UNIVERSIDAD DE DEUSTO

RAZONADOR HÍBRIDO DE CONTEXTO PARA ENTORNOS INTELIGENTES

Tesis doctoral presentada por David Ausín Ortega
dentro del Programa de Doctorado en ingeniería informática y telecomunicación
Dirigida por Dr. Diego López-de-Ipiña
y Dr. Federico Castanedo

UNIVERSIDAD DE DEUSTO

RAZONADOR HÍBRIDO DE CONTEXTO PARA ENTORNOS INTELIGENTES

Tesis doctoral presentada por David Ausín Ortega
dentro del Programa de Doctorado en ingeniería informática y telecomunicación

Dirigida por Dr. Diego López-de-Ipiña
y Dr. Federico Castanedo

El doctorando | El director | El director

Bilbao, mayo de 2015

Razonador híbrido de contexto para entornos inteligentes

Autor: David Ausín
Director: Dr. Diego López-de-Ipiña
Codirector: Dr. Federico Castanedo

Impreso en Bilbao
Primera edición septiembre 2015

A mis padres y abuelos.

Resumen

Se estima que en el futuro casi un cuarto de la población europea será mayor de 65 años. Por esta razón, han cobrado especial interés los entornos inteligentes para promover la autonomía personal de las personas mayores. El método preferido para modelar la información de los entornos inteligentes son las ontologías. Sin embargo, estas presentan el inconveniente de que pueden no ser adecuadas cuando existe la incertidumbre en el contexto. Con el objetivo de solventar este problema se propone seguir un enfoque probabilístico que combine las ontologías con las redes bayesianas. Con este fin este trabajo presenta:

1. Un método para modelar ontologías probabilísticas que combine redes bayesianas con ontologías OWL, a la vez que permite reutilizar estas ontologías en razonadores no probabilísticos de manera transparente.
2. Un razonador que sea capaz de inferir aserciones probabilísticas en base a la ontología probabilística. Este razonador añade una capa de abstracción a los razonadores tradicionales para poder interpretar las ontologías probabilísticas.
3. Un lenguaje declarativo que permita la ejecución de consultas probabilísticas complejas. . Este lenguaje permitirá mezclar consultas de aserciones probabilísticas con aserciones de OWL.

Durante el desarrollo se evaluaron diferentes frameworks y razonadores semánticos con el objetivo de usarlos como base para el razonador probabilístico. Además, se ha medido el rendimiento del razonador en un escenario representativo para probar su idoneidad para añadir ”inteligencia” a los entornos inteligentes.

Abstract

It is estimated that in the near future almost a quarter of the European population is going to be older than 65 years. Due to this fact, Intelligent Environments have gained a great interest to promote personal autonomy. Ontologies are the favourite method to model the information of Intelligent Environments. However, they may not be suitable when uncertainty is present in the context. To solve this drawback, a probabilistic approach which combines Bayesian Networks with ontologies is proposed. To achieve it, this work contributes with:

1. A method to model probabilistic ontologies that combines Bayesian networks with OWL ontologies. At the same time, this method allows developers to reutilize these ontologies in traditional reasoners, in a transparent way.
2. A reasoner that is able to infer probabilistic assertions based on the probabilistic ontology. This reasoner adds a new layer to traditional reasoners in order to be able to understand probabilistic ontologies.
3. A declarative language that supports the execution of complex probabilistic queries. This language supports the mixture of queries about probabilistic assertions and OWL assertions.

On the other hand, this work has evaluated the semantic frameworks and reasoners available for the development of the probabilistic reasoners. Its aim is to select the best OWL reasoners and semantic frameworks that can be employed to develop the base for the probabilistic reasoner. Besides, the performance of the probabilistic reasoner has been measured in a representative scenario to probe its suitability to add ”intelligence” to intelligent environments.

Agradecimientos

Esta tesis es el trabajo de casi cinco años de una vida, la mía. No obstante, sería injusto atribuirse el mérito del mismo única y exclusivamente a un servidor. La ejecución del mismo y la consecución de su terminación no hubieran sido posibles sin el apoyo, la ayuda y confianza de otros muchos, cuyos nombres se desconocerían si no fuese por la posibilidad que estas líneas me brindan. En primer lugar me gustaría agradecer a mis padres y mis abuelos el inculcarme la importancia del conocimiento. Ellos me transmitieron el amor por la ciencia y el afán por la búsqueda de conocimientos. El destino de un hombre no está escrito, pero el camino a este depende de sus decisiones. Tal vez, sin ellos mi camino hubiese sido otro.

En segundo lugar, me gustaría exponer mi gratitud a mis dos directores de tesis, Diego y Federico, por haberme marcado el camino a seguir a lo largo de los años. Sin su estimable ayuda no hubiera sido posible la culminación de este trabajo, ni haber alcanzado las cuotas de excelencia necesarias para la presentación del mismo.

Adicionalmente, me gustaría dar las gracias a la unidad de Internet de DeustoTech - Deusto Institute of Technology por haber apostado por mí, ofreciéndome la oportunidad de realizar el doctorado con ellos, y a mis compañeros de trabajo por haberme ofrecido su ayuda e intercambiado puntos de vista sobre mis ideas.

Finalmente, no puedo acabar estos agradecimientos sin mencionar a mis amigos que me han dado ánimos para proseguir con el desarrollo de la tesis en los momentos de pesimismo.

Gracias a todos,

David

septiembre 2015

Índice general

Índice de figuras **xiii**

Índice de cuadros **xvii**

1 Introducción **1**

1.1 Motivación 3
1.2 Hipótesis, objetivos y limitaciones 4
1.3 Metodología de investigación 6
1.4 Estructura de la tesis 8

2 Antecedentes **9**

2.1 El lenguaje de ontologías OWL 9
2.1.1 Sintaxis 10
2.1.2 Semántica 10
2.1.3 Perfiles en OWL 2 11
2.1.4 Estructura conceptual de OWL 2 12
2.2 Lógicas descriptivas 23
2.3 Métodos para gestionar la incertidumbre 32
2.3.1 La teoría de la probabilidad 33
2.4 Conclusiones 35

3 Estado del arte **37**

3.1 Extensiones probabilísticas de lógicas descriptivas 37
3.1.1 P-CLASSIC 38
3.1.2 Pronto 38

3.1.3 Bayesian *DL–Lite*$_{\mathcal{R}}$ 40
3.2 Extensiones probabilísticas de OWL 41
3.2.1 BayesOWL 41
3.2.2 OntoBayes 43
3.2.3 PR-OWL 44
3.3 Conclusiones 46

4 Representación y razonamiento de la incertidumbre 49
4.1 Primera iteración: añadiendo incertidumbre a las ontologías 50
4.2 Segunda iteración: separación del modelo probabilístico del vocabulario del dominio 56
4.3 Tercera iteración: definición final del sistema de representación y de la inferencia a realizar 64
4.3.1 Objetivo 64
4.3.2 La ontología probabilística de Turambar 65
4.3.3 Funcionamiento básico 66
4.3.4 Modelado de la probabilidad 69
4.3.5 Funciones de estado y tipos de nodos 72
4.3.6 Diferencias con la versión 2 73
4.4 Conclusiones 74

5 Consultando las ontologías probabilísticas 75
5.1 Respondiendo una consulta probabilística 76
5.1.1 Lenguaje de consultas en Turambar 79
5.1.2 Introducción a SPARQL-DL 79
5.1.3 Extensión de Derivo 80
5.1.4 Extensión propuesta 84
5.1.4.1 Ejemplos de consultas 84
5.2 Conclusiones 87

6 Evaluación 89
6.1 Evaluación de razonadores y frameworks semánticos 90
6.1.1 Preparación del experimento 90
6.1.2 Resultados del experimento 92

6.1.3 Conclusiones del experimento 99
6.2 Experimento de Turambar 100
6.2.1 Definición de la ontología de Turambar 103
6.2.2 Ejecución del experimento 109
6.2.3 Conclusiones del experimento 122
6.3 Conclusiones 124

7 Conclusiones **127**
7.1 Conclusiones generales y contribuciones 127
7.1.1 Publicaciones 130
7.2 Trabajo Futuro 131
7.3 Comentarios finales 133

Bibliografía **135**

Índice de figuras

1.1 Representación gráfica de la metodología de investigación 6

3.1 Representación de los conceptos básicos de PR-OWL 45

4.1 Introducción de las principales caracterísitcas del sistema de representación de la incertidumbre a lo largo de las iteraciones. 51
4.2 Ejemplo de la versión 1 de Turambar 52
4.3 Ejemplo de red bayesiana que describe la probabilidad de que un individuo se haya tomado mal un medicamento. 56
4.4 Ontología de Turambar de la segunda versión de nuestra propuesta 58
4.5 Diferencias entre el modo de mundo abierto estricto y no estricto en la segunda versión de Turambar 59
4.6 Aserciones probabilísticas disponibles en la tercera versión de Turambar 65
4.7 Relación entre una ontología OWL 2 DL vinculada a una ontología de dependencias en la tercera versión de Turambar. 68
4.8 Ontología de Turambar en la tercera iteración de Turambar 70

5.1 Algoritmo que muestra los pasos que se siguen a la hora de calcular una aserción probabilística 78

6.1 Parte de las principales subclases de *Activity* definidas en la ontología utilizada para determinar cuándo un usuario debe de tomar la medicación (1 de 3) 93

6.2 Parte de las principales subclases de *Activity* definidas en la ontología utilizada para determinar cuándo un usuario debe de tomar la medicación (2 de 3) 94

6.3 Parte de las principales subclases de *Activity* definidas en la ontología utilizada para determinar cuándo un usuario debe de tomar la medicación (3 de 3) 95

6.4 Tiempo requerido para procesar las actividadees del experimento de las medicinas a lo largo de siete días. El gráfico superior muestra los resultados de ejecutar los razonadores como librerías y el inferior el de ejecutarlos con OWLlink API en modo cliente/servidor. 96

6.5 Resultados de la memoria entregada (*committed memory*) cuando se incrementa el número de días (el volumen de datos crece) La imagen superior muestra la memoria utilizada por el uso de los razonadores como librerías y la inferior, los resultados de la parte cliente de las combinaciones con OWLlink API 97

6.6 Resultados de la memoria usada cuando se incrementa el número de días (el volumen de datos crece). La imagen superior muestra la memoria utilizada por el uso de los razonadores como librerías y la inferior, los resultados de la parte cliente de las combinaciones con OWLlink API 98

6.7 Red bayesiana de ejemplo modelada en Turambar para estimar dónde se encuentra un usuario. 102

6.8 Tiempo de ejecución total a medida que se incrementan el número de usuarios. 109

6.9 La imagen superior muestra la media del tiempo empleado por etapa y la tabla inferior el porcentaje de tiempo de cada etapa respecto al tiempo total del experimento junto con la media y el error estándar. Ambas incluyen los tiempos de carga y de la primera consistencia. Los resultados son para un usuario. 112

6.10 La imagen superior muestra la media del tiempo empleado por etapa y la tabla inferior el porcentaje de tiempo de cada etapa respecto al tiempo total del experimento junto con la media y el error estándar. Ambas incluyen los tiempos de carga y de la primera consistencia. Los resultados son para dos usuarios. 113
6.11 La imagen superior muestra la media del tiempo empleado por etapa y la inferior el porcentaje de tiempo de cada etapa respecto al tiempo total del experimento junto con la media y el error estándar. Ambas incluyen los tiempos de carga y de la primera consistencia. Los resultados son para tres usuarios. 114
6.12 La imagen superior muestra la media del tiempo empleado por etapa y la tabla inferior el porcentaje de tiempo de cada etapa respecto al tiempo total del experimento junto con la media y el error estándar. Ambas incluyen los tiempos de carga y de la primera consistencia. Los resultados son para cuatro usuarios. 115
6.13 Tiempo que transcurre realizando cambios en la ontología para cada etapa cuando hay uno y dos usuarios. 116
6.14 Tiempo que transcurre realizando cambios en la ontología para cada etapa cuando hay tres y cuatro usuarios. 117
6.15 Tiempo transcurrido para determinar la consistencia de la ontología en cada etapa para uno y dos usuarios. 118
6.16 Tiempo transcurrido para determinar la consistencia de la ontología en cada etapa para tres y cuatro usuarios. 119
6.17 Tiempo empleado para responder una consulta probabilística para uno y dos usuarios. 120
6.18 Tiempo empleado para responder una consulta probabilística para tres y cuatro usuarios. 121

Índice de cuadros

2.1 Lista de constructores disponibles en $\mathcal{SROIQ}$ junto con su sintaxis y semántica. En ella a, representa individuos con nombre, A es un nombre de concepto, C y D son conceptos y R, un rol. Tabla extraída de (Krötzsch *et al.*, 2012). 30

2.2 Lista de axiomas disponibles en $\mathcal{SROIQ}$ junto con su sintaxis y semántica. En ella a y b representan individuos con nombre, A es un nombre de concepto, C y D son conceptos y $R, S, R1$ y $R2$ son roles. Tabla extraída de (Krötzsch *et al.*, 2012). 31

3.1 Lista de diferencias relevantes para esta tesis entre BayesOWL, PR-OWL y OntoBayes 47

4.1 Tabla de probabilidad condicionada de la segunda versión de Turambar para un nodo A que tiene de padres el nodo C y B, teniendo todos ellos dos estados verdadero (T) y falso (F). 61

5.1 Satisfacción de un atom de consulta respecto una interpretación. Tabla extraída de (Sirin & Parsia, 2007). 81

6.1 Diferentes combinaciones de razonadores y frameworks semánticos utilizados para simular la toma de medicinas de un usuario. . . 91

6.2 Diferencia entre el tiempo medio de la consulta de contexto y el tiempo medio de consistencia en las etapas en segundos. 122

6.3 Memoria en MB utilizada de media en el experimento Turambar. . 122

La ciencia no es sino una perversión de sí misma a menos que tenga como objetivo final el mejoramiento de la humanidad

Nikola Tesla

CAPÍTULO

1

Introducción

Se estima que en el año 2030 la población mayor de 65 años representará el 23.5 % de la población total frente al 17.1 % del 2008 en la Europa de los 27(Giannakouris, 2010). Este factor ha contribuido a un incremento en el interés por el desarrollo de sistemas y aplicaciones que ayuden a aumentar la autonomía de la población de la tercera edad, así como la de las personas que sufren algún tipo de discapacidad. Por otra parte, la popularización de las nuevas tecnologías y la reducción del coste de las mismas han supuesto el impulso necesario para el desarrollo de los entornos inteligentes.

