiva del modelo Linda: es claro que una operación de lectura secuencial de dos tuplas t_1 y t_2 no puede conseguirse en el modelo básico aunque en un instante dado ambas aparezcan en el espacio de tuplas, pues una operación concurrente que consuma t_2, tras el proceso de lectura de t_1, dejará bloqueada la operación de lectura inicial. Por otra, como veremos más adelante en ejemplos, nuestro nuevo modelo admite, a alto nivel conceptual, el diseño y realización de transacciones.

Las dos principales dificultades para que nuestro modelo tenga un interés real en la práctica son:

- dar una implementación eficiente y fiable de las operaciones con múltiples tuplas, y
- comprobar la adecuación de nuestra implementación para operar en entornos distribuidos y heterogéneos (basados en Internet, por supuesto).

En el resto de este capítulo nos ocupamos del primer aspecto. El segundo, que requiere tener al menos prototipos ejecutables y que seguramente necesitará de algún concepto de *compensación* [LZH08], queda como trabajo futuro.

5.3. El papel del algoritmo RETE para la correspondencia de múltiples tuplas

En este apartado retomamos una idea que ya apareció cuando introdujimos, en el capítulo anterior, el emparejamiento complejo con restricciones: al incluir ese tipo de consultas compuestas, las similitudes del modelo Linda con los sistemas de Bases de Datos y con los Sistemas Basados en Reglas (presentes desde la propuesta inicial del modelo), se hacen mucho más manifiestas. En el apartado precedente hemos evocado cómo la similitud con las Bases de Datos ha influido en el tratamiento que de las transacciones

en el modelo Linda se hizo en el pasado en la literatura (muy explítamente en [AS92], por ejemplo). Nosotros aquí, sin embargo, vamos a privilegiar el paralelismo con los Sistemas Basados en Reglas. Más concretamente, en esta sección presentamos el procedimiento de correspondencia atómica de múltiples tuplas, implícito en la operaciones de lectura y extracción, desde un punto de vista similar a los lenguajes basados en reglas. El proceso de reconocimiento de patrones es la mayor fuente de ineficiencia en los motores de inferencia dirigidos por patrones. Algoritmos como RETE [For82] o TREAT [Mir87] han sido desarrollados para mejorar la eficiencia de este proceso. La idea subyacente en estos algoritmos es mejorar la eficiencia emparejando con las reglas únicamente aquellos elementos de la memoria de trabajo que hayan sido modificados recientemente, en lugar de repetidamente emparejar todas las plantillas contra los datos. Con este propósito en mente, la información sobre los emparejamientos previos se almacena en una estructura de grafo, la red RETE.

Por ejemplo, dada la siguiente regla en CLIPS:

```
(defrule Regla-2
   (t1 (a ?x) (b rojo))
   (t2 (x ~verde) (y ?x))
   (t2 (x ?x) (y ?x))
=>)
```

el algoritmo RETE construye la red de la figura 5.1. Esta red contiene distintos tipos de nodos para representar las restricciones y las relaciones entre variables definidas en la regla. Además, los nodos llevarán asociadas unas memorias para almacenar los resultados parciales.

En nuestro caso, hay dos razones para basarnos en la estructura RETE a la hora de implementar un algoritmo para comprobar la correspondencia de múltiples tuplas:

- disponer de un algoritmo eficiente;
- usar la estructura RETE como ayuda para la semántica de las operaciones de lectura y extracción con varias plantillas.

Con este segundo fin, la porción de red asociada a cada operación de lectura o extracción múltiple recogerá, en la memoria asociada al nodo terminal, todas las tuplas que satisfagan las plantillas, y también los procesos bloqueados que estén esperando por tuplas que se correspondan con las plantillas.

Si existe en el espacio de tuplas un conjunto de tuplas que se emparejan con las plantillas de lectura, entonces el algoritmo las encontrará y las propagará hasta su correspondiente nodo terminal; en otro caso, el proceso lector se bloqueará hasta que se alcance el nodo terminal. En ambos casos, la extracción de todas las tuplas debe ser realizada como una acción atómica, es decir, mientras se resuelve la operación ninguna otra operación concurrente puede eliminar las tuplas que satisfacen la operación de lectura.

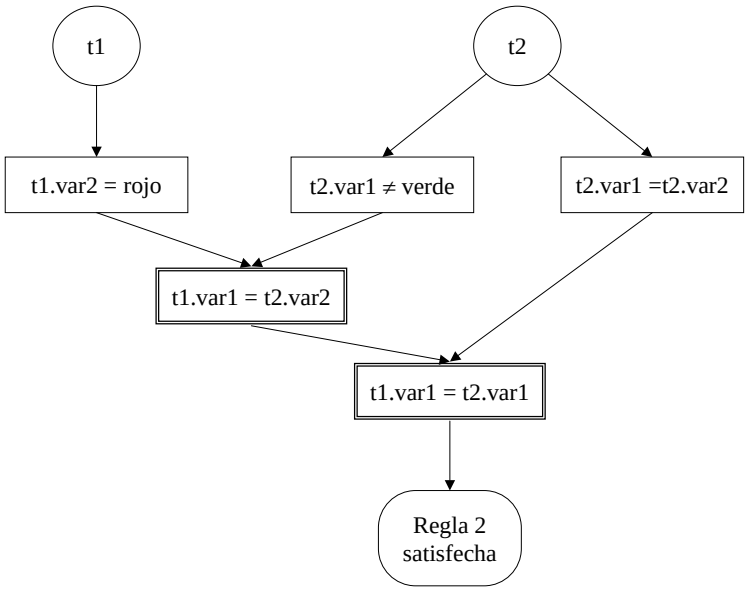

Figura 5.1: Red RETE resultante de la compilación de la regla

Es importante destacar la diferencia entre esta aproximación y el uso clásico del algoritmo RETE. En los lenguajes basados en reglas clásicos, las reglas se definen y se compilan para generar la red RETE desde el principio, y la estructura permanece inmutable. En nuestra propuesta, cada operación de lectura implica construir una estructura temporal que posteriormente será borrada. La estructura permanece mientras se resuelve la operación de lectura. Por otra parte, la principal razón de la red RETE original es su eficiencia en el proceso de reconocimiento de patrones. En nuestro caso, la red y los canales proporcionan, además, un medio para filtrar las tuplas y realizar las comprobaciones de consistencia de ligadura de variables.

5.3.1. Descripción de un algoritmo basado en RETE

La red RETE está compuesta por un **árbol de test** global común a todas las reglas y por un **árbol de enlace** específico a cada regla. El árbol de test realiza todos aquellos tests que involucren a los valores de una única tupla, mientras que el árbol de enlace va a realizar aquellos tests que involucren a dos o más tuplas.

El árbol de test, también llamado *parte alfa*, realiza tests para seleccionar tuplas individuales para cada plantilla involucrada en la operación, basándose en su esquema y en los valores de sus atributos. Contiene nodos *tipo*, nodos α y memorias α.

- Los **nodos tipo** filtran las tuplas según su esquema XML. Existirá un *nodo tipo*

por cada plantilla definida en la operación (siendo más precisos, existirá un *nodo tipo* por cada canal distinto involucrado en la operación). Comprueba que la tupla tiene el mismo esquema XML que la plantilla correspondiente.

- Los **nodos** α son nodos de una entrada. Realizan tests sobre los valores de las tuplas. Cada nodo α representa un test sobre algún atributo. Por ejemplo, comprobar que `t1.var2=rojo` o que `origen='Madrid'`. También realizan los tests de consistencia de ligadura de variables dentro de la misma plantilla (tests intra-plantilla) cuando una variable aparece más de una vez en la plantilla, por ejemplo `t2.var1=t2.var2`, así como tests relacionales simples (mayor, menor, mayor que, ...), como `edad<26`, o cualquier test definido mediante una función booleana.

- Cada nodo α almacena sus resultados en su correspondiente **memoria** α. La memoria α almacena las tuplas en curso del canal que satisfacen la condición definida en el nodo.

El árbol de enlace, también llamado parte beta, relaciona la información proveniente de diferentes plantillas y recoge los resultados de las instancias parciales. Contiene nodos β, memorias β y nodos terminales.

- Los **nodos** β, también llamados nodos de enlace, tienen dos entradas. Cada nodo β realiza un test de consistencia de ligaduras de valores de variables entre plantillas distintas.

- Cada nodo β tiene asociadas dos **memorias** β. Las memorias β almacenan las instancias parciales, es decir, las combinaciones de tuplas que se corresponden con las plantillas evaluadas hasta ese momento, pero no con todas. Estas instancias parciales se denominan *tokens*.

- El último nodo de enlace se denomina **nodo terminal**. Cada nodo terminal representa una operación de lectura múltiple. La lista de tuplas asociada al nodo terminal representa un conjunto completo de tuplas que se corresponden con las plantillas definidas en la operación. También se asociarán a este nodo los procesos bloqueados que están esperando completar la operación.

Cuando se invoca una operación de lectura con varias plantillas $rd_n(templ_1, temp_2, \ldots, templ_n)$, se crea un *nodo tipo* por cada plantilla (ver ejemplo figura 5.2). Cada *nodo tipo* se asocia con su correspondiente canal (según su esquema XML) y actúa como filtro permitiendo pasar únicamente a aquellas tuplas que se correspondan con su esquema. Los nodos β realizan el filtrado. Las memorias β intermedias recogen las ligaduras de variables consistentes; y las memorias β finales, nodos terminales, contienen todas las posibles combinaciones de tuplas (*tokens*) en el espacio de tuplas XML que satisfacen la operación. Una opción alternativa sería guardar únicamente las memorias β finales, como se hace en el algoritmo TREAT.

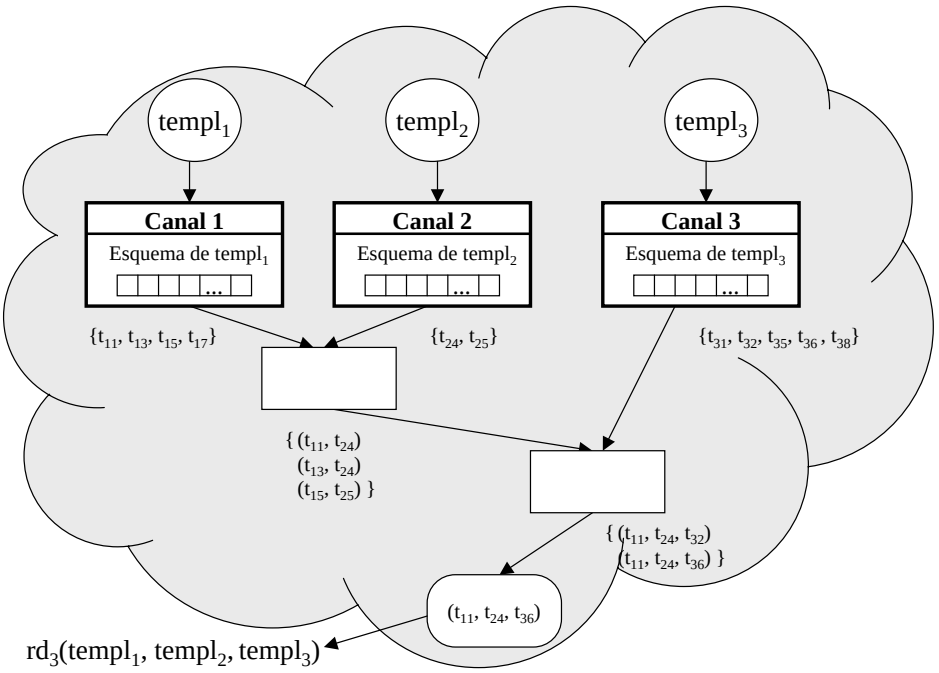

Figura 5.2: Uso de una red RETE en el proceso de correspondencia múltiple

El resultado almacenado en la memoria α de la plantilla $templ_j$ es equivalente al resultado de la operación $rd(templ_j^?)$, donde $templ_j^?$ representa a la plantilla original, pero en la que se han sustituido todas aquellas restricciones en las que aparecen variables de otras plantillas por el símbolo comodín ? y el nombre del atributo correspondiente.

Ahora, las posibles tuplas solución de la operación de lectura parcial $rd_k(templ_1, temp_2, \ldots, templ_k)$ serán el resultado de la operación $rd(templ_1^?) \bowtie \ldots \bowtie rd(templ_k^?)$, donde las operaciones JOIN ($\bowtie$) realizan las comprobaciones de consistencia de ligadura de variables apropiadas.

Los nodos β en la parte β de la red realizan estas operaciones JOIN, y cada memoria β almacena los resultados de uno de estos *joins* intermedios $rd(templ_1^?) \bowtie \ldots \bowtie rd(templ_i^?)$ para algún $i < k$.

Siempre que se modifica el espacio de tuplas hay que actualizar la red. Nótese que esta observación no contradice nuestra anterior afirmación de que, en nuestro uso del algoritmo RETE, las estructuras intermedias desaparecen tras la finalización de cada operación transaccional. Esto es así, puesto que una operación transaccional (de lectura, por ejemplo) puede perdurar en el tiempo, y por tanto verse afectada por modificaciones en el espacio de tuplas, debido al efecto de bloqueo subyacente a las operaciones básicas. La red se actualiza de la siguiente manera: los cambios en el espacio se propagan a través de la parte α de la red y se actualizan las memorias α apropiadas. Estos cambios se propagan entonces a través de los nodos β asociados, activando estos nodos. Si se crea alguna instanciación parcial nueva, entonces se añaden a las memorias β involucradas y se siguen propagando por la parte β de la red, activando otros nodos. Si la propagación

alcanza un nodo terminal, esto indica que la operación de lectura múltiple se ha ejecutado completamente.

5.3.2. Ejemplo de uso

En un caso correspondiente a una agencia de viajes, supongamos que los servicios Web de distintas compañías aéreas, de ferrocarril y de autobuses introducen, en un servicio Web de coordinación, tuplas con información sobre plazas libres para reservar, de forma que los clientes, mediante otro servicio Web, puedan buscar combinaciones de vuelos, trenes y autobuses para viajar desde un lugar de origen hasta otro lugar de destino.

Por ejemplo, un cliente quiere consultar la forma de trasladarse desde Londres a Zaragoza. Una posible solución sería buscar una combinación de vuelos desde Londres a Madrid y luego un tren que conecte Madrid con Zaragoza. Esta consulta, expresada como una regla en CLIPS, sería de la forma:

```
(defrule Regla-1
   (PrimerVuelo  (origen Londres) (destino ?PrimerAeropuerto)
       (salida ?salida&Después(?salida 8:30 0:00) )
       (llegada ?llegadaPrimerAeropuerto)
       (NúmeroBillete ?billetePrimerVuelo))
   (SegundoVuelo  (origen ?PrimerAeropuerto) (destino Madrid)
       (salida ?salida&Después(?salida ?llegadaPrimerAeropuerto 2:00) )
       (llegada ?llegadaSegundoAeropuerto)
       (NúmeroBillete ?billeteSegundoVuelo))
   (Tren (origen Madrid) (destino Zaragoza)
       (salida ?salida&Después(?salida ?llegadaSegundoAeropuerto 1:00) )
       (NúmeroBillete ?BilleteTren))
=> ...)
```

La regla será satisfecha si el espacio de tuplas contiene las tuplas con la información adecuada. En este caso la regla se traduce en una operación de lectura múltiple con tres plantillas, $rd_3(PrimerVuelo, SegundoVuelo, Tren)$, la primera plantilla para obtener un vuelo desde Londres a cualquier ciudad, la segunda para obtener un vuelo desde esta ciudad hasta Madrid, y finalmente la tercera para obtener un tren desde Madrid a Zaragoza.

En el modelo estructurado con restricciones, las plantillas tendrán la siguiente forma:

Plantilla PrimerVuelo:

```
( (tipo, vuelo)
```

```
(origen, Londres)
(destino, ?destino)
(horaSalida, ?horaSalida&(> ?horaSalida 8:30))
(horaLlegada, ?horaLlegada)
(NúmeroBillete, ?NúmeroBillete) )
```

Plantilla SegundoVuelo:

```
( (tipo, vuelo)
  (origen, ?origen&(= ?origen ?PrimerVuelo.destino)
  (destino, Madrid)
  (horaSalida ?horaSalida&(> ?horaSalida (+ ?PrimerVuelo.horaLlegada 2:00)))
  (horaLlegada ?horaLlegada)
  (NúmeroBillete ?NúmeroBillete) )
```

Plantilla Tren:

```
( (tipo, tren)
  (origen, Madrid)
  (destino, Zaragoza)
  (horaSalida ?horaSalida(> ?horaSalida&(+ ?SegundoVuelo.horaLlegada 1:00))
  (horaLlegada ?horaLlegada)
  (NúmeroBillete ?NúmeroBillete) )
```

La consulta de la operación $rd_3(PrimerVuelo, SegundoVuelo, Tren)$ generará, en la memoria de trabajo, la red de la figura 5.3. La evaluación de esta red producirá que aquellas tuplas que pasen los tests del árbol de test quedarán asociadas a sus respectivos nodos β en el árbol de enlace. Si hay tres tuplas que se corresponden con sus respectivas plantillas, éstas se propagarán por la red hasta el último nodo β y la operación se realizará devolviendo las tres tuplas al proceso que invocó la operación de lectura. En caso contrario, los últimos nodos de enlace recogerán todos los datos que satisfacen las plantillas y se bloqueará al proceso invocante. Cada vez que una nueva tupla se añada al espacio de tuplas, ésta se propaga por la red para comprobar si la operación de lectura se completa y desbloquear al proceso.

5.3.3. Algoritmos sobre la red RETE

Existen cuatro algoritmos principales que actuarán sobre la red:

1. Creación de la red a partir de las plantillas de una operación de lectura múltiple.
2. Propagación de las tuplas a través de la red.
3. Eliminación de la presencia de una tupla en la red.
4. Eliminación de una operación de lectura múltiple.