Los entornos inteligentes se definen como aquellos espacios en los que la computación se utiliza para facilitar las actividades diarias del usuario de manera transparente (Coen, 1998). Uno de sus elementos centrales es el cómo se modela el comportamiento del usuario y su razonamiento a lo largo del tiempo. Un correcto modelado y razonamiento permitirán responder adecuadamente a las necesidades del usuario en su actividad diaria, así como reaccionar adecuadamente a los cambios que se produzcan en el contexto. Se entiende por contexto a cualquier información que pueda ser utilizada para definir el estado de una entidad (Dey, 2001). Una entidad puede ser una persona, lugar o cualquier objeto que se considere importante en la interacción entre un usuario y una aplicación, incluidos el propio usuario y la aplicación.

Entre los métodos más populares de modelado de contexto destacan las ontologías (Strang & Linnhoff-Popien, 2004). Además, se han posicionado como técnica alternativa para el reconocimiento de actividades, especialmente en el área de reconocimiento de actividades basadas en objetos (Chen & Nugent, 2009). Estos hechos han contribuido a la aparición de diferentes ontologías para el modelado de entornos inteligentes, como SOUPA (Chen *et al.*, 2004) y CONON(Wang *et al.*, 2004).

La primera versión del lenguaje de ontologías OWL se publicó el 29 de julio de 2002 (Dean *et al.*, 2002). Desde entonces, se han publicado multitud de nuevas revisiones del mismo hasta conformar su última revisión el 10 de febrero de 2004 Schreiber *et al.* (2004). A pesar de las continuas revisiones que sufrió OWL 1, esta seguía presentando una serie de carencias (Grau *et al.*, 2008):

- Una expresividad insuficiente para modelar dominios complejos.
- Diferentes inconvenientes con su sintaxis.
- La ausencia de metamodelado en OWL 1 DL y OWL 1 Lite.
- Un sistema de importación de ontologías poco eficiente.
- La imposibilidad de usar propiedades de anotaciones en los axiomas OWL 1 DL.
- Los diversos problemas que presenta la semántica de OWL 1.
- La no existencia de una implementación de OWL 1 Full y su dificultad de llevarlo a la práctica.
- La expresividad de OWL 1 Lite puede ser confusa para los usuarios y su complejidad es ExpTime frente al NExp-Time de OWL 1 DL.
- La dificultad para determinar si una ontología OWL 1 era OWL 1 Lite, OWL 1 DL o OWL 1 Full.

Estas limitaciones provocaron que el 19 de diciembre de 2006 se propusiese OWL 1.1 (Patel-Schneider & Horrocks, 2006), sentando las bases de la versión

OWL 2, que sería publicada como recomendación de la W3C el 27 de octubre de 2009 (Group, 2009) y su revisión más actual (a fecha de escritura de esta tesis), el 11 de diciembre de 2012 (Group, 2012).

A pesar de las múltiples revisiones de OWL, su uso puede no ser el más adecuado cuando existe incertidumbre en el dominio (Lukasiewicz & Straccia, 2008). Este hecho se contrapone con la necesidad de soportar incertidumbre en entornos inteligentes (Riboni & Bettini, 2011; Almeida & López-de-Ipiña, 2012). Para solventar este problema, esta tesis pretende realizar tres contribuciones principales:

- Ofrecer un método para modelar el conocimiento con incertidumbre.
- La creación de un razonador que calcule aserciones probabilísticas en función del modelo.
- La definición de un lenguaje de consulta para ontologías con incertidumbre.

1.1 Motivación

Las ontologías OWL siguen la asunción de mundo abierto, según la cual no se puede garantizar la veracidad de un hecho si este no ha sido definido en la base de conocimiento. Por el contrario, en la asunción de mundo cerrado, todo hecho no definido como cierto es falso. Sirva de ejemplo la analogía entre las bases de datos y las ontologías (Baader & Nutt, 2003). Mientras que en una base de datos la información ausente se considera información falsa, en una ontología indicaría falta de conocimiento. El uso de ontologías implica asumir que el conocimiento disponible no es completo. Por otra parte, cuando el dominio de la información presenta incertidumbre, el uso de ontologías OWL puede no ser tan adecuado (Lukasiewicz & Straccia, 2008).

A la hora de hablar de la incertidumbre es muy importante definir qué se entiende como tal. La incertidumbre es generalmente considerada como los diferentes aspectos del conocimiento imperfecto, tales como la vaguedad o la incompletitud; el razonamiento con incertidumbre como la colección de métodos usados para modelar y razonar con conocimiento cuyos valores de verdad booleanos son desconocidos, incognoscibles o inaplicables(Laskey *et al.*, 2008). Sin embargo, otros autores

(Baader *et al.*, 2003; Lukasiewicz & Straccia, 2008) consideran que existen diferencias suficientes para distinguir entre el conocimiento con vaguedad e incertidumbre. Para estos investigadores, la incertidumbre en el conocimiento está formada por declaraciones que son verdaderas o falsas, de las cuales no se está seguro debido a la falta de conocimiento. En cambio, el conocimiento con vaguedad está compuesto por declaraciones que son ciertas hasta un determinado grado debido a nociones vagas. En esta tesis, la palabra incertidumbre se refiere a aquella producida por la falta de conocimiento.

Al igual que en otros dominios, la incertidumbre también se encuentra presente en los entornos inteligentes y afecta a la toma de decisiones. A la hora de realizar un acción, el sistema tiene que tener en cuenta información de diversas fuentes para poder tomar la correcta. Estas fuentes suelen estar formadas generalmente por sensores, actuadores, pequeños dispositivos y servicios que están sujetos a fallos. Algunos de los incidentes que pueden surgir son fallos en los sensores, que el dispositivo se quede sin batería o fuese olvidado por el usuario, problemas en servidores remotos, mala o nula conectividad de red, dispositivos que se estropean, etc. El acaecimiento de cualquiera de estos sucesos puede conllevar la imposibilidad de responder correctamente a las necesidades del usuario. Con el objetivo de ofrecer una información más amplia para la toma de decisiones, esta tesis propone un método para modelar la información con incertidumbre, así como un lenguaje para consultarla.

1.2 Hipótesis, objetivos y limitaciones

La hipótesis de esta disertación es la siguiente:

> *Es posible combinar el conocimiento de las ontologías con incertidumbre para la toma de decisiones en entornos inteligentes cuando existe ausencia de información requerida.*

Así mismo, el principal objetivo de esta investigación es:

La creación de un motor de razonamiento que combine las ontologías OWL con incertidumbre para tomar decisiones cuando exista ausencia de información requerida y ofrezca tiempos de respuesta adecuados para entornos inteligentes.

Lo que plantea los siguientes subobjetivos para la investigación:

- Objetivo 1. Estudiar el estado del arte y las diferentes propuestas relacionadas. Identificar nichos de investigación potenciales.
- Objetivo 2. Elegir la aproximación que se va a utilizar para resolver el problema de la incertidumbre del contexto.
- Objetivo 3. Definir una especificación para el modelado de ontologías con incertidumbre.
- Objetivo 4. Desarrollar un motor de razonamiento y de consultas que ayude a la toma de decisiones con incertidumbre.
- Objetivo 5. Validar los conceptos y aplicaciones presentados.

Se ha determinado dos características deseables que el razonador y las ontologías con incertidumbre deben de tener:

- Las ontologías con incertidumbre deben de poder ser reutilizadas por razonadores tradicionales, mediante el uso de una IRI única. Es decir, el conocimiento con incertidumbre debe estar lo más aislado posible del tradicional.
- Si para interpretar el conocimiento con incertidumbre fuese necesario el uso de software adicional que no fuese incluido por el razonador, este debería de configurarse de manera automática y transparente.

Finalmente, se consideran fuera del alcance de esta tesis las siguientes características:

- El aprendizaje de vocabularios o sistemas de representación que modelen el conocimiento con incertidumbre.

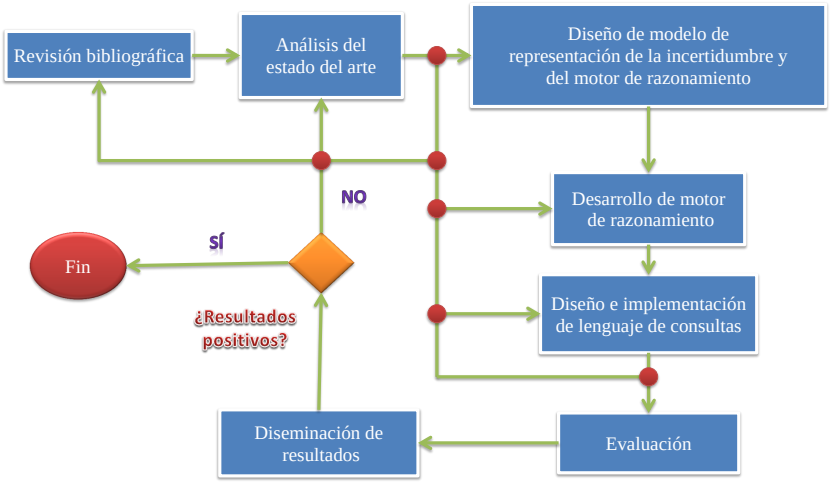


Figura 1.1: Representación gráfica de la metodología de investigación

- La comprobación de que los modelos con incertidumbre utilizados cumplan el propósito para el que han sido creados. Es decir, no se valida si una ontología con incertidumbre creada con un objetivo es capaz de cumplirlo. Por ejemplo, si una ontología con incertidumbre diseñada para determinar la localización de un usuario localiza correctamente al usuario.
- Aquellos tiempos de respuestas no relacionados con la interacción entre el usuario y la aplicación de monitorización. Entendiendo como tiempo de respuesta el tiempo necesario para responder una consulta probabilística.

1.3 Metodología de investigación

Con la finalidad de alcanzar los objetivos fijados en la sección anterior se plantea la metodología de investigación recogida en la figura 1.1 y que ha consistido en la realización de las siguientes actividades:

1. *Revisión bibliográfica de artículos relacionados con la incertidumbre en las ontologías y entornos inteligentes*. Esta actividad tiene como objetivo identificar aquellos artículos, proyectos y tecnologías relevantes para la tesis.

2. *Análisis del estado del arte*. Una vez identificado los recursos de interés se evalúan si son aplicables para el problema que se pretende abordar. Por otra parte, se analizan sus ventajas y desventajas frente a otras soluciones existentes.

3. *Diseño del modelo de representación de la incertidumbre y del motor de razonamiento*. A partir del análisis de la biografía se decide cómo se va a representar el conocimiento con incertidumbre. Por otra parte, se definirán las funcionalidades que debe de ofrecer el motor de razonamiento con incertidumbre.

4. *Desarrollo de un motor de razonamiento con incertidumbre*. Una vez decidido cómo se va a representar la información y qué tareas de razonamiento han de realizarse, se implementará un motor capaz de manejar conocimiento con incertidumbre.

5. *Diseño e implementación de un lenguaje declarativo de consultas*. El objetivo es ofrecer un lenguaje para consultar el conocimiento inferido por el razonador y así poder reutilizar este conocimiento en el proceso de toma de decisiones.

6. *Evaluación*. Se comprobará que los resultados obtenidos por el razonador sean correctos, así como su rendimiento.

7. *Diseminación de resultados*. Consistirá en diseminar las contribuciones obtenidas a congresos y revistas, en las cuales la comunidad científica proveerá comentarios constructivos sobre el avance y estado de la investigación.

Finalmente, cabe destacar que el desarrollo de estas actividades es iterativo, como se ve en la figura 1.1. Es decir, los resultados obtenidos en la evaluación y los comentarios recibidos en la diseminación podrán implicar la repetición parcial o total de algunas actividades con el objetivo de mejorar el trabajo realizado.

1.4 Estructura de la tesis

La tesis se organiza en siete capítulos, siendo el contenido de los restantes el siguiente:

- Capítulo 2 ofrece una introducción al mundo de las ontologías OWL, al mismo tiempo que se describe su base teórica. Además, se comentan los problemas que las ontologías tienen a la hora de gestionar las probabilidades.
- Capítulo 3 presenta y analiza las principales propuestas probabilísticas existentes para la gestión de la incertidumbre.
- Capítulo 4 describe la solución presentada para representar la incertidumbre, así como el tipo de inferencia que se puede esperar de ella.
- Capítulo 5 define el funcionamiento interno de cómo se responde a una consulta, así como el lenguaje de consultas ideado para acceder a la información probabilística.
- Capítulo 6 ofrece una evaluación del sistema propuesto. Adicionalmente, se exponen los experimentos que explican la razón de la elección de algunas de las herramientas utilizadas en el trabajo y no otras.
- Capítulo 7 expone las conclusiones generales de la tesis, resume sus principales contribuciones e indica cuales serían las líneas de trabajo futuro.