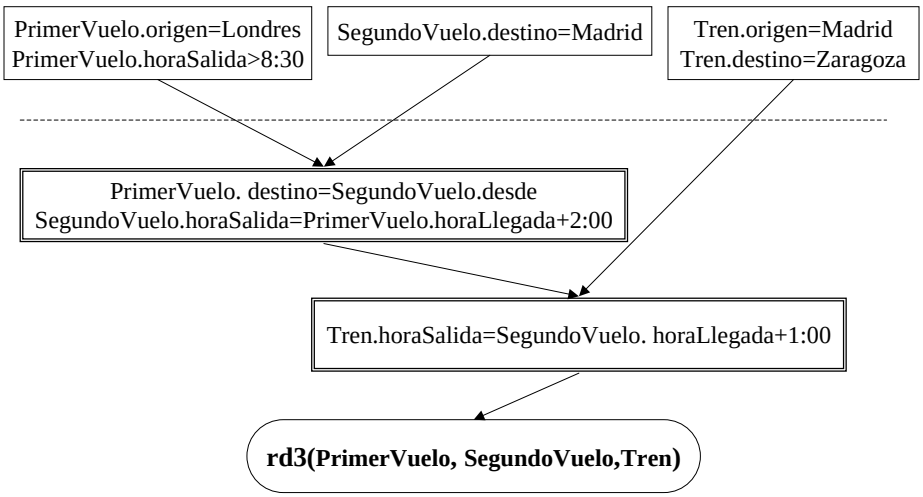

Figura 5.3: Memoria de trabajo de la red RETE

5.3.3.1. Creación de la red a partir de las plantillas de una operación de lectura múltiple

Este algoritmo se ejecutará cada vez que se invoque una nueva operación de lectura múltiple sobre el espacio de tuplas.

- Se creará un *nodo tipo* por cada esquema XML distinto al que pertenezca cada plantilla. Este nodo identifica a la plantilla y a todas las tuplas pertenecientes al mismo canal. Habrá un *nodo tipo* por cada plantilla de la operación múltiple, pero si dos plantillas distintas tienen el mismo esquema XML compartirán su *nodo tipo*.

- Se creará un nodo α por cada restricción que contenga únicamente constantes o funciones y predicados que involucren variables de la propia plantilla (relaciones intraplantillas). Estos nodos colgarán de su correspondiente *nodo tipo*.

- Se creará un nodo β por cada restricción que relacione variables de dos plantillas distintas (relaciones inter-plantillas). Estos nodos colgarán de los últimos nodos α de sus correspondientes plantillas.

- Cada nodo α o β lleva asociado una memoria donde se almacenan las tuplas (o apuntadores a las tuplas) que satisfacen esa condición.

- Cada operación de lectura múltiple generará un nodo terminal enlazado con los correspondientes nodos β asociados a cada una las plantillas definidas en la operación. Si la operación contiene una plantilla perteneciente a un nuevo esquema XML, se generará un nuevo *nodo tipo*. El camino desde el nodo raíz hasta el nodo hoja define la operación.

5.3.3.2. Propagación de las tuplas a través de la red

Este algoritmo se ejecuta sobre cada una de las tuplas existentes en el espacio de tuplas cuando se crea una red nueva a partir de una operación de lectura múltiple o cuando se añade una nueva tupla al espacio de tuplas, y existe una red porque la operación no ha sido resuelta todavía. Su objetivo es suministrar información inicial a las memorias asociadas a los nodos de la red cuando se crea una nueva red y actualizar estas memorias cuando se añaden nuevas tuplas al espacio.

- Los nodos tipo actúan como centinelas, de forma que cada tupla entra en la red a través del *nodo tipo* correspondiente a su canal (mismo esquema XML), en caso de existir. Si no existe dicho *nodo tipo*, la tupla es rechazada.
- La tupla atraviesa los nodos α siempre que verifique sus respectivas condiciones. Si no cumple alguna condición, la tupla es rechazada.
- Si ha pasado todas las condiciones de los nodos α, desciende por los nodos β, formando *tokens*.
- Si se alcanza algún nodo terminal significa que se puede realizar la operación correspondiente con las tuplas pertenecientes al *token* asociado.
- Si se llega a un nodo β en el que se necesite la presencia, en la memoria de este nodo, de otra tupla (correspondencia parcial), se conserva la red y se bloquea al proceso invocante.

5.3.3.3. Eliminación de la presencia de una tupla en la red

Este algoritmo tiene como objetivo la modificación de las memorias β actualizando aquellos *tokens* a los que perteneciera la tupla (actualización de las correspondencias parciales).

- Se entra en la red a través del *nodo tipo* correspondiente al canal de la tupla. Si no existe dicho *nodo tipo*, termina.
- La tupla atraviesa los nodos α y β siempre que verifique sus respectivas condiciones, eliminando su presencia en las memorias correspondientes a cada nodo.
- Termina si no cumple alguna condición o cuando alcanza el nodo terminal.

5.3.3.4. Eliminación de una operación de lectura múltiple

Este algoritmo se activa cuando se completa la ejecución de una operación de lectura múltiple. Su objetivo es la eliminación de la red asociada a la operación, por medio de la eliminación de los nodos y enlaces correspondientes cuando termina la operación.

En una versión alternativa, se podría pensar en mantener la red si se cree que la operación va a ser invocada más veces con las mismas plantillas. Esta creencia podría estar basada en heurísticas ligadas al campo concreto de aplicación, o incluso a cada aplicación particular. Es evidente que, con una decisión de este tipo, es necesario tener en consideración un compromiso espacio/tiempo. Para mitigar esta dicotomía, se puede pensar en un mecanismo de monitorización que determine que sean eliminadas del la red operaciones no pendientes que hace mucho tiempo que no se han invocado.

5.4. Incrementando la fiabilidad del algoritmo basado en RETE

Recordemos que estamos buscando un algoritmo eficiente y fiable en un entorno distribuido que implemente la operación de lectura múltiple rd_n sobre un espacio de tuplas TS. Algunas similitudes entre los lenguajes basados en reglas y los lenguajes de coordinación basados en Linda (como por ejemplo, entre hechos y tuplas, reglas y plantillas, o las operaciones assert/retract y las primitivas out/in) nos dirigieron hacia una implementación de una variante del algoritmo RETE.

Aunque el uso de un algoritmo basado en RETE sugiere la posibilidad de obtener un programa eficiente, surgen dificultades cuando se intenta su análisis formal, debido a la complejidad del algoritmo RETE que actúa como una "caja negra" en muchos entornos de desarrollo de sistemas expertos como OPS5 [SM90], CLIPS [GR94], JESS [Hil03] o Soar [LNR87]. No sólo su análisis formal, sino incluso su comprobación (*testing*) podría ser difícil debido a su empleo en un contexto distribuido.

Para solventar estas dificultades, nuestro objetivo es seguir las mismas ideas explicadas en el artículo [ALR07]. En ese trabajo se estudia el papel de herramientas de razonamiento mecanizado (como ACL2 [KMM04], Isabelle [NPW02] o Coq [Log08]) en la verificación de sistemas software, específicamente en el campo de la manipulación algebraica. Los herramientas teóricas usadas para modelizar programas de computación algebraicos son muy diferentes de las empleadas en los sistemas de coordinación basados en Linda. En el primer caso se usan especificaciones algebraicas [ALR07]; en el segundo, los métodos formales se basan en álgebras de procesos y sistemas de transición (ver [VR02], por ejemplo). Pese a estas diferencias técnicas, la metodología propuesta en [ALR07] puede ser aplicada también en nuestro caso.

La idea general es que, en primer lugar, se verifica la corrección de una versión más simple estructuralmente (aunque seguramente ineficiente) del algoritmo original (eficiente pero supuestamente difícil de verificar); y después se procede a un proceso exhaustivo y mecanizado de comprobación (*testing*) de la versión eficiente *contra* la versión ineficiente (pera ya verificada). En [ALR07] se denomina a esta técnica *testing automatizado*.

Este esquema genérico será reflejado en nuestro caso del siguiente modo: el algoritmo

"complicado" será el de la *lectura múltiple basado en RETE*; introduciremos un algoritmo, pretendidamente equivalente al anterior, pero más simple conceptualmente (y menos eficiente). A continuación daremos una especificación formal completa de este algoritmo ineficiente y argumentos "semi-formales" que demuestran su corrección. La mecanización de esta demostración, y la fase de *testing automatizado* del algoritmo basado en RETE, quedan como trabajo futuro.

La organización de este último apartado de la tesis es la que sigue. En el siguiente subapartado diseñamos el algoritmo simple de lectura múltiple. En el 5.4.2 especificamos el subalgoritmo principal de dicho diseño, y damos una demostración (informal) de su corrección, *basada en la especificación formal de los subalgoritmos que intervienen en él* (dichos procedimientos auxiliares quedan especificados, pero no hemos creído necesario incluir un diseño detallado de los mismos, puesto que una implementación directa es bastante sencilla, teniendo en cuenta que la eficiencia no es un aspecto importante en nuestra aproximación). El mismo esquema se utiliza en el último subapartado, el 5.4.3, respecto al procedimiento general de lectura múltiple: especificación formal, argumentos que muestran su corrección y especificación formal de los subalgoritmos involucrados (sin dar detalles de realización de estos últimos).

5.4.1. Un algoritmo simple de lectura múltiple

La estructura de red RETE puede mejorar la eficiencia en la implementación de la operación de lectura múltiple en dos aspectos: la red almacena los emparejamientos parciales cuando hay ligaduras entre diferentes plantillas, así se evita tener que volver a realizar los emparejamientos desde cero después de cada modificación en el espacio de tuplas por el borrado o insertado de nuevas tuplas; y además, se aprovecha de la similitud entre las plantillas en la misma operación de lectura, de forma que, cuando dos o más plantillas tienen una restricción común, comparten el mismo nodo de la red, reduciendo ciertos tipos de redundancia y cálculos. Si nos olvidamos de ese aprovechamiento fino de la red RETE, vemos que un auxiliar importante sería una función que obtuviese, en cada instante dado, una "fotografía" de qué tuplas del espacio (en ese momento dado) satisfacen una colección de plantillas. Es decir, mantenemos la estructura de datos de la red RETE, pero la "desaprovechamos" realizando un recorrido exhaustivo del espacio

de tuplas. Esta idea queda recogida en la siguiente función.

Function `instantánea_multiple_rd`($\overline{templ}$:*lista de plantillas*):lista de tuplas

```
c̄ ();
foreach plantilla in templ do
    c  obtener_el_correspondiente_canal_de_TS(plantilla);
    c' seleccionar_tuplas_que_satisfacen_tests_locales(c);
    c̄  añadir_canal_a_lista(c̄ ,c');
end
token  ligar_tuplas_seleccionadas(c̄ ,templ);
return token
```

En esta función se inicia una lista de canales $\overline{c}$ vacía. Por cada plantilla de entrada, se obtiene su correspondiente canal c en el espacio de tuplas TS. La operación `seleccionar_tuplas_que_satisfacen_tests_locales` obtiene las tuplas de c que satisfacen las restricciones locales (es decir, aquellas restricciones que involucran a una única tupla en el canal). Las tuplas resultantes c' se concatenan secuencialmente. Cuando finaliza este preprocesamiento, la operación `ligar_tuplas_seleccionadas` realiza las comprobaciones de consistencia de ligadura de variables apropiadas sobre la lista de canales $\overline{c}$ con respecto a las restricciones relativas a varias tuplas.

Si, como se pretende, este procedimiento es equivalente al primer bloque del algoritmo basado en RETE, entonces la equivalencia del siguiente algoritmo con el basado en RETE resulta bastante plausible.

Function `multiple_rd_ineficiente`($\overline{templ}$:*lista de plantillas*):lista de tuplas

```
token  instantánea_multiple_rd(templ);
while not matching_completado?(token) do
    escuchar_una_operación_Linda;
    if TS ha cambiado then
        token  instantánea_multiple_rd(templ);
    end
end
return token
```

Este algoritmo es especificado y estudiado formalmente en el subapartado 5.4.3. En el siguiente realizamos el mismo trabajo para el subalgoritmo principal de éste: `instantánea_multiple_rd`.

5.4.2. Análisis de la función `instantánea_multiple_rd`

En la especificación y verificación de la función `instantánea_multiple_rd` es necesario hacer uso del concepto de *plantilla olvido*. Las siguientes definiciones formalizan

ese concepto.

Definición 5.5 (**Plantilla olvido**)**.** Dada un plantilla *templ* se llama **plantilla olvido**, y la denotamos por $templ^?$, a la plantilla que se obtiene al sustituir en *templ* todas aquellas restricciones en las que se hace referencia a alguna variable que no es propia de la plantilla por el carácter comodín ?.

Definición 5.6 (**Plantilla olvido respecto a una lista de plantillas**)**.** Dada una lista de plantillas $\overline{templ} = templ_1, \ldots, templ_n$ llamamos plantilla olvido de $templ_i$ $1 \leq i \leq n$ respecto a la lista $\overline{templ}$ a la plantilla denotada $templ_i^?$ que se obtiene al sustituir en $templ_i$ todas aquellas restricciones en las que se hace referencia a alguna variable propia de las plantillas posteriores a $templ_i$ en la lista por el carácter comodín ?.

5.4.2.1. Especificación de la función `instantánea_multiple_rd`

Pre $\equiv \overline{templ} = templ_1, \ldots, templ_n \ , \ n > 0$
Función `instantánea_multiple_rd`($\overline{templ}$) **dev** ($\overline{t}$: lista de tuplas)
Post $\equiv \overline{t} = t_1, \ldots, t_j \ , \ j \leq n, \quad t_1, \ldots, t_j \quad TS,$
$mtc_j(templ_1^?, \ldots, templ_j^?, t_1, \ldots, t_j)$

Según el diseño realizado de esta función en la sección 5.4.1, dado un espacio de tuplas TS, representado su estado en un instante dado por $\overline{t}, \overline{w}$, y dada una lista de plantillas $\overline{templ}$, el algoritmo `instantánea_multiple_rd` crea primero una lista de canales $\overline{c}$. Comenzamos con la lista de canales vacía y entramos en un bucle en el que para cada plantilla $templ_i$ perteneciente a la lista de plantillas $\overline{templ}$ se obtiene su correspondiente canal $\overline{c}_i \subseteq \overline{t}$, es decir un multiconjunto con todas las tuplas pertenecientes a $\overline{t}$ con el mismo esquema que la plantilla. Una vez obtenido el canal se seleccionan del mismo aquellas tuplas que satisfagan los tests locales, obteniendo un nuevo multiconjunto de tuplas denominado $\overline{c}_i^?$. El multiconjunto $\overline{c}_i^?$ contiene todas aquellas tuplas que se corresponden con la plantilla olvido $templ_i^?$. Una vez obtenido este canal olvido, se añade a la lista de canales. Por último, obtenida la lista de canales $\overline{c} = \overline{c}_1^?, \ldots, \overline{c}_n^?$, la función `ligar_tuplas_seleccionadas` nos devolverá un *token* que será la lista de tuplas más larga posible que satisface las condiciones de la parte β de la red.

Para que el anterior argumento informal sobre la corrección del algoritmo sea válido es necesario que cada uno de los subalgoritmos involucrados se comporten del modo esperado. En lugar de dar una implementación de cada uno de ellos, nos conformamos aquí con especificar formalmente cada una de las siguientes funciones: `obtener_el_correspondiente_canal_de_TS`, `seleccionar_tuplas_que_satisfacen_tests_locales`, `añadir_canal_a_lista` y `ligar_tuplas_seleccionadas`. Si las precondiciones respectivas son convenientemente encadenadas, se cumplirá la postcondición de la función `instantánea_multiple_rd`.

5.4.2.2. Especificación de la función `obtener_el_correspondiente_canal_de_TS`

Pre $\equiv TS = \overline{t}$
Función `obtener_el_correspondiente_canal_de_TS`($templ$) **dev** ($\overline{c}$: multiconjunto de tuplas)
Post $\equiv \overline{c} = \overline{canal}(templ, \overline{t})$

Esta función recibe una plantilla $templ$ y mediante la operación $\overline{canal}(templ, \overline{t})$, definida en la definición 3.15, se obtiene un multiconjunto con todas las tuplas existentes en el espacio de tuplas TS que tienen el mismo esquema XML que la plantilla.

5.4.2.3. Especificación de la función `seleccionar_tuplas_que_satisfacen_tests_locales`

Pre $\equiv \overline{c}$ canal de $templ$
Función `seleccionar_tuplas_que_satisfacen_tests_locales`($templ$:plantilla; $\overline{c} : canal$) **dev** ($\overline{t}$: multiconjunto of tuplas)
Post $\equiv \overline{t} \subseteq \overline{c}, \quad t \quad \overline{t} \quad mtc(t, templ^?)$

Esta función aplica los tests correspondientes de los nodos α de la red RETE a las plantillas pertenecientes al canal $\overline{c}$ de la plantilla $templ$, que son los parámetros de entrada. El resultado será un multiconjunto de tuplas, subconjunto del canal, que se corresponden con la plantilla olvido $templ^?$.

5.4.2.4. Especificación de la función `añadir_canal_a_lista`

Pre $\equiv \overline{\overline{c}}$
Función `añadir_canal_a_lista`($\overline{\overline{c}}$:lista de canales; $\overline{c}$:canal) **dev** ($\overline{\overline{c}}$: lista de canales)
Post $\equiv caso \quad \overline{\overline{c}} = \quad \text{‘} \quad \overline{\overline{c}} = \overline{c}$
$caso \quad \overline{\overline{c}} = \overline{c}_1, \ldots, \overline{c}_k \quad \text{‘} \quad \overline{\overline{c}} = \overline{c}_1, \ldots, \overline{c}_k, \overline{c}$

Esta función únicamente concatena secuencialmente un canal a una lista de canales.