Inteligencia es el poder de aceptar el entorno

William Faulkner

CAPÍTULO

2

Antecedentes

Antes de comenzar el viaje, el explorador debe de preparar cuidadosamente su equipaje, seleccionando todos aquellos enseres que prevea que van a ser necesarios durante la larga aventura que le espera. En mi caso, el viaje es la tesis y los utensilios, aquellos conocimientos básicos que permitan dar los primeros pasos hacia la consecución de esta.

Con ese objetivo, este capítulo ofrece una breve introducción a las ontologías, así como a otros conceptos matemáticos relacionados con la incertidumbre.

2.1 El lenguaje de ontologías OWL

En el momento de escribir esta tesis, la versión actual de OWL es la versión 2 (Group, 2012). Esta revisión viene a extender las capacidades de la primera versión, además de solucionar los problemas descritos en el capítulo 1. Las siguientes subsecciones resumen las características más importantes de OWL 2 descritas en los diferentes documentos de la W3C (Group, 2012).

2.1.1 Sintaxis

La sintaxis principal de OWL para el intercambio de información es RDF/XML (Beckett, 2004). Esta sintaxis debe de ser soportada por las herramientas OWL 2 que cumplan la especificación (Horrocks *et al.*, 2012), aportando así la interoperabilidad entre aplicaciones.

Además de la sintaxis RDF/XML, existen otras sintaxis alternativas:

- OWL/XML (Patel-Schneider *et al.*, 2012a) define una sintaxis alternativa para RDF/XML y también hace uso de documentos XML (Gao *et al.*, 2009) para la definición de la ontología.
- Sintaxis Manchester (Horridge & Patel-Schneider, 2012) es una sintaxis de lectura más sencilla por parte de los usuarios.
- Turtle (Beckett *et al.*, 2014) es una sintaxis alternativa para la serialización de RDF.
- La sintaxis de estilo funcional puede ser usada para la serialización, aunque su propósito principal es especificar la estructura del lenguaje (Patel-Schneider *et al.*, 2012b).

2.1.2 Semántica

El significado de las ontologías OWL 2 viene definido por la semántica. En OWL 2 existen dos semánticas diferentes: la semántica directa (*Direct Semantics*) (Grau *et al.*, 2012) y la semántica basada en RDF (Schneider, 2012). Además existe un teorema que hace de enlace entre ambas semánticas.

La semántica basada en RDF asigna el significado directamente a grafos RDF y de manera indirecta a las estructuras de la ontología mediante los mapeos a grafos RDF. La semántica basada en RDF es completamente compatible con la semántica de RDF (Hayes, 2004), y extiende las condiciones semánticas definidas para RDF. Cualquier ontología OWL 2 puede ser dotada de significado mediante esta semántica.

Por su parte la semántica directa asigna el significado a las estructuras de la ontología, dando lugar a una semántica compatible con el modelo de la semántica

teórica de la lógica descriptiva $\mathcal{SROIQ}$ (Horrocks *et al.*, 2006). No obstante, las estructuras de las ontologías deben de cumplir una serie de condiciones determinadas en la especificación estructural de OWL 2 (Patel-Schneider *et al.*, 2012b) para poder ser traducidas a $\mathcal{SROIQ}$. Aquellas ontologías que cumplen estas condiciones sintácticas se denominan ontologías OWL 2 DL.

2.1.3 Perfiles en OWL 2

Los perfiles de OWL 2 pueden ser vistos como subconjuntos de OWL 2 que intentan ofrecer un punto intermedio entre expresividad y eficiencia de razonamiento (Motik *et al.*, 2012). A continuación se enumeran los nuevos perfiles añadidos:

- OWL 2 EL: es muy útil para aplicaciones que utilizan un gran número de clases y propiedades. Además, posee expresividad suficiente para una gran variedad de ontologías y el tiempo dedicado a comprobación de la consistencia y el emparejamiento de instancias a clases de una ontología es polinomial.
- OWL 2 QL: su uso está enfocado a las aplicaciones que utilicen una gran cantidad de datos y donde la respuesta a consultas es la tarea de razonamiento más importante.
- OWL 2 RL: su objetivo son las aplicaciones que necesitan un razonamiento escalable sin sacrificar expresividad. Esto se logra mediante la definición de un subconjunto sintáctico de OWL 2, sobre el que se pueden aplicar tecnologías basadas en reglas.

A la hora de elegir un perfil u otro es importante tener en cuenta lo siguiente:

- OWL 2 EL se debe utilizar con ontologías grandes y simples cuando se necesita un buen rendimiento.
- Si se requiere un perfil que pueda interactuar con sistemas de bases de datos relacionales y donde sea necesario un razonamiento escalable con grandes conjuntos de datos, se debe de considerar el uso de OWL 2 QL.
- OWL 2 RL se debe utilizar cuando el objetivo sea el razonamiento escalable de grandes conjuntos de datos y exista la necesidad de integrar motores de reglas y sistemas de gestión de bases de datos con reglas.

2.1.4 Estructura conceptual de OWL 2

La estructura de OWL 2 es definida en la recomendación del W3C denominada *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition)* (Patel-Schneider *et al.*, 2012b). Una ontología OWL 2 es una descripción formal de un dominio de interés. Las ontologías son identificadas mediante una IRI (Duerst & Suignard, 2005) y adicionalmente puede tener una IRI adicional para identificar su versión. Las ontologías existentes pueden ser reutilizadas por otras mediante el uso de *imports*.

Las ontologías OWL 2 pueden referenciar diferentes tipos de datos (*datatypes*) como valores numéricos, cadenas de texto, datos binarios, valores booleanos, IRIs, valores temporales y literales en XML. Existen tres categorías sintácticas diferentes para expresar la parte lógica de las ontologías OWL 2: entidades, expresiones y axiomas.

Las entidades son identificados por IRIs. En OWL 2, una IRI puede utilizarse para referirse a más de un tipo de entidades. Existen los siguientes tipos de entidades:

- Clases para representar conjuntos de individuos. Además son un tipo de expresión de clase (*class expression*).
- Propiedades de objeto (*object properties*) y propiedades de datos (*data properties*) representan relaciones en el dominio.
- Tipos de datos (*datatypes*) son un conjunto de literales como números o cadenas de texto. A su vez, son también un tipo de rango de datos (*data range*).
- Propiedades de anotaciones asocian información sin implicaciones lógicas a ontologías, axiomas y entidades.
- Individuos con nombre (*named individuals*) son individuos con una IRI global. Junto con los individuos anónimos (aquellos individuos locales a la ontología y sin IRI global) forman los individuos de la ontología: los objetos del dominio.

A parte de las entidades, existen los individuos anónimos que son similares a los nodos en blanco de RDF y los literales, que son la representación de un valor de un tipo de datos. Los rangos de datos se utilizan como restricciones para los valores de propiedades de datos. Además de los tipos de datos, existen varios tipos de rangos de datos:

- Intersecciones de rango de datos.
- Uniones de rango de datos.
- Complemento de rango de datos.
- Enumeración de literales.
- Restricciones de tipo de datos.

Por su parte, las expresiones representan nociones complejas del dominio descrito. Las expresiones de clases y propiedades se suelen utilizar para construir expresiones de clases, también llamadas descripciones o conceptos complejos. Existen diferentes tipos de expresiones de clases, siendo la más sencilla las ya mencionadas clases. Las otras expresiones de clases son las siguientes:

- *Intersecciones de expresiones de clases* representan a aquellos individuos que son instancias de varias expresiones de clases a la vez. Por ejemplo, se pueden utilizar para definir los individuos que son familiares y cuidadores a la vez.

  ```
  ObjectIntersectionOf( a:Familiar a:Cuidador )
  ```

- *Uniones de expresiones de clases* describen a aquellos individuos que son instancias de una expresión de clases, dado un conjunto de expresiones de clases. Por ejemplo, se pueden utilizar para definir los individuos que son familiares o cuidadores de un enfermo.

  ```
  ObjectUnionOf( a:Familiar a:Cuidador )
  ```

- *Complemento de expresiones de clases* definen a aquellos individuos que no son instancias de una expresión de clases. Por ejemplo, se pueden utilizar para definir al conjunto de individuos que no son cuidadores.

  ```
  ObjectComplementOf( a:Cuidador )
  ```

- *Enumeraciones de individuos* enumeran los individuos que son instancias de una clase. Por ejemplo, pueden definir los colores primarios de la luz.

  ```
  ObjectOneOf( a:Rojo a:Azul a:Verde)
  ```

- *Expresiones de cuantificación existencial* describen a aquellos individuos que están conectados mediante una expresión de propiedad de objeto a otro individuo que es una instancia de una expresión de clase. Existe su equivalente para expresiones de propiedades de datos, en las que la expresión de propiedades de objetos es sustituida por la expresión de propiedades de datos y la expresión de clases por el rango de datos. Por ejemplo, para definir a aquellos individuos que cuidan a un paciente.

  ```
  ObjectSomeValuesFrom ( a:cuida a:Paciente)
  ```

  En el caso de expresiones de propiedades de datos se podrían utilizar para definir aquellos individuos que son menores de edad:

  ```
  DataSomeValuesFrom(a:tieneEdad DatatypeRestriction(
   xsd:integer xsd:maxExclusive "18"^^xsd:integer ) )
  ```

- *Expresiones de cuantificación universal* definen a aquellos individuos en las que todas sus conexiones mediante una expresión de propiedad de objeto a otros individuos que son instancias de una expresión de clase. Existe su equivalente para expresiones de propiedades de datos, en las que la expresión de propiedades de objetos es sustituida por la expresión de propiedades de datos y la expresión de clases por el rango de datos. Un ejemplo de uso es la definición de aquellos individuos que solo cuidan a personas.

  ```
  ObjectAllValuesFrom( a:cuidan a:Persona )
  ```

En el caso de las expresiones de propiedades de datos, se podrían utilizar para definir a aquellos individuos cuyo identificador de paciente sea un número entero.

```
DataAllValuesFrom( a:identificadorPaciente
 xsd:integer )
```

- *Restricciones de valor de individuo* describen a los individuos que están conectados con otro individuo mediante una expresión de propiedad de objeto. Existe su equivalente para expresiones de propiedades de datos, *restricciones de valor de literal*, en las que la expresión de propiedades de objetos es sustituida por la expresión de propiedades de datos y el individuo por un literal. Por ejemplo, para definir los individuos que viven en Bilbao.

  ```
  ObjectHasValue( a:viveEn a:Bilbao )
  ```

  En el caso de las restricciones de valor de literal, se podría utilizar para definir a aquellos individuos que acaban de alcanzar la edad legal para votar.

  ```
  DataHasValue( a:tieneEdad "18"^^xsd:integer )
  ```

- *Restricciones reflexivas* describen el conjunto de individuos que está conectado consigo mismo mediante una expresión de propiedad de objeto. Se pueden usar para definir a aquellos individuos que se alimentan solos.

  ```
  ObjectHasSelf( a:alimenta )
  ```

- *Cardinalidad mínima* define a aquellos individuos que se conectan un número mínimo de veces mediante una expresión de propiedades de objetos con individuos que son instancias de una expresión de clase. Existe su equivalente para expresiones de propiedades de datos, en las que la expresión de propiedades de objetos es sustituida por la expresión de propiedades de datos y la expresión de clases por el rango de datos. Por ejemplo, para definir a aquellos individuos que tienen al menos dos familiares.

  ```
  ObjectMinCardinality( 2 a:tiene a:Familiar )
  ```

En el caso de las expresiones de propiedades de datos, se podrían usar para clasificar a aquellos individuos que tienen múltiples nombres.

```
DataMinCardinality( 2 a:tieneNombre )
```

- *Cardinalidad máxima* define a aquellos individuos que se conectan un número máximo de veces mediante una expresión de propiedades de objetos con individuos que son instancias de una expresión de clase. Existe su equivalente para expresiones de propiedades de datos, en las que la expresión de propiedades de objetos es sustituida por la expresión de propiedades de datos y la expresión de clases por el rango de datos. Por ejemplo, para definir a los individuos que tienen como máximo dos cuidadores.

  ```
  ObjectMaxCardinality( 2 a:tieneCuidador )
  ```

  En el caso de las expresiones de propiedades de datos, se podrían usar para clasificar a aquellos individuos que tienen como máximo dos nombres.

  ```
  DataMaxCardinality( 2 a:tieneNombre )
  ```

- *Cardinalidad exacta* define a aquellos individuos que se conectan un número exacto de veces mediante una expresión de propiedades de objetos con individuos que son instancias de una expresión de clase. Existe su equivalente para expresiones de propiedades de datos, en las que la expresión de propiedades de objetos es sustituida por la expresión de propiedades de datos y la expresión de clases por el rango de datos. Por ejemplo, para definir aquellos individuos que tienen dos cuidadores.

  ```
  ObjectExactCardinality( 2 a:tieneCuidador )
  ```

  En el caso de las expresiones de propiedades de datos, se podrían usar para clasificar a aquellos individuos que tienen dos nombres.

  ```
  DataExactCardinality( 2 a:tieneNombre )
  ```

Los axiomas son declaraciones que son aseveradas como ciertas en el dominio que se describe. Obviando las declaraciones y los axiomas de anotación, se pueden clasificar en los siguientes grupos:

- *Axiomas de expresiones de clases* definen relaciones entre expresiones de clases:
  - *Axiomas de subclases* define una expresión de clase como subclase de otra expresión de clase. Por ejemplo, para definir que un hijo es un tipo de familiar.