5.4.2.5. Especificación de la función `ligar_tuplas_seleccionadas`

Pre $\equiv \overline{\overline{c}} = \overline{c}_1, \ldots, \overline{c}_n$ lista de canales respectivos de $templ_1, \ldots, templ_n$
Función `ligar_tuplas_seleccionadas`($\overline{\overline{c}}$: lista de canales; $\overline{templ}$:lista de plantillas) **dev** (token : lista de tuplas)
Post $\equiv token = t_1, \ldots, t_j \ , \quad j \leq n, \quad t_1 \quad c_1, \ldots, t_j \quad c_j,$
$mtc_j(templ_1^?, \ldots, templ_j^?, t_1, \ldots, t_j)$

Esta función aplica los tests de ligaduras de variables de los nodos β. Devolverá el contenido de la memoria β asociada al último nodo evaluado. Este contenido será una lista de tuplas de longitud j que se corresponden respectivamente con las j primeras plantillas olvido. Formalmente, será el resultado de la operación JOIN sobre la salida de la operaciones lectura con las plantillas olvido respectivas $rd(templ_1^?) \bowtie \ldots \bowtie rd(templ_j^?)$

5.4.3. Análisis de la función `multiple_rd_ineficiente`

5.4.3.1. Especificación de la función `multiple_rd_ineficiente`

Pre $\equiv TS = \bar{t}$: multiconjunto de tuplas; $\overline{templ} = templ_1, \ldots, templ_n$: lista de plantillas
Función `multiple_rd_ineficiente`($\bar{t}$: multiconjunto de tuplas; $\overline{templ}$:lista de plantillas) **dev** (token : lista de tuplas)
Post $\equiv token = t_1, \ldots, t_n \ , \quad t_1, \ldots, t_n \quad \bar{t}$
$mtc_n(\overline{templ}, token)$

Según la postcondición $token = t_1, \ldots, t_n \quad t_1, \ldots, t_n \quad \bar{t} \qquad mtc_n(\overline{templ}, \bar{t}\ )$, tenemos que demostrar que dado un espacio de tuplas $\bar{t}$ y una lista de plantillas $\overline{templ}$, la salida del algoritmo ineficiente es una lista de tuplas $\bar{t} = t_1, \ldots, t_n$, que llamaremos *token*, tal que se cumple $mtc_n(\overline{templ}, \bar{t}\ )$.

Analizamos a continuación la satisfacción de esa especificación por el algoritmo introducido en 5.4.1.

Dado un espacio de tuplas TS, representado su estado en un instante dado por $\bar{t}, \overline{w}$, y dada una lista de plantillas $\overline{templ}$, el algoritmo `multiple_rd_ineficiente` obtiene primero (invocando a la función `instantánea_multiple_rd`) un lista de tuplas $token = t_1, \ldots, t_j$, con $j \leq n$, tal que $mtc_j(templ_1^?, \ldots, templ_j^?, t_1, \ldots, t_j)$.

El algoritmo entrará en el bucle en el caso de que $j < n$, es decir, mientras tengamos una lista parcial de emparejamientos. Saldrá del bucle cuando la variable j alcance el valor n, o sea, cuando obtengamos la lista completa de emparejamientos.

Dentro del bucle, el algoritmo comprueba si hay alguna operación pendiente (mediante la función `escuchar_una_operación_Linda`). Si es así, y la operación ha modificado el contenido del espacio de tuplas, entonces se vuelve a invocar a la función `instantánea_multiple_rd` para obtener un nuevo *token* actualizado. La instantánea nos puede llevar a tres situaciones distintas:

- Hemos obtenido el mismo *token*. Seguimos teniendo el mismo valor de la variable j (número de tuplas en el *token*). Esta situación se da cuando se añaden o eliminan

tuplas del espacio de tuplas que no afecta a los emparejamientos parciales.

- La operación ha eliminado alguna tupla perteneciente al *token*. El valor de j es menor porque se ha reducido el número de tuplas en el *token*.
- La operación ha insertado una nueva tupla en el espacio de tuplas y ésta se ha añadido al *token*. El valor de j es ahora mayor porque se ha aumentado el número de tuplas en el *token*.

Ahora se comprueba nuevamente si se ha completado la operación de lectura múltiple, es decir, si la función `matching_completado?` devuelve el valor `verdad`. Como la postcondición de esta función es que el número de tuplas del *token* sea igual al número de plantillas definidas en la operación de lectura, el bucle no terminará hasta obtener un *token* de tamaño n, o sea, hasta obtener una lista completa de emparejamientos.

Por tanto, la corrección de la función `multiple_rd_ineficiente` se basa principalmente en la corrección de la función `instantánea_multiple_rd`, previamente justificada en 5.4.2, y en que las funciones `matching_completado?` y `escuchar_una_operación_Linda` satisfagan sus especificaciones, que son expresadas formalmente a continuación.

5.4.3.2. Especificación de la función `matching_completado?`

Pre $\equiv token = \ t_1, \ldots, t_s \ , \quad 0 \leq s \leq n$
Función `matching_completado?`(token : lista de tuplas; n : entero) **dev** (resp : booleano)
Post $\equiv$ resp=verdad si $s = n$
resp=falso si $s < n$

La operación de correspondencia múltiple se completará cuando, en la evaluación de la red RETE asociada, se alcance un nodo terminal. Cuando esto ocurra, la longitud de la lista de tuplas *token* será igual a n, que es el número de plantillas involucradas en la operación de lectura múltiple.

5.4.3.3. Especificación de la función `escuchar_una_operación_Linda`

Pre $\equiv \overline{t}$: multiconjunto de tuplas
Función `escuchar_una_operación_Linda`
Post $\equiv$ caso $S(w, \overline{t}),\ \overline{t}, w\ \overline{w}, oe, \overline{t}', \overline{w}' \quad E(w) \ \text{‹} \quad \overline{t}, w\ \overline{w} \ \stackrel{oe}{-}_O \ \overline{t}', \overline{w}$
caso $\nexists\, w \quad \overline{w} : S(w, \overline{t}) \ \text{‹} \quad \overline{t}, \overline{w} \ \stackrel{ie}{-}_I \ \overline{t}, ie\ \overline{w}$

Esta función estará a la espera de una operación, de lectura, de escritura o extracción, sobre el espacio de tuplas. Si la operación puede ser atendida se producirá una

modificación en el espacio de tuplas. En caso contrario se añadirá a la lista de peticiones pendientes. Las modificaciones que se producen en el espacio de tuplas están definidas formalmente en las secciones 3.1 y 5.1.7.

Conclusiones y trabajo futuro

La conclusión principal del trabajo de investigación recogido en esta memoria es que los métodos formales, cuando son usados con el grado adecuado de abstracción, pueden resultar útiles en el análisis y mejora de sistemas software de aplicación *real*. Lejos de ser una verdad obvia o de amplia aceptación (véase, solamente a título de ejemplo, que las dos charlas plenarias del congreso *Formal Methods and Software Engineering 2008* estuvieron dedicadas a estudiar las razones por las que los métodos matemáticos no son aceptados en la industria del software [Kat08, Off08]), es ésta una conclusión provisional que debe seguir siendo sometida a indagación y desarrollo.

Para ilustrar esta tesis, el hilo conductor de la memoria ha sido un ejemplo real de sistema software empleado con éxito en aplicaciones industriales. Más concretamente, se trata de un Servicio Web de Coordinación de Servicios Web(WCS) [Álv03]. Su aplicabilidad real ha sido comprobada en varios proyectos y productos relacionados con sistemas basados en la localización [Álv03], [ÁBM+03c], [ZÁG+00].

Es importante destacar que no sólo el Servicio de Coordinación genérico ha guiado nuestra investigación, sino también el *dominio concreto de aplicación*. Es ésa otra lección aprendida durante el proceso de esta investigación: si los métodos formales deben ser adecuados, no pueden aspirar a aplicarse con *universalidad*. Cada ámbito de aplicación requiere un justo grado de formalización, y unas ciertas simplificaciones sintácticas y semánticas. En caso contrario, la investigación sobre métodos formales se convierte en una subárea de la matemática pura, digna de respeto y útil para el progreso general del conocimiento, pero de la que no se podrá esperar una ayuda directa para el proceso práctico de desarrollo de software.

Tras un primer capítulo de preliminares, en el que introducimos tanto el ámbito general de nuestra investigación (la coordinación de servicios Web) como el formalismo de base de nuestros estudios (el modelo Linda) y su tecnología asociada (*JavaSpaces*), el segundo capítulo sirve para describir, con el nivel adecuado de detalle, el diseño y las decisiones de implementación del Servicio Web de Coordinación. Esta descripción nos permitió delimitar un punto clave en el que los métodos formales podrían ser de utilidad: determinar si el comportamiento ofrecido por el Servicio Web de Coordinación respondía a la semántica del modelo Linda. Por otra parte, como hemos mencionado más arriba, el campo concreto de aplicación (los Sistemas Basados en la Localización) y el uso *real* en él del Servicio de Coordinación, permitió una definición precisa del alcance del marco en

el que debíamos abordar nuestro estudio formal. Concretamente, pese a que el Servicio de Coordinación podría emplear para comunicar procesos cualquier documento XML, las necesidades reales mostraron que bastaba con tratar con documentos no anidados que, en los capítulos posteriores, fueron representados de un modo cómodo en nuestros modelos formales.

En el capítulo 3, desarrollamos nuestro modelo formal en el que demostramos, con precisión matemática, que el proceso de *emparejamiento en dos pasos* del Servicio de Coordinación respeta el modelo Linda. Por supuesto, esto no es equivalente a afirmar que la *implementación* del Servicio de Coordinación respeta la semántica de Linda (puesto que nuestra demostración está realizada en el *modelo* y no en el *producto real*), pero al menos aumenta la confianza en el mismo, pues asegura que las ideas que guiaron su diseño son consistentes con los objetivos perseguidos.

Es interesante destacar (como otro aspecto más de nuestra apuesta por la *aplicabilidad* de los métodos formales) que el modelo formal de base elegido no ha sido el más extendido, basado en el *álgebra de procesos*, con sus distintas variantes [BGZ98], [BGZ00], [Zav00], sino el propuesto en [VR02], construido sobre sistemas de transición [Plo81]. La aproximación mediante sistemas de transición nos ha permitido ser más explícitos en la representación de las tuplas que sirven como mecanismos de coordinación (en concreto, como hemos indicado más arriba, se trata de una representación de tuplas XML *planas*). Gracias a este control explícito de las estructuras de datos (que aparecen solo implícitamente en las aproximaciones basadas en álgebras de procesos), hemos podido establecer un modelo muy ajustado al Servicio de Coordinación, a través del concepto de *canal*, que resulta ser central tanto en la implementación real como en el modelo formal.

Como un *subproducto* de este capítulo 3 (pero al que le damos tanta importancia, si no más, que a las demostraciones formales de propiedades en el modelo) obtuvimos la idea de que el modelo Linda puede verse sumamente enriquecido sin más que cambiar el modo en que las tuplas son emparejadas. Nuestro modelo formal fue esencial para ese apercibimiento, pues en él se vio con claridad que el *matching en dos pasos* propuesto no alteraba la semántica de Linda. Esto abrió el camino para explorar otros procedimientos de *matching complejo*.

El capítulo 4 recoge dos aplicaciones con emparejamiento complejo. En la primera, presentamos un *matching semántico*. Aquí el emparejamiento de patrones no se realiza entre cadenas de caracteres, sino entre los *conceptos* que las cadenas representan, basándonos en una noción heurística de *distancia conceptual*. Esta distancia es calculada por un sistema de *votación* entre términos que aparecen en tesauros (relacionados con la información geográfica; de nuevo el dominio determina la metodología), apoyándonos para ello en la base de conocimiento léxica *WordNet*. Esto nos introdujo en el ámbito de la Lingüística Computacional y la Inteligencia Artificial, dando a esta parte de la memoria un estilo bastante diferente del de todo el resto. Estas aportaciones sobre la *desambiguación semántica* también conocieron una aplicación para el *alineamiento de tesauros*, que fue utilizada en el proyecto ISIGIS [BBG+01], [NLN+01].

En la segunda parte del capítulo 4 nos ocupamos de un tipo de emparejamiento que aparece como una generalización natural del *matching* estructurado introducido en el capítulo 3. Admitimos aquí que el emparejamiento esté basado en *restricciones* (por ejemplo, se pueden emparejar valores numéricos que sean mayores que una cierta constante o que estén dentro de un rango). Más que por su valor en sí, esta pequeña generalización nos sirvió como inspiración para la contribución recogida en el capítulo 5. En concreto, al introducir el lenguaje de restricciones resultó más explícita la cercanía entre el modelo Linda y los Sistemas Basados en Reglas, lo que nos guió hacia la propuesta de un mecanismo de emparejamiento *múltiple*, en el que aparecen involucradas varias plantillas y varias tuplas.

Es de destacar que las definiciones del capítulo 5 suponen realmente un *cambio de modelo*, frente a lo realizado en los capítulos anteriores, en los que hemos trabajado con *particularizaciones* del modelo Linda. Nuestro modelo con correspondencia múltiple tiene un capacidad expresiva estrictamente mayor que la del modelo Linda, como explicamos en la memoria (en una situación similar a la demostrada formalmente en [Zav00]). El nuevo modelo propuesto permite enriquecer el modelo Linda con capacidades *transaccionales*, que parecen ser una condición necesaria para su utilización en situaciones de coordinación complejas (véanse en [AS92], [BCGZ01], [LZH08], [Zav00] otras aproximaciones alternativas a este mismo problema). Para conseguir un algoritmo eficiente de emparejamiento en este contexto, hemos diseñado una variante del conocido algoritmo RETE [For82] (ampliamente utilizado en Sistemas Basados en Reglas como CLIPS [GR94]).

Puesto que la verificación formal de la corrección de nuestro algoritmo parece complicada (como complicada es la verificación de RETE [For82]), terminamos el capítulo 5 con una propuesta metodológica que fue introducida por primera vez en [ALR07]. Se trata de reemplazar el algoritmo original a verificar por una variante más sencilla desde el punto de vista lógico (lo que, en la mayoría de los casos, implica una perdida significativa de eficiencia). La variante sencilla es entonces validada formalmente, y se entra en una fase posterior de comprobación sistemática y automatizada de la equivalencia entre el algoritmo eficiente (pero no verificado) y el algoritmo verificado (en [ALR07] se denominó a esta técnica *automated testing*). Pese a que evidentemente esto no implica una demostración de la corrección del algoritmo inicial, sí que aumenta su fiabilidad al haberse guiado el *testing* por el comportamiento de un programa del que sí se ha demostrado su corrección.

En nuestro caso, hemos propuesto un algoritmo (claramente ineficiente) que pretendemos equivalente al basado en RETE. Hemos especificado formalmente dicho algoritmo, y hemos dado argumentos (semi-formales) que justifican su corrección. El paso al *automated testing* queda como trabajo futuro.

Y empalmando así con las líneas de investigación abiertas, podemos decir que mucho queda por hacer. En lo relativo a los métodos formales, es evidente que sería necesario realizar una comparación de nuestra aproximación con las (mayoritarias) basadas en álgebras de procesos. Más allá de la *completitud teórica* que ello significaría, esta com-

paración tendría su interés para continuar la labor que ha quedado pendiente al terminar el capítulo 5. Esto es así puesto que, pese a adecuarse peor para la representación explícita de los espacios de tuplas, la sintaxis sumamente abstracta del álgebra de procesos se adapta mejor a los procesos de demostración mecanizada que sería necesario aplicar para formalizar y automatizar completamente la comparación entre nuestro algoritmo ineficiente (que actúa como modelo) y el basado en RETE (que juega el papel de aplicación real).

Desde un punto de vista más tecnológico será conveniente profundizar en la investigación de la potencia transaccional de nuestro *modelo con correspondencia múltiple* en entornos distribuidos. En concreto, convendría proponer operaciones que actúen como *compensaciones* (véase [LZH08]) en el caso de que una operación de lectura múltiple distribuida no pueda finalizar su ejecución. En esa misma línea, convendría analizar la relación de nuestra contribución con otras basadas en los estándares para la orquestación [OAS07] y la coreografía [BK04] de servicios Web.

Por último, y desde el punto de vista de la ingeniería, seguramente el reto más importante que queda pendiente es reconstruir un Servicio de Coordinación Web que esté basado en nuestro modelo con correspondencia múltiple. Si ese hipotético producto tuviese una aplicación industrial real, cerraría el bucle que conlleva nuestra propuesta: de un producto software en funcionamiento, a un modelo formal que lo analiza; a partir del modelo se experimenta con mejoras y nuevos diseños, demostrando su corrección; como conclusión, el nuevo modelo enriquecido se implementa produciendo un artefacto ejecutable, de alta fiabilidad y que mejora las prestaciones del modelo inicial. O volviendo a citar el título de uno de los artículos previos a esta memoria: "*From practice to theory, and going back again*".

Bibliografía

[ÁBM+03a] P. Álvarez, J. A. Bañares, E. J. Mata, P. R. Muro-Medrano, and J. Rubio. Generative Communication with Semantic Matching in Distributed Heterogeneous Environments. In *Computer Aided Systems Theory – EUROCAST 2003, 9th International Workshop on Computer Aided Systems Theory, Las Palmas de Gran Canaria, Spain, February 24-28, 2003, Revised Selected Papers*, volume 2809 of *Lecture Notes in Computer Science*, pp. 231–242. Springer, 2003.

[ÁBM03b] P. Álvarez, J. A. Bañares, and P. R. Muro-Medrano. An Architectural Pattern to Extend the Interaction Model between Web-Services: The Location-Based Service Context. In *Service-Oriented Computing – ICSOC 2003, First International Conference, Trento, Italy, December 15-18, 2003, Proceedings*, volume 2910 of *Lecture Notes in Computer Science*, pp. 271–286. Springer, 2003.

[ÁBM+03c] P. Álvarez, J. A. Bañares, P. R. Muro-Medrano, J. Nogueras, and F. J. Zarazaga. A Java Coordination Tool for Web-Service Architectures: The Location-Based Service Context. In *Scientific Engineering for Distributed Java Applications, International Workshop, FIDJI 2002, Luxembourg-Kirchberg, Luxembourg, November 28-29, 2002, Revised Papers*, volume 2604 of *Lecture Notes in Computer Science*, pp. 1–14. Springer, 2003.

[ACKM04] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services. Concepts, Architectures and Applications*. Springer, 2004.

[AFG+02] L. F. Andrade, J. L. Fiadeiro, J. Gouveia, G. Koutsoukos, and M. Wermelinger. Coordination for Orchestration. In *Coordination Models and Languages, 5th International Conference, COORDINATION 2002, YORK, UK, April 8-11, 2002, Proceedings*, volume 2315 of *Lecture Notes in Computer Science*, pp. 5–13. Springer, 2002.

[ALR07] M. Andrés, L. Lambán, and J. Rubio. Executing in Common Lisp, Proving in ACL2. In *Towards Mechanized Mathematical Assistants, 14th Symposium, Calculemus 2007, 6th International Conference, MKM 2007, Hagenberg, Austria, June 27-30, 2007, Proceedings*, volume 4573 of *Lecture Notes in Computer Science*, pp. 1–12. Springer, 2007.