    ```
    SubClassOf( a:Hijo a:Familiar )
    ```

  - *Clases equivalentes* describe que un conjunto de expresiones de clases son equivalentes. Por ejemplo, para expresar que las clases padre y papá son los mismos.

    ```
    EquivalentClasses( a:Hijo a:Familiar )
    ```

  - *Clases disjuntas* establece un conjunto de expresiones de clases disjuntas entre ellas. Por ejemplo, para expresar que las clases padre y madre no tienen individuos en común.

    ```
    DisjointClasses( a:Padre a:Madre )
    ```

  - *Unión disjunta de expresiones de clases* especifica que una expresión de clase es disjunta de la unión de un conjunto de expresiones de clases. Por ejemplo, para expresar que los progenitores son o bien padres o bien madres, pero no pueden ser padre y madre a la vez.

    ```
    DisjointUnion (a:Progenitor a:Padre a:Madre )
    ```

- *Axiomas de propiedades de objeto* establecen relaciones entre expresiones de propiedades de objeto.
  - *Subpropiedades de objeto* define que una expresión de propiedad de objeto es una subpropiedad de otra expresión de clase de objeto; tal que si un individuo está conectado por una expresión de propiedad de objeto a otro individuo, este también lo estará por la otra expresión. Por ejemplo, para expresar que la propiedad estaCenando es una subpropiedad de estaIngiriendo.

    ```
    SubObjectPropertyOf( a:estaCenando
    a:estaIngiriendo )
    ```

- *Propiedades de objeto equivalentes* establece que un conjunto de expresiones de propiedades de objeto son equivalentes. Por ejemplo, para definir que la tomarPastillas es lo mismo que ingerirPastillas.

  ```
  EquivalentObjectProperties(
  a:tomarPastillas a:ingerirPastillas )
  ```

- *Propiedades de objeto disjuntas* define un conjunto de expresiones disjuntas entre sí. Por ejemplo, para indicar que el individuo A que cuida al individuo B no puede ser cuidado por este.

  ```
  DisjointObjectProperties( a:cuida a:esCuidado )
  ```

- *Propiedades de objeto inversas* define que una expresión de propiedad de objeto es la inversa de otra. Por ejemplo, para definir que si un individuo A cuida a B, entonces B cuida a A.

  ```
  InverseObjectProperties( a:cuida a:esCuidado )
  ```

- *Dominio de propiedades de objeto* define que el dominio de una expresión de propiedades de objeto es una expresión de clase. Por ejemplo, para definir que los individuos que cuidan son cuidadores.

  ```
  ObjectPropertyDomain( a:cuida a:Cuidador )
  ```

- *Rango de propiedades de objeto* define que el rango de una expresión de propiedades de objeto es igual a una expresión de clase. Por ejemplo, para definir que si un individuo A cuida a otro individuo, este último será entonces un paciente.

  ```
  ObjectPropertyRange( a:cuida a:Paciente )
  ```

- *Propiedades de objeto funcionales* define que un individuo solo puede tener una única relación con una expresión de propiedad de objetos. Por ejemplo, para indicar que un individuo solo tiene un doctor.

  ```
  FunctionalObjectProperty( a:tieneDoctor )
  ```

- *Propiedades de objetos funcional inversa* indica que la inversa de una expresión de propiedad de objeto es funcional.. Por ejemplo, para definir que el doctor tiene muchos pacientes, pero que los pacientes solo tienen un doctor.

```
InverseFunctionalObjectProperty( a:esDoctor )
```

– *Propiedades de objeto reflexivas* indican que una expresión de propiedad de objetos es reflexiva. Es decir, todos los individuos están conectados así mismo mediante una expresión de propiedad de objeto. Por ejemplo, para definir que todo individuo se conoce a sí mismo.

```
ReflexiveObjectProperty( a:conoce )
```

– *Propiedades de objeto irreflexivas* indican que un individuo no puede relacionarse consigo mismo mediante una expresión de propiedad de objeto. Por ejemplo, para indicar que un paciente no se puede cuidarse a sí mismo

```
IrreflexiveObjectProperty( a:cuida )
```

– *Propiedad de objeto simétrica* define que si un individuo A conectado con otro individuo B mediante una expresión de propiedad de objeto, entonces B también estará conectado mediante esa misma expresión con A. Por ejemplo, para indicar que si un individuo A es familiar de un individuo B, entonces B también será familiar de A.

```
SymmetricObjectProperty( a:esFamiliarDe )
```

– *Propiedad de objeto asimétric*a define que si un individuo A está conectado con otro individuo B mediante una expresión de propiedad de objeto, B no puede estar conectado con A. Por ejemplo, para definir que si un individuo A es padre de B, entonces B no puede ser padre de A.

```
AsymmetricObjectProperty ( a:esFamiliarDe )
```

– Propiedades de objeto transitiva describen que si el individuo A está conectado con el individuo B mediante una expresión de propiedades de objeto y el individuo B con esa misma expresión al individuo C, entonces A también estará conectado a C. Por ejemplo, para definir que si una caja A contiene a otra caja B que contiene un medicamento, entonces la caja A también contiene el medicamento.

```
TransitiveObjectProperty  ( a:esFamiliarDe )
```

- *Axiomas de propiedades de tipos de datos* son similares a los axiomas con propiedades de objeto, solo que con tipos de datos. Existen los siguientes tipos:

  - *Subpropiedades de datos* establecen que todo individuo que este conectado por una subpropiedad de una expresión de propiedades de datos a un literal, también lo estará por la expresión padre.Por ejemplo, para definir que la propiedad iluminando es una subpropiedad de encendido.

    ```
    SubDataPropertyOf( a:iluminando a:encendido )
    ```

  - *Propiedades de datos equivalentes* especifican que un conjunto expresiones de propiedades de datos son semánticamente equivalentes. Por ejemplo, para definir que la propiedad se llama es la misma que la propiedad tiene de nombre.

    ```
    EquivalentDataProperties( a:seLlama
    a:tieneDeNombre )
    ```

  - *Propiedades de datos disjuntas* definen que si una expresión de propiedad del conjunto de expresiones de propiedades de datos conecta a un individuo con un literal, ninguna otra expresión de ese conjunto puede conectarle con ese mismo literal. Por ejemplo, para definir que la contraseña y el nombre de un usuario no pueden ser el mismo texto.

    ```
    DisjointDataProperties( a:nombreUsuario
     a:password )
    ```

  - *Dominio de propiedades* de datos establece el conjunto de expresiones de clase que forman parte del dominio de una expresión de propiedad de datos. Por ejemplo, para definir que las personas tienen DNI.

    ```
    DataPropertyDomain( a:tieneDNI a:Persona )
    ```

  - *Rango de propiedad de datos* describe el rango de datos que puede tomar una expresión de propiedades de datos. Por ejemplo, para definir que el nombre de una persona debe de ser del tipo string.

    ```
    DataPropertyRange( a:nombre xsd:string )
    ```

  - *Propiedades de datos funcionales* definen que un individuo solo puede tener una única relación con una expresión de propiedad de datos. Por ejemplo, para definir que un individuo solo puede tener un nombre.

    ```
    FunctionalDataProperty( a:tieneNombre )
    ```

- *Definición de tipos de datos* permite definir nuevos tipos de datos que sean semánticamente equivalentes a un rango de datos. Por ejemplo, para definir el patrón de una licencia software.

```
Declaration( Datatype( a:Licencia ) )
DatatypeDefinition(
a:Licencia
DatatypeRestriction( xsd:string xsd:pattern
 "[0-9]{3}-[0-9]{3}-[0-9]{4}" )
)
```

- *Claves* (*keys*) sirven para especificar que las instancias de una expresión de clase no pueden estar conectadas mediante una expresión de propiedad de objetos a un individuo. Lo mismo es aplicable para las expresiones de propiedades de datos. Por ejemplo, para definir que la claves de activación de los programas son únicas.

```
HasKey( owl:Thing () ( a:tieneLicencia ) )
```

- *Aserciones son axiomas* sobre individuos a los cuales se les denomina hechos. Existen los siguientes tipos de aserciones:

  - *Igualdad individual* identifica que un conjunto de individuos son iguales. Por ejemplo, para definir que dos ciudades son las mismas.

    ```
    SameIndividual( a:Bilbao a:Bilbo )
    ```

  - *Desigualdad individua*l describe un conjunto de individuos como diferentes. Por ejemplo, para definir que dos ciudades son diferentes.

    ```
    DifferentIndividuals(a:Madrid_ESP a:Madrid_USA)
    ```

  - *Aserciones de clase* indican que un individuo es miembro de una expresión de clases. Por ejemplo, para declarar que John es un paciente.

```
ClassAssertion( a:Paciente a:john )
```

- *Aserciones positivas de propiedades de objeto* establecen que un individuo está conectado con otro mediante una expresión de propiedad de objeto. Por ejemplo, para definir que dos individuos son amigos.

  ```
  ObjectPropertyAssertion(a:amigo a:John a:David)
  ```

- *Aserciones negativas de propiedades de objeto e*stablecen que un individuo no está conectado con otro mediante una expresión de propiedad de objeto. Por ejemplo, para definir que dos individuos no son amigos.

  ```
  NegativeObjectPropertyAssertion( a:amigo a:John
   a:David )
  ```

- *Aserciones positivas de propiedades de datos* establecen que un individuo está conectado con un literal mediante una expresión de propiedad de datos. Por ejemplo, para definir la edad de un individuo.

  ```
  DataPropertyAssertion( a:edad a:john
  "87"^^xsd:integer )
  ```

- *Aserciones negativas de propiedades de datos* establecen que un individuo no está conectado con un literal mediante una expresión de propiedad de datos. Se puede utilizar para definir que un individuo no tiene una cierta edad.

  ```
  NegativeDataPropertyAssertion( a:edad a:John
  "18"^^xsd:integer )
  ```

Finalmente es importante destacar que tanto las ontologías, los axiomas y entidades pueden ser anotadas. Las anotaciones añaden información adicional a las ontologías que puede ser utilizada por las aplicaciones que usen las ontologías. Sin embargo, esta no tiene ninguna implicación lógica. Las entidades, axiomas y expresiones definidos en una ontología *B* pueden ser accedida por una ontología *A*, si esta importa a la ontología *B*. Por ejemplo, se podría usar una anotación para añadir un comentario sobre una clase de la ontología.

```
AnnotationAssertion( rdfs:label a:Paciente
 "Representa a las personas que se van a cuidar." )
```

2.2 Lógicas descriptivas

La siguiente sección es una breve introducción a las lógicas descriptivas y se basa principalmente en los siguientes artículos: (Baader & Nutt, 2003), (Krötzsch *et al.*, 2012) y (Krotzsch *et al.*, 2014). Las lógicas descriptivas son una familia de formalismos para la representación del conocimiento. Su objetivo es representar el conocimiento del dominio de una aplicación mediante la definición de los conceptos relevantes, para posteriormente utilizarlos en la definición de las propiedades de los objetos e individuos que pertenecen a este (Baader & Nutt, 2003). Sus características más destacadas son que poseen una semántica formal basada en la lógica y su énfasis en el razonamiento como servicio central. El razonamiento permite inferir el conocimiento implícito a partir del conocimiento que está explícitamente definido en la base de conocimiento. Los hechos declarados en una base de conocimiento se les denomina axiomas y pueden clasificarse en tres grupos ABox para las aserciones, TBox para la terminología y RBox para las relaciones entre roles (Krotzsch *et al.*, 2014).

$\mathcal{SROIQ}$ (Horrocks *et al.*, 2006) es la lógica descriptiva que sirve de base para OWL 2 DL y forma parte familia de lenguajes $\mathcal{AL}$. En $\mathcal{AL}$ la descripción de un concepto se forma de la siguiente manera:

$$
\begin{aligned}
C, D \longrightarrow\ & A| \\
& \top| \\
& \bot| \\
& \neg A| \\
& \forall R.C| \\
& \exists R.\top| \\
& C \sqcap D
\end{aligned}
$$

En la expresión anterior A es un concepto atómico y R es un rol atómico, mientras que C y D son descripciones de conceptos. El símbolo $\bot$ equivale al concepto vacío y $\top$ al concepto universal. $C \sqcap D$ representan la intersección de C y D, $\neg A$ es la negación atómica del concepto A. Finalmente $\forall R.C$ representa una

restricción de valor y $\exists R.\top$ es la cuantificación existencial limitada. La negación solo se aplica a conceptos atómicos, y solo se permite el concepto universal en el ámbito de una cuantificación existencial sobre un rol.

Por otra parte, se denomina $\mathcal{FL}^-$ al sublenguaje de $\mathcal{AL}$ que se obtiene al deshabilitar la negación atómica y a los sublenguajes de $\mathcal{FL}^-$ que se obtienen al deshabilitar la cuantificación existencial limitada, $\mathcal{FL}_0$. Los lenguajes que permiten la intersección de conceptos y la cuantificación existencial, pero no la restricción de valores forman la familia $\mathcal{EL}$. Esta familia provee los constructores de intersección de conceptos y la cuantificación existencial.

Añadiendo nuevos constructores a $\mathcal{AL}$ se pueden obtener lenguajes más expresivos:

- La unión de conceptos ($\mathcal{U}$).
- La cuantificación existencial completa ($\mathcal{E}$).
- Restricciones numéricas ($\mathcal{N}$). Se escriben como $\geqslant nR$y $\leqslant nR$, la primera representa la restricción de al menos *n* y la segunda como máximo *n*, donde *n* es un número entero no negativo.
- La negación de conceptos arbitrarios ($\mathcal{N}$).