[Álv03] P. Álvarez. *Arquitecturas y Patrones de Diseño para el Modelado e Integración de Sistemas de Información Heterogéneos*. PhD thesis, Universidad de Zaragoza, Diciembre 2003.

[AR96] E. Agirre and G. Rigau. Word Sense Disambiguation using Conceptual Density. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING'96)*, pp. 16–22, Copenhagen, Denmark, 1996. Association for Computational Linguistics.

[Arn99] Ken Arnold. *The Jini specification*. Addison Wesley, 1999.

[AS92] B. G. Anderson and D. Shasha. Persistent Linda: Linda + transactions + query processing. In *Research Directions in High-Level Parallel Programming Languages, Mont Saint-Michel, France, June 17-19, 1991, Proceedings*, volume 574 of *Lecture Notes in Computer Science*, pp. 93–109. Springer, 1992.

[BBG+01] J. A. Bañares, M. A. Bernabé, M. Gould, P. R. Muro-Medrano, and F. J. Zarazaga. Aspectos tecnológicos de la creación de una Infraestructura Nacional Española de Información Geográfica. *Mapping*, 67, pp. 68–77, 2001.

[BCGZ01] N. Busi, P. Ciancarini, R. Gorrieri, and G. Zavattaro. *Coordination of Internet Agents: Models, Technologies, and Applications*, chapter Coordination Models: A Guided Tour, pp. 6–24. Springer, 2001.

[BGLZ04] M. Bravetti, R. Gorrieri, R. Lucchi, and G. Zavattaro. Probabilistic and Prioritized Data Retrieval in the Linda Coordination Model. In *Coordination Models and Languages, 6th International Conference, COORDINATION 2004, Pisa, Italy, February 24-27, 2004, Proceedings*, volume 2949 of *Lecture Notes in Computer Science*, pp. 55–70. Springer, 2004.

[BGZ98] N. Busi, R. Gorrieri, and G. Zavattaro. A Process Algebraic View of Linda Coordination Primitives. *Theoretical Computer Science*, 192(2), pp. 167–199, 1998.

[BGZ00] N. Busi, R. Gorrieri, and G. Zavattaro. Process Calculi for Coordination: From Linda to Javaspaces. In *Algebraic Methodology and Software Technology. 8th International Conference, AMAST 2000, Iowa City, Iowa, USA, May 20-27, 2000, Proceedings*, volume 1816 of *Lecture Notes in Computer Science*, pp. 198–212. Springer, 2000.

[BH08] P. A. Bernstein and L. M. Haas. Information Integration in the Enterprise. *Communications of the ACM*, 51(9), pp. 72–79, 2008.

[BK04] D. Burdett and N. Kavantzas. Ws choreography model. Technical report, W3C, 2004.

[BM93] J. Banâtre and D. Le Métayer. Programming by Multiset Transformation. *Communications of the ACM*, 36(1), pp. 98–111, 1993.

[BMR+96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture. A System of Patterns*. John Wiley and Sons Ltd, 1996.

[BZ08] N. Busi and G. Zavattaro. A process algebraic view of shared dataspace coordination. *Journal of Logic and Algebraic Programming*, 75(1), pp. 52–85, 2008.

[CG89] N. Carriero and D. Gelernter. Linda in context. *Communications of the ACM*, 32(4), pp. 444–458, 1989.

[CGH96] E. Castillo, J. M. Gutiérrez, and A. S. Hadi. *Sistemas Expertos y modelos de Redes Probabilísticas*. Academia de Ingeniería, 1996.

[Cha77] R. L. Chapman, editor. *Roget's International Thesaurus*. Harper and Row, New York, forth edition, 1977.

[Cia96] P. Ciancarini. Coordination Models and Languages as Software Integrators. *ACM Computing Survey*, 28(2), pp. 300–302, June 1996.

[DF99] J. J. Donlon and K. D. Forbus. Using a Geographic Information System for Qualitative Spatial Reasoning about Trafficability. In *Proceedings of the 13th International Workshop on Qualitative Reasoning (QR'99)*, June 1999.

[DPR99] J. Daudé, L. Padró, and G. Rigau. Mapping multilingual hierarchies using relaxation labelling. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC'99)*, pp. 12–19, Maryland, United States, 1999.

[FAGV01] D. Fernández-Amorós, J. Gonzalo, and F. Verdejo. The Role of Conceptual Relation in Word Sense Disambiguation. In *Applications of Natural Language to Information Systems, 6th International Workshop NLDB'01, June 28-29, 2001, Madrid, Spain, Proceedings*, volume 3 of *Lecture Notes in Informatics*, pp. 87–98. GI-Edition, 2001.

[Fen04] D. Fensel. Triple-Space Computing: Semantic Web Services Based on Persistent Publication of Information. In *Intelligence in Communication Systems, IFIP International Conference, INTELLCOMM 2004, Bangkok, Thailand, November 23-26, 2004, Proceedings*, volume 3283 of *Lecture Notes in Computer Science*, pp. 43–53. Springer, 2004.

[FHA99] E. Freeman, S. Hupfer, and K. Arnold. *JavaSpaces. Principles, Patterns, and Practice*. Addison Wesley, 1999.

[For82] C. Forgy. Rete: A Fast Algorithm for the Many Patterns/Many Objects Match Problem. *Artificial Intelligence*, 19(1), pp. 17–37, 1982.

[FT02] R. T. Fielding and R. N. Taylor. Principled Design of the Modern Web Architecture. *ACM Transactions on Internet Technology*, 2(2), pp. 115–150, 2002.

[GC92] D. Gelernter and N. Carriero. Coordination Languages and their Significance. *Communications of the ACM*, 35(2), pp. 97–107, 1992.

[GCY92] W. Gale, K. W. Churh, and D. Yarowsky. A Method for Disambiguating Word Senses in a Large Corpus. *Computers and the Humanities*, 26, pp. 415–439, 1992.

[Gel85] D. Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1), pp. 80–112, 1985.

[Gla04] R. L. Glass. The Mystery of Formal Methods Disuse. *Communications of the ACM*, 47(8), pp. 15–17, 2004.

[GR94] J. Giarratano and G. Riley. *Expert Systems. Principles and Programming.* PWS Publishing Co, 1994.

[Gru93] T. R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2), pp. 199–220, 1993.

[GSB+02] S. Graham, S. Simeonov, T. Boubez, D. Davis, G. Daniels, Y. Nakamura, and R. Neyama. *Building Web Services with Java. Making sense of XML, SOAP, WSDL, and UDDI.* Sams Publishing, 2002.

[GV08] O. Grumberg and H. Veith, editors. *25 Years of Model Checking - History, Achievements, Perspectives*, volume 5000 of *Lecture Notes in Computer Science*. Springer, 2008.

[GVPC98] J. Gonzalo, F. Verdejo, C. Peters, and N. Calzolari. Applying EuroWordNet to Cross-Language Text Retrieval. *Computers and the Humanities, Special Issue on EuroWordNet*, 32, pp. 185–207, 1998.

[HB95] M. Hinchey and J. Bowen, editors. *Applications of Formal Methods.* International Series in Computer Science. Prentice Hall, 1995.

[Hil03] E. F. Hill. *JESS in Action: Java Rule-Based Systems.* Manning Publications Co., 2003.

[HMPH08] T. Harris, S. Marlow, S. Peyton-Jones, and M. Herlihy. Composable Memory Transactions. *Communications of the ACM*, 51(8), pp. 91–100, 2008.

[Jac06] S. Jacobs. *Beginning XML with DOM and Ajax: From Novice to Professional.* Apress, 2006.

[Kat08] T. Katayama. How Can We Make Industry Adopt Formal Methods? In *Formal Methods and Software Engineering, 10th International Conference on Formal Engineering Methods, ICFEM 2008, Kitakyushu-City, Japan, October 27-31, 2008. Proceedings*, number 5256 in Lecture Notes in Computer Science, pp. 1. Springer, 2008.

[KMM04] M. Kaufmann, P. Manolios, and J. Moore. *Computer-Aided Reasoning: An Approach*. Addison Wesley, 2004.

[Les86] M. Lesk. Automating sense disambiguation: How to tell a pine cone from an ice cream cone. In *Proceedings of the SIGDOC 1986 Conference*, pp. 24–26. ACM Press, June 1986.

[LNR87] J. E. Laird, A. Newell, and P. S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(1), pp. 1–64, 1987.

[Log08] LogiCal-project. The Coq Proof Assistant. Technical report, 2008. (Accesible in http://coq.inria.fr/).

[LZH07] J. Li, H. Zhu, and J. He. Algebraic Semantics for Compensable Transactions. In *Theoretical Aspects of Computing – ICTAC 2007, 4th International Colloquium, Macau, China, September 26-28, 2007, Proceedings*, volume 4711 of *Lecture Notes in Computer Science*, pp. 306–321. Springer, 2007.

[LZH08] J. Li, H. Zhu, and J. He. An Observational Model for Transactional Calculus of Services orchestration. In *Theoretical Aspects of Computing - ICTAC 2008, 5th International Colloquium, Istanbul, Turkey, September 1-3, 2008, Proceedings*, volume 5160 of *Lecture Notes in Computer Science*, pp. 201–215. Springer, 2008.