Utilizando un subconjunto de los anteriores constructores se obtiene un lenguaje $\mathcal{AL}$ determinado. Las lenguas que se pueden formar se nombran de la siguiente manera $\mathcal{AL}[\mathcal{U}][\mathcal{E}][\mathcal{N}][\mathcal{C}]$. No obstante es posible describir todos los lenguajes únicamente con $\mathcal{U}, \mathcal{E}, \mathcal{N}$, ya que mediante el uso de la unión y la cuantificación existencial completa se puede expresar la negación de conceptos. También se da por hecho que todo lenguaje que contenga la unión y la cuantificación existencial completa contiene la negación y viceversa. Por tanto, el número total de lenguajes obtenidos son ocho entre los que no existen parejas equivalentes. En vez de usar la letras $\mathcal{UE}$ para los lenguajes que contienen la unión y la cuantificación existencial completa, se utiliza $\mathcal{C}$.

Se podría decir que cada letra que forma parte del lenguaje identifica las características del mismo. En cuanto al caso concreto de $\mathcal{SROIQ}$, la $\mathcal{S}$ describe la abreviación de $\mathcal{ALC}$ con roles transitivos; la $\mathcal{R}$, la presencia de inclusión de roles,

reflexividad local, el rol universal U, transitividad, simetría, asimetría, roles disjuntos, reflexividad e irreflexividad; $\mathcal{O}$, el uso de nominales; $\mathcal{I}$ roles inversos; $\mathcal{Q}$ restricciones numéricas calificadas.

Por lo tanto, en $\mathcal{SROIQ}$ existen los siguientes axiomas que se suelen representar de la siguiente forma:

- *Axiomas del ABox* expresan hechos sobre individuos.
  - *Aserciones de conceptos*. Definen la pertenencia de un individuo a una clase. El siguiente axioma define que John es un usuario.

$$Usuario(John) \tag{2.1}$$

  - *Aserciones de roles*. Describen la relación entre dos individuos. Por ejemplo, para expresar que John vive en Bilbao.

$$viveEn(John, Bilbao) \tag{2.2}$$

  - *Aserción de desigualdad entre individuo*s. Dos individuos no tienen por qué ser diferentes por tener diferentes nombres, ya que en las lógicas descriptivas no se sigue la asunción de nombre único. Si se desease expresar que John y Bilbao son individuos diferentes, entonces habría que declararlo explícitamente.

$$John \not\approx Bilbao \tag{2.3}$$

  - *Aserción de igualdad entre individuos*. Dos nombres diferentes de individuos pueden hacer referencia al mismo individuo. Si se desease expresar que Bilbo y Bilbao son la misma ciudad.

$$Bilbo \approx Bilbao \tag{2.4}$$

- *Axiomas del TBox* describen relaciones entre conceptos:
  - *Inclusión de conceptos*. Expresa que todos los individuos miembros de un concepto son miembros de otro concepto. Por ejemplo, todos los individuos de la clase cocina, son miembros de la clase habitación.

$$Cocina \sqsubseteq Habitación \tag{2.5}$$

  - *Equivalencia de conceptos*. Define que dos conceptos son iguales. Si se quiere expresar que el concepto usuario y paciente son en realidad el mismo concepto.

$$Usuario \equiv Paciente \tag{2.6}$$

- *Axiomas de RBox* definen relaciones entre roles.

  - *Inclusión de roles*. Define que todos los individuos que se relacionan mediante un rol *A*, también estarán relacionados por el rol *B*, pero no todos los individuos relacionados por el rol *B* tienen que estar relacionados también por el rol *A*. El siguiente ejemplo muestra al rol *estaDesayunando* como un subrol de *estaComiendo*.

$$estaDesayunando \sqsubseteq estaComiendo \tag{2.7}$$

    Existe una forma compleja de inclusión de roles que hace uso de la composición de roles. La composición de roles permite expresar que si un individuo *A* está relacionado por un rol *R1* con un individuo *B* y este está relacionado con otro individuo *C* mediante un rol *R2*, entonces *A* está relacionado con *C* mediante el rol *R3*. Por ejemplo, si se desea definir que si un cuidador atiende a un usuario que vive en un lugar, entonces ese cuidador trabaja en ese lugar.

$$atiendeA \circ viveEn \sqsubseteq trabajaEn \tag{2.8}$$

  - *Equivalencia de roles*. Expresa que dos roles son iguales. Por ejemplo, el rol *irA* es igual al rol *desplazarseA*.

$$irA \equiv desplazarseA \tag{2.9}$$

  - *Roles disjuntos*. Indican que un individuo no puede estar relacionado con otro mediante un rol *R1* y otro rol *R2* a la vez. Para expresar que un individuo no puede cuidar y ser cuidado por la misma persona.

$$Disjoint(cuida, esCuidado) \tag{2.10}$$

– *Transitividad de roles*. Es una forma especial de la inclusión compleja de roles. Describe que si un individuo A está relacionado mediante *R* con otro individuo *B* y *B* está relacionado con *C* mediante *R*, entonces *A* también estará relacionado con *C* mediante *R*. Un ejemplo, los cajones contienen todos los objetos almacenados en las cajas que están dentro de ellos.

$$contiene \circ contiene \sqsubseteq contiene \tag{2.11}$$

– *Rol simétrico*. Expresa que si un individuo está relacionado mediante un rol con otro individuo, el segundo individuo también estará relacionado con el primer individuo mediante el mismo rol. Por ejemplo, si John está casado con Laura, Laura también estará casada con John. En otras palabras el rol tiene su equivalencia en su inverso.

$$casado \equiv casado^{-} \tag{2.12}$$

– *Rol asimétrico*. Expresa que si un individuo *A* está relacionado con un individuo *B* mediante un rol, no puede existir una relación mediante ese rol entre *B* y *A*. Por ejemplo, los individuos que cuidan a usuarios no pueden ser cuidados por aquellos a los que cuidan.

$$Disjoint(cuida, cuida^{-}) \tag{2.13}$$

– *Reflexividad global*. Define que todos los individuos están relacionados consigo mismo mediante un rol. Si se desea definir que todo individuo se conoce así mismo.

$$\top \sqsubseteq \exists conoce.Self \tag{2.14}$$

– *Rol irreflexivo*. Indica que un individuo no puede relacionarse consigo mismo mediante un rol. Por ejemplo, para definir que un individuo no puede cuidarse así mismo.

$$\top \sqsubseteq \neg \exists cuida A.Self \tag{2.15}$$

Las lógicas descriptivas cuentan, además, con un conjunto de constructores de conceptos y roles que permiten la definición de conceptos y roles complejos:

- *Intersección de conceptos*. Sirve para definir un concepto como el conjunto de individuos que son miembros de un concepto *A* y un concepto *B* al mismo tiempo. Por ejemplo, para definir el conjunto de individuos que son familiares y que cuidan de un individuo.

$$Familiar \sqcap Cuidador \tag{2.16}$$

- *Unión de conceptos*. Definen el conjunto de individuos que son miembros de un concepto *A* o *B*. Por ejemplo, para definir los individuos que son cuidadores o familiares.

$$Familiar \sqcup Cuidador \tag{2.17}$$

- *Negación de conceptos o complemento*. Expresan al conjunto de individuos que no son miembros de un concepto. Para definir los individuos que no son usuarios.

$$\neg Usuarios \tag{2.18}$$

- *Concepto universal*. Engloba todos los individuos que existe.

$$\top \sqsubseteq \neg Usuarios \sqcup Usuarios \tag{2.19}$$

- *Concepto vacío*. Es el conjunto sin individuos.

$$\neg Usuarios \sqcap Usuarios \sqsubseteq \bot \tag{2.20}$$

- *Restricción existencial*. Define el conjunto de individuos que están relacionados con al menos un individuo perteneciente a otro concepto. Si se desea definir al conjunto de individuos que cuidan al menos a un Usuario.

$$\exists cuidan.Usuario \tag{2.21}$$

- *Restricción universal*. Define el conjunto de individuos que están relacionados mediante un rol únicamente con individuos pertenecientes a un concepto. Si se desea definir al conjunto de individuos que solo cuidan hombres.

$$\forall cuidan.Hombre \tag{2.22}$$

  Es importante recordar que si un individuo no cuidase a ningún otro individuo, este cumpliría con la definición del concepto.

- *Restricciones numéricas*. Establecen el número de relaciones que puede tener un individuo mediante un rol con los miembros de un conjunto. Por ejemplo, se puede definir como dormitorio doble a aquella habitación que contenga exactamente dos camas.

$$HabitaciónDoble \equiv \geqslant 2contiene.Cama \sqcap \leqslant 2tiene.Cama \tag{2.23}$$

- *Reflexividad local*. Expresa los individuos que se relacionan consigo mismos mediante un rol. Por ejemplo, para definir los individuos que comen solos.

$$\exists daDeComer.Self \tag{2.24}$$

- *Nominales*. Permiten la definición de conceptos mediante la enumeración de las instancias que forman parte del concepto. Se puede utilizar para definir los días de la semana.

$$\begin{aligned} DíasSemana \equiv \{Lunes\} \sqcup \{Martes\} \sqcup \{Miércoles\} \\ \sqcup \{Jueves\} \sqcup \{Viernes\} \sqcup \{Sábado\} \sqcup \{Domingo\} \end{aligned} \tag{2.25}$$

- *Rol inverso*. Permite declarar que un rol es el inverso de otro rol. Por ejemplo, para declarar que el rol ***esCuidadoPor*** es el inverso de *cuidaA*.

$$esCuidadoPor \equiv cuidaA^{-} \tag{2.26}$$

- *Rol universal* U. Relaciona todas las parejas de individuos.

Constructores		
	Sintaxis	Semántica
Individuos		
Individuos con nombre	a	$a^{\mathcal{I}}$
Roles		
Rol atómico	R	$R^{\mathcal{I}}$
Rol inverso	R^{-}	$\{\langle x, y\rangle \mid \langle y, x\rangle \in R^{\mathcal{I}}\}$
Rol universal	U	$\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Conceptos		
Concepto atómico	A	$A^{\mathcal{I}}$
Intersección	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Unión	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Complemento	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
Concepto universal	$\top$	$\Delta^{\mathcal{I}}$
Concepto vacío	$\bot$	$\emptyset$
Restricción existencial	$\exists R.C$	$\{x \mid$ algún sucesor $R^{\mathcal{I}}$ de x en $C^{\mathcal{I}}\}$
Restricción universal	$\forall R.C$	$\{x \mid$ todos los sucesores $R^{\mathcal{I}}$ de x en $C^{\mathcal{I}}\}$
Restricción al menos	$\geqslant R.C$	$\{x \mid$ al menos n sucesores de $R^{\mathcal{I}}$ de x en $C^{\mathcal{I}}\}$
Restricción como máximo	$\leqslant R.C$	$\{x \mid$ como mucho n sucesores de $R^{\mathcal{I}}$ de x en $C^{\mathcal{I}}\}$
Reflexividad local	$\exists R.Self$	$\{x \mid \langle x, x\rangle \in R^{\mathcal{I}}\}$
Nominal	$\{a\}$	$\{a^{\mathcal{I}}\}$

Cuadro 2.1: Lista de constructores disponibles en $\mathcal{SROIQ}$ junto con su sintaxis y semántica. En ella a, representa individuos con nombre, A es un nombre de concepto, C y D son conceptos y R, un rol. Tabla extraída de (Krötzsch *et al.*, 2012).

Axiomas		
	Sintaxis	Semántica
ABox		
Aserción de concepto	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
Aserción de rol	$R(a.b)$	$\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$
Igualdad de individuos	$a \approx b$	$a^{\mathcal{I}} = b^{\mathcal{I}}$
Desigualdad de individuos	$a \not\approx b$	$a^{\mathcal{I}} \neq b^{\mathcal{I}}$
TBox		
Inclusión de conceptos	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq B^{\mathcal{I}}$
Equivalencia de conceptos	$C \equiv D$	$C^{\mathcal{I}} = B^{\mathcal{I}}$
RBox		
Inclusión de roles	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
Equivalencia de roles	$R \equiv S$	$R^{\mathcal{I}} = S^{\mathcal{I}}$
Inclusión compleja de roles	$R1 \circ R2 \sqsubseteq S$	$R1^{\mathcal{I}} \circ R2^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
Roles disjuntos	$Disjoint(R, S)$	$R^{\mathcal{I}} \cap S^{\mathcal{I}} = \emptyset$

Cuadro 2.2: Lista de axiomas disponibles en $\mathcal{SROIQ}$ junto con su sintaxis y semántica. En ella a y b representan individuos con nombre, A es un nombre de concepto, C y D son conceptos y $R, S, R1$ y $R2$ son roles. Tabla extraída de (Krötzsch *et al.*, 2012).

Las tablas 2.2 y 2.1 resumen la sintaxis y semántica de los axiomas y constructores de $\mathcal{SROIQ}$.

Como se puede observar existe una relación bastante evidente entre OWL 2 DL y $\mathcal{SROIQ}$, pudiéndose considerar la primera una variante sintáctica del lenguaje descriptivo (Krotzsch *et al.*, 2014). No obstante, OWL 2 añade alguna característica nueva como tipos de datos y literales de tipos de datos. Además, tiene en cuenta aspectos no lógicos que no aparecen en las lógicas descriptivas como los nombres de ontologías y la importación de axiomas desde otras ontologías, el meta modelado o *"punning"*, axiomas no lógicos para la declaración de identificadores y el soporte de anotaciones.

Como ya se ha dicho anteriormente las ontologías OWL y las lógicas descriptivas siguen la asunción de mundo abierto, por lo que han sido diseñadas para poder tratar con conocimiento incompleto. Una consecuencia lógica de una ontología es un axioma que cumple con todas las interpretaciones que satisfacen la ontología. Las lógicas descriptivas son monotónicas, es decir la adición de nuevos axiomas, siempre provocará nuevas consecuencias. Si no existiese una interpretación válida para una ontología, entonces se diría que esta es inconsistente o insatisfacible.

2.3 Métodos para gestionar la incertidumbre

Al comienzo de esta tesis, se mencionaba la importancia de definir el concepto incertidumbre. El *"W3C Uncertainty Reasoning for the World Wide Web Incubator Group"* considera a la incertidumbre como los diferentes aspectos del conocimiento imperfecto, tales como la vaguedad o la incompletitud; el razonamiento con incertidumbre como la colección de métodos usados para modelar y razonar con conocimiento cuyos valores de verdad booleanos son desconocidos, incognoscibles o inaplicables(Laskey *et al.*, 2008). Sin embargo, otros autores (Baader *et al.*, 2003; Lukasiewicz & Straccia, 2008) consideran que existen diferencias suficientes para distinguir entre el conocimiento con vaguedad e incertidumbre. Para estos investigadores, la incertidumbre en el conocimiento está formado por declaraciones que son verdaderas o falsas, de las cuales no se está seguro debido a la falta de conocimiento. En cambio, el conocimiento con vaguedad está compuesto por declaraciones que son ciertas hasta un determinado grado debido a nociones vagas.

En esta tesis, la palabra incertidumbre se refiere a aquélla producida por el conocimiento incompleto. Por tanto, los métodos más apropiados para tratar la incertidumbre son mediante probabilidad o posibilidad. Bajo estas teorías, las declaraciones serán ciertas o falsas con una probabilidad o posibilidad asociada. La principal diferencia entre las teorías de probabilidad y posibilidad es que la probabilidad de un evento es la suma de las probabilidades de todos los mundos en los que se satisface un evento. Por su parte, la posibilidad de un evento es simplemente la posibilidad más optimista del mundo que satisfaga el evento. Este hecho unido a un mayor interés por el desarrollo de soluciones que hagan uso de la probabilidad, han contribuido a que me decante por el uso de la teoría de la probabilidad para el desarrollo del razonador.

2.3.1 La teoría de la probabilidad

Cuando se habla de probabilidad vulgarmente suele referirse al grado de confidencia de que un suceso de naturaleza incierta pueda ocurrir. Por ejemplo, la probabilidad de que una persona se levante antes de las seis de la mañana es baja. La teoría de la probabilidad ofrece los fundamentos formales para estudiar este tipo de estimaciones y las reglas que obedecen (Koller & Friedman, 2009).

Un suceso es un subconjunto de los resultados posibles que se pueden obtener cuando se produce una acción de naturaleza incierta. Por ejemplo, los resultados posibles para un partido de la quiniela son la victoria local, el empate y la victoria visitante. A partir de estos resultados, se podría preguntar por la probabilidad del suceso victoria local o por la probabilidad del sucesor empate o victoria visitante, el cual engloba dos resultados posibles. En otras palabras, los resultados se pueden agrupar en sucesos. La probabilidad asignada a un suceso es igual o mayor que cero, pero nunca superior a uno. La suma de las probabilidades asignadas a cada resultado es igual a uno. Por su parte la distribución de probabilidad empareja los sucesos con la probabilidad de que estos ocurran.

Una variable aleatoria representa una variable cuyo valor viene determinado por un experimento aleatorio. El conjunto de valores que puede tomar dicha variable se le denomina dominio A la distribución de probabilidad sobre los sucesos

que se pueden describir utilizando una variable aleatoria se le denomina distribución marginal. Por ejemplo. la variable aleatoria *estadoBombilla* puede tomar los valores *encendido* y *apagado* y tendría la siguiente distribución: P(estadoBombilla = encendido) = 0.3, P(estadoBombilla = apagado) = 0.7.

Sin embargo, hay ocasiones en los que la probabilidad a representar es la probabilidad de que sucedan dos sucesos de dos variables aleatorias a la vez. Este tipo de distribución se le denomina probabilidad conjunta. Un ejemplo, es la probabilidad de estar en la cama y que la luz esté apagada. La variablle *estaEnLaCama* tiene los valores *sí* y *no*; por lo que la probabilidad conjunta será P(estadoBombilla = encendido, estaEnLaCama = Sí) = 0.2, P(estadoBombilla = encendido, estaEnLaCama = No) = 0.4, P(estadoBombilla = apagado, estaEnLaCama = Sí) = 0.3 y P(estadoBombilla = apagado, estaEnLaCama = No) = 0.1.

Finalmente, la probabilidad condicional representa la probabilidad de que un suceso de una variable aleatoria ocurra conociendo que el suceso de otra variable aleatoria ha ocurrido. Por ejemplo, la probabilidad de que la luz este encendida sabiendo que el usuario está en la cama P (estadoBombilla | estaEnLaCama = Sí). Dos sucesos son independientes, si la probabilidad condicionada del suceso A respecto el suceso B es igual a la probabilidad del suceso A. En el caso de las variables, dos variables aleatorias A y B son independientes dada una tercera variable aleatoria Z si y solo si para cualquiera de los resultados de Z, la distribución de probabilidad de X es la misma para todos los valores de Y y viceversa.

El teorema de Bayes, ver fórmula 2.27, fue enunciado por Thomas Bayes y permite la actualización de las creencias en función de las evidencias. En otras palabras, permite calcular $P(A|B)$ en función de $P(B|A)$

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.27}$$

Por su parte las redes bayesianas son un grafo acíclico dirigido en el que los nodos representan variables aleatorias y los arcos entre los nodos dependencias entre estos cuantificadas por la probabilidad condicional (Pearl, 1986). Por ejemplo, si hay dos nodos *A* y *B* conectados por un arco dirigido de *A* a *B*, entonces se dice que *B* depende de *A*. Uno de las principales utilidades de las redes bayesianas es la representación de conocimiento de un dominio.

2.4 Conclusiones

En este capítulo se han descrito las bases sobre las que se asienta este trabajo. Por una parte, se ha dado una introducción a las ontologías OWL y a su base teórica, las lógicas descriptivas. Finalmente, se han expuestos los problemas que pueden presentar las ontologías a la hora de gestionar incertidumbre, y una breve introducción al enfoque escogido.

Con esta base teórica se pretende ofrecer una idea general sobre las ontologías, así como los problemas que presentan para la gestión de la incertidumbre. En el siguiente capítulo se analizan diferentes propuestas que existen para gestionar la incertidumbre que hacen uso de métodos probabilísticos.

Una cosa es continuar la historia y otra repetirla

Jacinto Benavente

CAPÍTULO

3

Estado del arte

Hecha la maleta, el explorador analiza el mapa y estudia las diferentes rutas existentes para llegar a su destino. Es consciente de que no existe un itinerario seguro, el peligro acecha en cada esquina y en cualquier momento puede surgir un problema imprevisto que dé al traste su aventura. No obstante, sabe que por desconocido que sea el terreno siempre hay un camino más seguro que otro. La decisión de recorrer uno u otro puede ser la diferencia entre alcanzar su objetivo o perecer en el intento.

Siguiendo la metáfora del explorador, en este capítulo se exponen los diferentes caminos que existen para combinar las ontologías con la probabilidad. Estas aproximaciones se suelen dividir en dos grupos en aquellas basadas en extensiones probabilísticas de lógicas descriptivas y en extensiones probabilísticas de OWL (Lukasiewicz & Straccia, 2008; Costa, 2005).

3.1 Extensiones probabilísticas de lógicas descriptivas

Las extensiones probabilísticas de las lógicas descriptivas se suelen clasificar en función de la lógica descriptiva que extienden, el tipo de conocimiento probabilísticos que soportan y el razonamiento subyacente que soportan (d'Amato *et al.*, 2008). Existen diferentes propuestas como la de Heinsohn (Heinsohn, 1994) que

extendió la lógica descriptiva $\mathcal{ALC}$ con probabilidad, la cual es una de las primeras propuestas que extienden las lógicas descriptivas con probabilidad. Otro de los primeros trabajos fue el Jaeger (Jaeger *et al.*, 1994). Aparte de propuestas antiguas como la anteriormente mencionadas, existen otras más actuales como la semántica DISPONTE(Riguzzi *et al.*, 2014) para lógicas descriptivas probabilísticas y su razonador BUNDLE (Riguzzi *et al.*, 2013). También existen extensiones probabilística de $\mathcal{EL}$ como $\mathcal{BEL}$ que hace uso de redes bayesianas (Ceylan & Penaloza, 2014) o la propuesta en (Lutz & Schröder, 2010).

A pesar de la existencia de múltiples lógicas descriptivas probabilísticas, las más importantes y relevantes en la literatura para este trabajo son las que se describen en las siguientes subsecciones.

3.1.1 P-CLASSIC

P-CLASSIC (Koller *et al.*, 1997) es la versión probabilística de la lógica descriptiva CLASSIC (Borgida & Patel-Schneider, 1994). En ella hacen uso de las redes bayesianas, cuyas variables aleatorias son los conceptos primitivos, el número de *fillers* y las propiedades de los *fillers*. Se denomina *filler* a los individuos con los que está relacionado un individuo a través de un rol. Su principal objetivo es expresar el grado de solapamiento entre conceptos. El componente probabilístico de P-CLASSIC son las p-class, que describen información probabilística relacionada con cierta clase de individuos. Cada p-class se representa mediante el uso de una red bayesina.

Una de las ventajas de esta lógica es que el algoritmo de inferencia se ejecuta en un tiempo lineal en función de las descripciones de conceptos y cuadrático en el número de las p-classes en la base de conocimiento. Si todas las redes bayesianas asociadas a las p-classes son *polytree* (red bayesiana cuyo grafo subyacente no dirigido es acíclico), entonces el tiempo también será polinomial en el tamaño de la base de conocimiento.

3.1.2 Pronto

Pronto (Klinov & Parsia, 2010; Klinov, 2011) es un razonador para la extensión probabilística de la lógica descriptiva $\mathcal{SROIQ}$, denominada P-$\mathcal{SROIQ}$. Pronto

hace uso de una anotación propia (pronto#certainty) para definir la probabilidad. El valor de la anotación es una cadena de texto que representa un intervalo. Este intervalo viene definido por dos valores numéricos comprendidos entre 0 y 1 que se encuentran separados por un punto y coma ";", como se puede ver en el listado 1. Mediante éstas se puede representar relaciones condicionadas probabilísticas entre conceptos complejos arbitrarios.

```
<SubClassOf>
<Annotation>
<AnnotationPropertyIRI="pronto#certainty"/>
<Literal>0.7;0.95</Literal>
</Annotation>
<ClassIRI="Bird"/>
<ObjectIntersectionOf>
<ClassIRI="FlyingObject"/>
<ClassIRI="WingedObject"/>
</ObjectIntersectionOf>
</SubClassOf>
<ClassAssertion>
<Annotation>
<AnnotationPropertyIRI="pronto#certainty"/>
<Literal>0.7;0.95</Literal>
</Annotation>
<ClassIRI="Bird"/>
<NamedIndividualIRI="Tweety"/>
</ClassAssertion>
```

Listado 1: Ejemplo de definición de incertidumbre en Pronto extraído de *Practical Reasoning in Probabilistic Description Logic* (Klinov, 2011). En él se muestran ejemplos de restricciones al PTBox y al PABox.

Las bases de conocimiento en P-$\mathcal{SROIQ}$ están formadas por la base de conocimiento de una ontología OWL normal, el PTBox y el PTAbox. El PTBox expresa que si un individuo aleatorio es una instancia de un determinado concepto, entonces

la probabilidad de ser una instancia de otra clase está en un determinado intervalo. Por su parte el PABox define que si un individuo específico es una instancia de una clase, la probabilidad de ser una instancia de otro concepto está en un intervalo determinado.

Finalmente, cabe destacar que ofrece una interfaz para usuarios mediante línea de comandos, así como una API para poder ser utilizado en aplicaciones de terceros.

3.1.3 Bayesian *DL–Lite*$_{\mathcal{R}}$

Bayesian *DL-Lite*$_{\mathcal{R}}$ (d'Amato *et al.*, 2008), o simplemente $BDL\text{-}Lite_{\mathcal{R}}$, es una lógica descriptiva probabilística que combina las redes bayesianas con la lógica descriptiva $DL\text{-}Lite_{\mathcal{R}}$, miembro de la familia $DL\text{-}Lite$ (Calvanese *et al.*, 2005). Esta lógica está enfocada en aplicaciones que hagan uso intensivo de datos. Una de sus grandes ventajas es que tanto la comprobación de satisfacibilidad y el proceso de consultas puede realizarse en LogSpace respecto al tamaño de los datos. Los axiomas probabilísticos se expresan de la siguiente forma:

$$\phi : X = x \tag{3.1}$$

Donde ϕ es un axioma que representa la pertenencia a un concepto, la pertenencia a un rol, una inclusión de conceptos, una inclusión de roles o la funcionalidad de un rol; y $X = x$, una anotación probabilística, donde X representa una variable aleatoria y x, un valor de su dominio. Por tanto, se puede definir al TBox probabilístico en $BDL\text{-}Lite_{\mathcal{R}}$ como el conjunto finito de axiomas de inclusión de conceptos probabilísticos y de inclusión de roles probabilísticos en $BDL\text{-}Lite_{\mathcal{R}}$ y al ABox probabilístico en $BDL\text{-}Lite_{\mathcal{R}}$ como el conjunto de axiomas probabilísticos de pertenencia a conceptos y de pertenencia a roles en $BDL\text{-}Lite_{\mathcal{R}}$. Finalmente, la base probabilística de conocimiento está formada por el TBox probabilístico, el ABox probabilístico y la red bayesiana. Las probabilidades de la red bayesiana se expresan en forma de tablas de probabilidad condicionada.