[LZPH07] J. Li, H. Zhu, G. Pu, and J. He. A Formal Model for Compensable Transactions. In *12th International Conference on Engineering of Complex Computer Systems (ICECCS 2007), 10-14 July 2007, Auckland, New Zealand, Proceedings*, pp. 64–.73. IEEE Computer Society, 2007.

[MAB+01] E. J. Mata, J. Ansó, J.A. Bañares, P. Muro-Medrano, and J. Rubio. Enriquecimiento de tesauros con WordNet: una aproximación heurística. In *Actas de la IX Conferencia de la Asociación Española para la Inteligencia Artificial*, pp. 593–602, Gijón, España, Noviembre 2001.

[MÁBR04] E. J. Mata, P. Álvarez, J. A. Bañares, and J. Rubio. Towards an Efficient Rule-Based Coordination of Web Services. In *Advances in Artificial Intelligence - IBERAMIA 2004, 9th Ibero-American Conference on AI, Puebla, México, November 22-26, 2004, Proceedings*, volume 3315 of *Lecture Notes in Computer Science*, pp. 73–82. Springer, 2004.

[MÁBR07a] E. J. Mata, P. Álvarez, J. A. Bañares, and J. Rubio. Formal Modelling of a Coordination System: From Practice to Theory, and Back Again. In

Engineering Societies in the Agents World VII, 7th International Workshop, ESAW 2006, Dublin, Ireland, September 6-8, 2006 Revised Selected and Invited Papers, volume 4457 of *Lecture Notes in Computer Science*, pp. 229–244. Springer, 2007.

[MÁBR07b] E. J. Mata, P. Álvarez, J. A. Bañares, and J. Rubio. Formal Reasoning on a Web Coordination System. In *Computer Aided Systems Theory – EUROCAST 2007, 11th International Conference on Computer Aided Systems Theory, Las Palmas de Gran Canaria, Spain, February 12-16, 2007, Revised Selected Papers*, volume 4739 of *Lecture Notes in Computer Science*, pp. 329–336, 2007.

[MBCT06] H. R. Motahari, B. Benatallah, F. Casati, and F. Toumani. Web Services Interoperability Specifications. *Computer*, 39(5), pp. 24–32, May 2006.

[MBG+02] E. J. Mata, J. A. Bañares, J. Gutiérrez, P. R. Muro-Medrano, and J. Rubio. Semantic Disambiguation of Thesaurus as a Mechanism to Facilitate Multilingual and Thematic Interoperability of Geographical Information Catalogues. In *Proceedings of the 5th AGILE Conference on Geographic Information Science*, pp. 61–66, Palma de Mallorca, Spain, April 2002.

[MC94] T. Malone and K. Crowston. The Interdisciplinary Study of Coordination. *ACM Computing Survey*, 26(2), pp. 87–119, March 1994.

[Mil90] WordNet: An on-line lexical database. *International Journal of Lexicography (Special Issue)*, 3(4), 1990.

[Mir87] D. P. Miranker. TREAT: A Better Match Algorithm for AI Production System Matching. In *Proceedings AAAI-87 Sixth National Conference on Artificial Intelligence*, pp. 42–47. AAAI Press, July 1987.

[MS06] A. Møller and M. I. Schwartzbach. *An Introduction to XML and Web Technologies*. Addison Wesley, 2006.

[NLN+01] J. Nogueras, M. A. Latre, M. Navas, R. Rioja, and P. R. Muro-Medrano. Towards the construction of the Spanish National Geographic Information Infrastructure. In *Proceedings of the EC-GIS 2001, 7th European Commission GI & GIS Workshop, Managing the Mosaic*, Potsdam, Germany, 2001.

[NPW02] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer Verlag, 2002.

[OAS06] OASIS. Web Services Coordination (WS-Coordination). Technical report, 2006. (http://docs.oasis-open.org/ws-tx/wscoor/2006/06).

[OAS07] OASIS. Web Services Business Process Execution Language version 2.0. Technical report, 2007. (http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf).

[Off08] J. Offutt. Programmers Ain't Mathematicians, and Neither Are Testers. In *Formal Methods and Software Engineering, 10th International Conference on Formal Engineering Methods, ICFEM 2008, Kitakyushu-City, Japan, October 27-31, 2008. Proceedings*, volume 5256 of *Lecture Notes in Computer Science*, pp. 2–2. Springer, 2008.

[OGC99] OGC. The OpenGIS Abstract Specification. Topic13: Catalog Services (version 4). Technical report, OpenGIS Consortium Inc, 1999.

[OGC00] OGC. OpenGIS Gazetteer Service Specification (version 0.7.1). Technical report, OpenGIS Consortium Inc, 2000.

[OGC01a] OGC. OpenGIS Geocoder Service Specification (version 0.7.5). Technical report, OpenGIS Consortium Inc, 2001.

[OGC01b] OGC. Web Feature Server Implementation Specification (version 0.0.14). Technical report, OpenGIS Consortium Inc, 2001.

[OGC03] OGC. OGC Web Map Service Interface (version 1.3.0). Technical report, OpenGIS Consortium Inc, 2003.

[Pea88] J. Pearl. *Probabilistic Reasoning in Intelligence Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.

[Pel03] C. Peltz. Web Services Orchestration and Choreography. *Computer*, 36(10), pp. 46–52, 2003.

[Plo81] G. D. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981.

[Pri08] D. Pritchett. BASE: An ACID Alternative. *ACM Queue*, 6(3), pp. 48–55, 2008.

[Res95] P. Resnik. Disambiguating Noun Groupings with Respect to WordNet sense. In *Proceedings 3rd Workshop on Very Large Corpora, MIT*, pp. 54–68, Cambridge, Massachusetts, June 1995.

[RY97] P. Resnik and D.d Yarowsky. A Perspective on Word Sense Disambiguation Methods and Their Evaluation. In Marc Light, editor, *ACL SIGLEX Workshop on Tagging Text with Lexical Semantics: Why, What and How?*, pp. 79–86, Washington, D.C., 1997.

[SF05] L. Sánchez and N. Fernández. La web Semántica: fundamentos y breve "estado del arte". *Novática*, (178), pp. 6–11, Noviembre–Diciembre 2005.

[SM90] P. D. Sherman and J. C. Martin. *An Ops5 Primer: Introduction to Rule-Based Expert Systems*. Prentice Hall, 1990.

[TBN05] R. Tolksdorf, E. P. Bontas, and L. J. B. Nixon. Towards a Tuplespace-Based Middleware for the Semantic Web. In *2005 IEEE / WIC / ACM International Conference on Web Intelligence (WI 2005), 19-22 September 2005, Compiegne, France*, pp. 338–344. IEEE Computer Society, 2005.

[TG01] R. Tolksdorf and D. Glaubitz. Coordinating Web-Based Systems with Documents in XMLSpaces. In *Cooperative Information Systems, 9th International Conference, CoopIS 2001, Trento, Italy, September 5-7, 2001, Proceedings*, volume 2172 of *Lecture Notes in Computer Science*, pp. 356–370. Springer, 2001.

[Vos01] P. Vossen. Extending, Trimming and Fusing WordNet for Technical Documents. In *Proceedings on NAACL-2001 Workshop on WordNet and Other Lexical Resources Applications, Extensions and Customizations*, Pittsburgh, USA, June 2001.

[VR02] M. Viroli and A Ricci. Tuple-Based Coordination Models in Event-Based Scenarios. In *IEEE 22nd International Conference on Distributed Computing Systems (ICDCS 2002 Workshops) - DEBS'02 International Workshop on Distributed Event-Based Sytem*, Vienna, Austria, July 2002.

[W3C04] W3C. Web Service Glossary. Technical report, 2004. (http://www.w3.org/TR/ws-gloss/).

[WMLF98] P. Wyckoff, S. W. McLaughry, T. J. Lehman, and D. A. Ford. TSpaces. *IBM System Journal. Java Technology*, 37(3), pp. 454–474, 1998.

[WOB98] J. Wiebe, T. O'Hara, and R. Bruce. Constructing Bayesian Networks from WordNet for Word-Sense Disambiguation: Representational and Processing Issues. In *Proceedings of COLING-ACL'98 Workshop on the Usage of WordNet in Natural Language Processing Systems, Montreal*, 1998.

[Yar92] D. Yarowsky. Word-Sense Disambiguation Using Statistical Models of Roget's Categories Trained on Large Corpora. In *Proceedings of the 14th International Conference on Computational Linguistics (Coling'92), Nantes, France*, pp. 454–460, 1992.

[ZÁG+00] F. J. Zarazaga, P. Álvarez, J. Guilló, R. López, and J. Valiño. Use cases of vehicle location systems based on distributed real-time GPS data. In *Proceedings of the Second International Symposium on TeleGeoProcessing, Shopia-Antipolis (France)*, pp. 53–61, 2000.

[Zav00] G. Zavattaro. *Coordination Models and Languages: Semantics and expressiveness*. PhD thesis, Department of Computer Science, University of Bologna, Italy, February 2000.

[zMNS05] M. zur Muehlen, J. V. Nickerson, and K. D. Swenson. Developing web services choreography standards – the case of REST vs. SOAP. *Decision Support Systems*, 40(1), pp. 9–29, 2005.

[ZW03] Y. Zhang and M. Weiss. Virtual communities and team formation. *Crossroads*, 10(1), pp. 5–5, 2003.