3.2 Extensiones probabilísticas de OWL

Aparte de las extensiones de las lógicas descriptivas, existen extensiones probabilísticas de las ontologías OWL. Muchas de ellas combinan las ontologías OWL con formalismos probabilísticos basados en redes bayesianas. Ejemplos de estas aproximaciones son OMEN, que utiliza redes bayesianas para el emparejamiento de ontologías (Mitra *et al.*, 2005); Knowledge Elicitation Environment for Probabilistic Event and EntityRelation (KEEPER), que es un sistema para la gestión de modelos relacionales con probabilidad (Pool & Aikin, 1994); y la propuesta de Fukushige que representa la probabilidad en RDF y hace uso de redes bayesianas para calcular las probabilidades (Fukushige, 2004). No obstante los trabajos más importantes y notorios son los descritos en las siguientes subsecciones.

3.2.1 BayesOWL

BayesOWL (Ding, 2005) es un framework que añade a OWL la capacidad de representar y razonar con incertidumbre mediante el uso de redes bayesianas. Se permite expresar dos tipos de probabilidades: probabilidades a priori o marginal y probabilidades condicionadas.

BayesOWL cuenta con un conjunto de normas que permiten la conversión de una ontología OWL en un grafo acíclico dirigido de una red bayesiana. Esta conversión consiste en la transformación de las clases de la ontología en nodos, así como la creación de nodos especiales conocidos como L-Nodes que modelan relaciones entre conceptos en los que interviene un operador lógico de OWL. Dicha conversión está regida por las siguientes reglas:

- Cada nodo es una variable binaria con los estados: verdadero y falso. Por ejemplo, la clase *Casa* tendría los estados $casa$ y $\overline{casa}$, siendo $P(casa)$ la probabilidad de pertenecer al concepto *Casa* y $P(\overline{casa})$ su inverso.

- El constructor *rdfs:subClassOf* es modelado como un arco dirigido de las clases padres a la hija.

- Si se define una clase como la intersección de varias clases mediante el constructor *owl:intersectionOf*, entonces se establecen arcos que parten de las clases que interseccionan hacia la clase que da lugar su intersección. Además, se crea un nuevo nodo del tipo L-Node al cual llegan arcos desde todas las clases implicadas en la intersección.

- Si se define una clase como la unión de varias clases, entonces los arcos van desde la clase definida hacia las clases cuya unión la forman. Además, se crea un nuevo nodo del tipo L-Node al cual llegan arcos desde todas las clases implicadas en la unión de clases.

- Cuando dos clases están relacionados por los constructores *owl:complementOf, owl:equivalentClass o owl:disjointWith*, entonces se crean arcos desde estas clases hasta un nuevo L-Node.

Las tablas de probabilidad condicionadas (CPTs) se construyen de manera diferente dependiendo del tipo de nodo. Para los nodos L-Node el CPT se construye en función de la relación lógica que se represente. En cambio, para los nodos que representan conceptos se calculan sus CPTs mediante el SD-IPFP que es una extensión del algoritmo *iterative proportional fitting procedure* (IPFP), que consiste en construir los CPTs basándose en el cumplimiento de unas restricciones.

La representación de las probabilidades en la ontología se realiza mediante el uso de las clases *PriorProb* para las propiedades a priori y *CondProb* para las probabilidades condicionadas. Los individiuos de estas clases se relacionan con el nombre de la variable a la que representan con la propiedad *hasVariable* y con su probabilidad con *hasProbValue*. En el caso de los individuos de la clase *CondProb* tienen una propiedad adicional, *hasCondition*, para indicar sus dependencias. Las variables se definen mediante la clase *Variable* y se relacionan con la clase a la que hacen referencia mediante el uso de la propiedad *hasClass* y el valor de su estado con la propiedad *hasState*. El listado de código 2 es un ejemplo de cómo se representan las probabilidades en BayesOWL.

BayesOWL es capaz de realizar tareas de razonamiento de ontologías, como razonamiento probabilístico sobre la red bayesiana creada. Ejemplos de ello son las

tareas de satisfactibilidiad de conceptos, superposición de conceptos o subsunción de conceptos.

De acuerdo al manual de referencia (Yun Peng, 2013), la versión actual de BayesOWL (1.0) presenta las siguientes limitaciones:

- Todas las variables han de ser binarias.
- Solo puede manejar probabilidades que tengan como componentes una variable aleatoria a priori.
- Las probabilidades descritas deben de ser completas.
- En caso de inconsistencia, la red bayesiana dada como respuesta puede no cumplir con las restricciones.
- El archivo que se genera describiendo la red para Netica 2.0, librería usada para el cálculo de probabilidad de las redes bayesianas, no contiene información de cómo mostrar los nodos alineados correctamente en el programa de visualización de Netica.

No obstante la última limitación no se considerara un limitación relevante para este trabajo, sino más bien una limitación de Netica.

3.2.2 **OntoBayes**

OntoBayes (Yang & Calmet, 2005) apuesta también por el uso de redes bayesianas junto con las ontologías. Para la integración de las redes bayesianas en las ontologías se definen tres nuevas clases:

- *PriorProb* representa una probabilidad a priori.
- *CondProb* define probabilidades condicionales.
- *FullProbDist* describe probabilidades completas disjuntas.

Tanto *PriorProb* y *CondProb* utilizan la propiedad de datos *ProbValue* para expresar un valor probabilístico entre 0 y 1. Por contra, *FullProbDist* tiene las propiedades objeto disjuntas *hasPrior* y *hasCond*. La primera de ellas relaciona *FullProbDist* con *PriorProb* para indicar que las instancias de *PriorProb* forman parte de *FullProbDist*. Por su parte, *hasCond* describe la relación entre *FullProbDist* y *CondProb*, siendo su significado similar al de *hasPrior*.

En OntoBayes las variables aleatorias solo pueden ser propiedades de objeto o propiedades de datos y las relaciones entre ellas se definen mediante el uso de la propiedad *rdfs:dependsOn*.

3.2.3 PR-OWL

PR-OWL (Costa, 2005) es una extensión de OWL que permite representar complejos modelos de Bayes. Se fundamenta en la lógica bayesiana de primer orden denominada Multi-Entity Bayesian Networks (MEBN) (Laskey, 2008).

MEBN otorga a PR-OWL la base matemática formal con la que extender OWL, a la vez que le permite beneficiarse de las ventajas que ofrece la inferencia bayesiana. En ella se representa el conocimiento probabilístico como una colección de fragmentos MEBN denominados MFrags que se organizan en teorías MEBN (MTheories). Por su parte, un MFrag es la distribución de probabilidad de un grupo de variables aleatorias *residentes* dados los valores de las variables aleatorias de contexto y de entrada. Un conjunto de MFrags representa una distribución de probabilidad conjunta sobre un número ilimitado de instancias de sus variables aleatorias. Dicho conjunto se denomina MTheroy si cumple con las restricciones de consistencia asegurando que existe únicamente una distribución de probabilidad conjunta sobre sus variables aleatorias.

La ontología base (*upper ontology*) de PR-OWL permite representar una MTheory con el objetivo de construir ontologías probabilísticas. Además, se permite la coexistencia de conceptos probabilísticos con no probabilísticos en una misma ontología probabilística. No obstante, solo los conceptos que formen parte de la MTheory se verán afectados por las ventajas de una ontología probabilística.

La condición mínima e indispensable para que una ontología se considere una ontología probabilística de PR-OWL es tener al menos un individuo de la clase

MTheory. Los individuos del tipo *MTheory* se relacionan con los MFrags que la constituyen por medio de la ObjectProperty *hasMFrag*, como muestra la figura 3.1. Cada MFrag está formado por un grupo de nodos representados por la clase

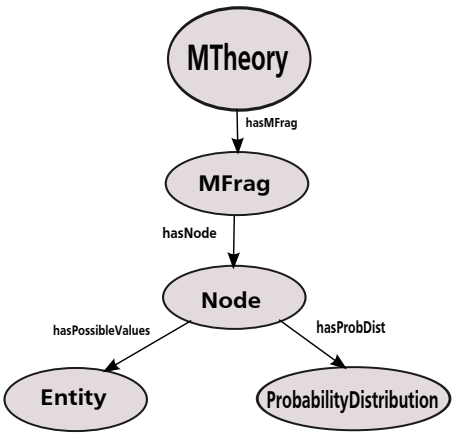


Figura 3.1: Representación de los conceptos básicos de PR-OWL

Node que pueden ser de tres tipos según su subclase:

- Residentes representados por la clase *Resident* son variables aleatorias cuya distribución de probabilidad está definida en la MFrag.
- De entrada definidos con la clase *Input* son variables aleatorias cuya distribución de probabilidad está definida en un MFrag distinto.
- De contexto representados con la clase *Context* indican cuáles son las condiciones que deben de satisfacerse para poder aplicar las influencias y distribuciones locales.

Para definir los diferentes estados que puede tener un nodo se describen mediante el uso de la ObjectProperty *hasPossibleValues* y tomando de rango individuos de la clase *Entity*. Los estados a su vez se enlazan con sus distribuciones de probabilidad, representadas por la clase *ProbDist*, mediante la propiedad de objeto *hasProbDist*.

Las distribuciones de probabilidad se pueden representar en diferentes formatos (ecuaciones de Netica, fórmulas de Quiddity, sintaxis de MEBN, etc.) mediante el uso de *DeclarativeDist* que es subclase de *ProbDist*, ya que se representan como

una cadena de texto. No obstante, esto presenta el inconveniente de que se deben utilizar conversores para poder obtener los valores. Otra forma de representar las distribuciones de probabilidad es como tablas de probabilidad condicionada mediante el uso de *PR-OWLTable*, subclase también de *ProbDist*.

El desarrollo de PR-OWL (Laskey, 2010) se centró en dos puntos: (i) el desarrollo de un framework lógico y semántico que fundamente el lenguaje y (ii) en la creación de un razonador MEBN para PR-OWL basado en UNBBayes[1].

Posteriormente, se presentó PR-OWL 2.0 (Carvalho *et al.*, 2010; Carvalho, 2011) que supera las limitaciones de PR-OWL: la carencia de un método para relacionar las definiciones de conceptos probabilísticos a sus homólogos en OWL. Para ello hace uso de la propiedad *definesUncertaintyOf* que relaciona una individuo de la clase *RandomVariable* con una propiedad. No obstante, esta solución presenta el problema de que para que funcione se necesita hacer uso de OWL Full, ya que esto no es posible en OWL 2 DL. Con el objetivo de mantener la expresividad dentro de OWL 2 DL en vez de relacionar las variables aleatorias directamente con una propiedad, establecen que el rango de esa propiedad sea una URI. De esta forma su valor es la URI de la propiedad que se relaciona con la variable, y el razonador PR-OWL es capaz de entender la asociación entre la propiedad y la variable aleatoria.

3.3 Conclusiones

En este capítulo se resumen los principales métodos para combinar las ontologías con probabilidad. Para obtener más información sobre diferentes métodos de representación de incertidumbre en ontologías, tanto probabilísticos como no probabilísticos, se pueden consultar (Lukasiewicz & Straccia, 2008) y (Predoiu & Stuckenschmidt, 2008), entre otros. En el grupo de las extensiones probabilísticas se situan aquellas propuestas que extienden las lógicas descriptivas, destacando P-CLASSIC, P-$\mathcal{SROIQ}$ y $DL\text{–}Lite_{\mathcal{R}}$. Este enfoque fue desechado debido a que se consideró más efectivo la extensión de OWL debido a su mayor popularidad, cantidad de herramientas existentes y respaldo por la W3C. Por otra parte, de las

[1]Es un proyecto de código abierto disponible en `http://sourceforge.net/projects/unbbayes`

	BayesOWL	OntoBayes	PR-OWL
Incertidumbre de tipo	✓		✓
Incertidumbre de valor de propiedad		✓	✓
Estructura del modelo probabilístico	Creación automática mediante reglas	Definida para la ocasión	Definida para la ocasión

Cuadro 3.1: Lista de diferencias relevantes para esta tesis entre BayesOWL, PR-OWL y OntoBayes

tres principales analizadas, sólo P-$\mathcal{SROIQ}$ ofrece la expresividad que se desea, respecto a la lógica descriptiva base que se extiende. No obstante, P-$\mathcal{SROIQ}$ no está orientado para el cálculo de la probabilidad de que un individuo esté relacionado mediante una propiedad con otro individuo o literal, por lo cual tampoco sirve para el propósito de esta tesis que es el cálculo de aserciones probabilísticas tanto de propiedades como de pertenencia a clases.

En el segundo grupo se encuentran las extensiones a OWL, entre los cuales destacaban BayesOWL, PR-OWL y OntoBayes. Las principales diferencias entre estas propuestas se recogen en la tabla 3.1. De las tres propuestas solo PR-OWL tiene en cuenta tanto propiedades como conceptos, lo que la convierte en la opción más interesante de cara al objetivo de la tesis. Sin embargo, no cumple con los objetivos de mantener aislado la definición del modelo probabilístico del tradicional, por lo que aquellos razonadores o sistemas no compatibles con PR-OWL cargarían la información probabilística y tradicional. Una forma de evitar esto sería la creación de una ontología para el conocimiento tradicional y una segunda que importase a la primera y qu modelase los MFrags. Desafortunadamente esta solución incumple el objetivo de usar una IRI única a la hora de referenciar la ontología probabilística y tradicional.

En el siguiente capítulo se describe la propuesta planteada para representar y gestionar la incertidumbre en entornos inteligentes. Así mismo, se presentan las soluciones que ofrecen para mantener aislado el conocimiento probabilístico de la información presentada en las ontologías.

```
<Variable rdf:ID="c">
<hasClass>C</hasClass>
<hasState>True</hasState>
</Variable>
<Variable rdf:ID="p1">
<hasClass>P1</hasClass>
<hasState>True</hasState>
</Variable>
<Variable rdf:ID="p2">
<hasClass>P2</hasClass>
<hasState>True</hasState>
</Variable>
<Variable rdf:ID="p3">
<hasClass>P3</hasClass>
<hasState>True</hasState>
</Variable>
<CondProb rdf:ID="P(c|p1, p2)">
<hasCondition>p1</hasCondition>
<hasCondition>p2</hasCondition>

<hasVariable>c</hasVariable>
<hasProbValue>0.8</hasProbValue>
</CondProb>
 <PriorProb rdf:ID="P(P3)">
<hasVariable>p3</hasVariable>
<hasProbValue>0.8</hasProbValue>
</PriorProb>
```

Listado 2: Ejemplo de definición de probabilidades en BayesOWL extraído de la tesis *BayesOWL: A Probabilistic Framework for Uncertainty in Semantic Web* (Ding, 2005). En él se representa que un individuo sea miembro de la clase C cuando este pertenece a la intersección de las clases p1 y p2 y que un individuo sea miembro de la clase P3.

En lo tocante a ciencia, la autoridad de un millar no es superior al humilde razonamiento de un hombre.

Galileo Galilei

CAPÍTULO

4

Representación y razonamiento de la incertidumbre

El viaje es largo y el camino no es siempre como se espera. A veces, el explorador tiene que desandar sus pasos para afrontar los retos desde una nueva perspectiva. Si no se puede cruzar la montaña, habrá que bordearla, pero siempre con el objetivo claro en la mente de llegar al otro lado.

En este capítulo se presenta la contribución base de esta tesis: la representación de la incertidumbre en las ontologías. Por ello, se define cómo modelar el conocimiento con incertidumbre y cumplir los requisitos de IRI única por ontologías y la posibilidad de extender el funcionamiento del razonador. La importancia de los conceptos introducidos en este capítulo es crucial para entender las capacidades del razonador presentado, así como sus limitaciones. Se han desarrollado diferentes iteraciones que se han ido revisando y refinando hasta alcanzar la versión final, como se muestra en la figura 4.1. En este capítulo se describe el funcionamiento del razonador y el método de modelado propuesto, denominado Turambar, así como las versiones anteriores; ya que sin estas no se podrían comprender algunos cambios y decisiones tomados de cara a la versión final.

Para el desarrollo de Turambar se han utilizado las siguientes librerías, sin las cuales no se podría haber implementado:

- HermiT [1].
- Pellet [2].
- OWL API [3].
- UnBBayes [4].
- Derivo SPARQL-DL [5].

4.1 Primera iteración: añadiendo incertidumbre a las ontologías

La primera iteración fue una prueba de concepto de cómo representar modelos probabilísticos en ontologías interfiriendo lo menos posible con el conocimiento descrito por la ontología. Aunque técnicamente no era muy avanzado, estableció las bases del uso de anotaciones para representar la información probabilística.

El modelo probabilístico era representado mediante el uso de redes bayesianas, que se convertirían en el principal y único método para la representación de probabilidad para futuras versiones. Se permitía asociar tanto clases como propiedades con variables aleatorias mediante el uso de las anotaciones: *talismanPropertyProbability* y *talismanClassProbability*.

La anotación *talismanPropertyProbability* relacionaba las propiedades de objetos y datos con las variables aleatorias de una red bayesiana, siendo los valores que esta podía tomar parte del dominio de la variable. Se asumía que las propiedades iban a ser funcionales, es decir un individuo estaría conectado mediante una propiedad a otro individuo o literal como máximo una vez. Por otra parte, la anotación *talismanClassProbability* servía para anotar clases. La primera característica que

[1] `http://hermit-reasoner.com/`
[2] `http://clarkparsia.com/pellet/`
[3] `http://owlapi.sourceforge.net/`
[4] `http://unbbayes.sourceforge.net/`
[5] `http://www.derivo.de/en/resources/sparql-dl-api.html`

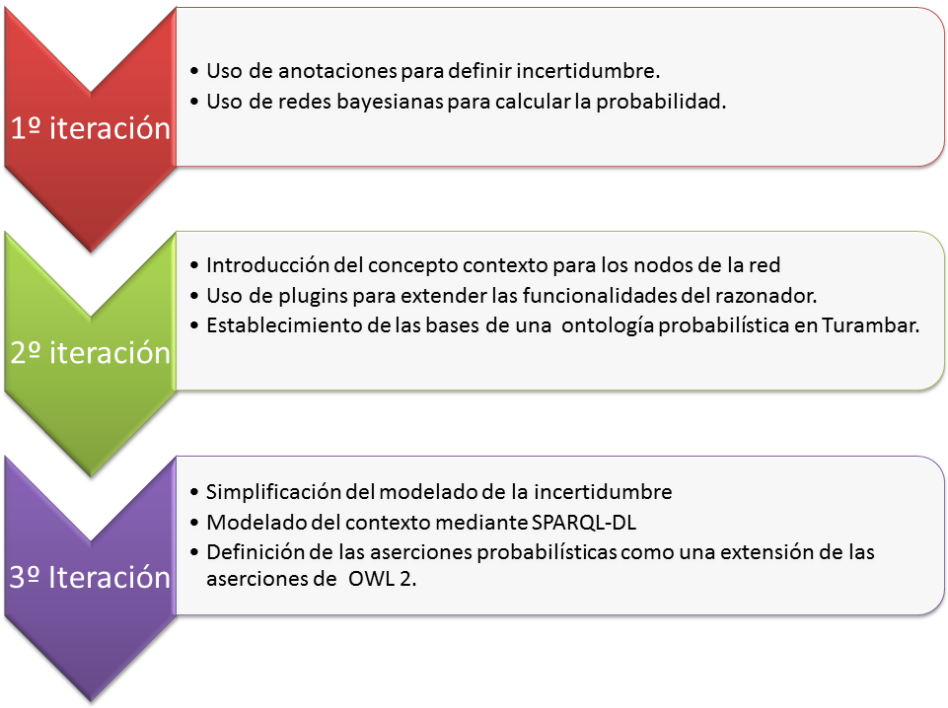
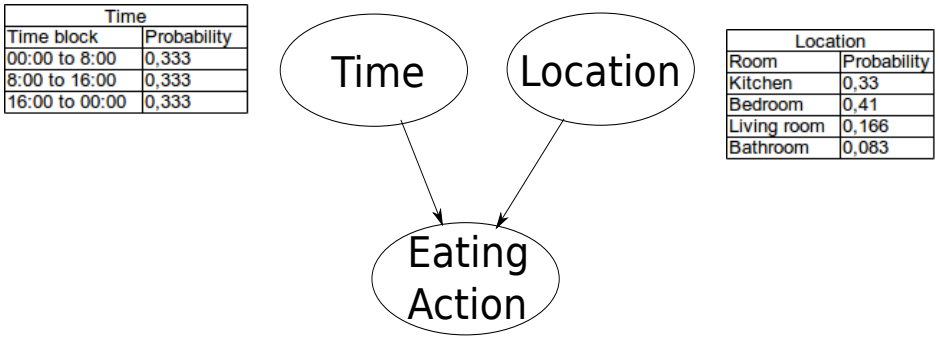


Figura 4.1: Introducción de las principales caracterísitcas del sistema de representación de la incertidumbre a lo largo de las iteraciones.

resulta llamativa de esta anotación es que la clase que se anota no es la clase de la que se quiere inferir la probabilidad, sino el padre de aquellas de las que se quiere inferir la probabilidad.

El valor de las anotaciones es bastante similar independientemente de si se ha anotado una clase o propiedad. Este contenido está dividido en tres secciones:

- El identificativo es el nombre de la variable aleatoria. Tiene que aparecer en la primera línea de la anotación y ser único.
- La definición del grafo establece los nodos que condicionan al nodo actual. Esta parte es opcional y debe de aparecer a continuación de la identificación. Las dependencias se describen especificando el nombre de la variable que depende seguido de "->*this*".
- La distribución de probabilidad viene definida en forma de tabla de probabilidad condicionada. Cada línea es una fila de la tabla y empieza con el valor



Time	
Time block	Probability
00:00 to 8:00	0,333
8:00 to 16:00	0,333
16:00 to 00:00	0,333

Location	
Room	Probability
Kitchen	0,33
Bedroom	0,41
Living room	0,166
Bathroom	0,083

Eating Action												
Time	00:00 to 8:00				8:00 to 16:00				16:00 to 00:00			
Location	Kitchen	Bedroom	Living room	Bathroom	Kitchen	Bedroom	Living room	Bathroom	Kitchen	Bedroom	Living room	Bathroom
Meal												
Breakfast	0,125	0,1	0	0	0,9	0,9	0,6	0	0	0	0	0
Lunch	0	0	0	0	0	0	0	0	0,27	0,2	0,1	0
Dinner	0,125	0,1	0	0	0	0	0	0	0,27	0,2	0,1	0
Snacking	0,75	0,8	1	1	0,1	0,1	0,4	1	0,45	0,6	0,8	1

Figura 4.2: Ejemplo gráfico de la primera versión de Turambar. En él se representan la probabilidad de que la acción que realice el usuario sea comer, cenar, desayunar o tomar un aperitivo, en función de la habitación en la que se encuentra y la hora del día.

que tomaría la propiedad o la clase a la que pertenecería el individuo seguido de dos puntos y la fila de la tabla de propiedad, cuyas columnas están separadas por comas. Tras la última probabilidad debe de aparecer un punto y coma, en vez de la coma. El orden de las probabilidades depende del orden en el que son especificadas en la sección de definición de grafo.

El código 3 y la figura 4.2 representan un ejemplo en el que se define la probabilidad de que un usuario este desayunando, comiendo, cenando o tomando un aperitivo en función de dónde esté y la hora del día.

```
Declaration(Class(p0:Action))
Declaration(Class(p0:Bathroom))
SubClassOf(p0:Bathroomp0:Location)
Declaration(Class(p0:Bedroom))
SubClassOf(p0:Bedroomp0:Location)
Declaration(Class(p0:BreakfastAction))
SubClassOf(p0:BreakfastActionp0:EatingAction)
Declaration(Class(p0:DinnerAction))
```

```
SubClassOf(p0:DinnerActionp0:EatingAction)
Declaration(Class(p0:EatingAction))
AnnotationAssertion(p0:talismanClassProbability
p0:EatingAction
"ID:EatingAction
InitGraph
Time->this;
Location->this;
EndGraph
p0.BreakfastAction:
0.125,0.1,0,0,0.9,0.9,0.6,0,0,0,0,0;
p0.LunchAction:
0,0,0,0,0,0,0,0,0.27,0.2,0.1,0;
p0.DinnerAction:
0.125,0.1,0,0,0,0,0,0,0.27,0.2,0.1,0;
p0.SnackingAction:
0.75,0.8,1,1,0.1,0.1,0.4,1,0.45,0.6,0.8,1;")
SubClassOf(p0:EatingActionp0:Action)
Declaration(Class(p0:Kitchen))
SubClassOf(p0:Kitchenp0:Location)
Declaration(Class(p0:LaunchAction))
SubClassOf(p0:LaunchActionp0:EatingAction)
Declaration(Class(p0:LivingRoom))
SubClassOf(p0:LivingRoomp0:Location)
Declaration(Class(p0:Location))
Declaration(Class(p0:SnackingAction))
SubClassOf(p0:SnackingActionp0:EatingAction)
Declaration(ObjectProperty(p0:atLocation))
AnnotationAssertion(p0:talismanPropertyProbability
p0:atLocation
"ID:Location
p0.Kitchen:
0.33;
```

```
p0.Bedroom:0.41;
p0.LivingRoom:0.166;
p0.Bathroom:0.083;")
FunctionalObjectProperty(p0:atLocation)
ObjectPropertyDomain(p0:atLocationp0:Action)
ObjectPropertyRange(p0:atLocationp0:Location)
Declaration(DataProperty(p0:time))
AnnotationAssertion(p0:talismanPropertyProbability
p0:time"ID:Time
>=0&&<=28800:0.333;
>=28800&&<=57600:0.333;
>=57600&&<=86400:0.333;")
FunctionalDataProperty(p0:time)
DataPropertyDomain(p0:timep0:Action)
DataPropertyRange(p0:timexsd:double)
Declaration(AnnotationProperty(
p0:talismanClassProbability))
Declaration(AnnotationProperty(
p0:talismanPropertyProbability))
```

Listado 3: Definición de una ontología probabilística en la primera versión de Turambar. En ella se representan la probabilidad de que la acción que realice el usuario sea comer, cenar, desayunar o tomar un aperitivo, en función de la habitación en la que se encuentra y la hora del día.

Esta solución presenta los siguientes problemas:

- La dificultad para definir la red bayesiana. El uso del formato propuesto hace difícil la modificación de las probabilidades, así como cambios en la estructura de la red.
- Todos los nodos de la red bayesiana tienen que estar relacionados con un mismo individuo. Es decir, el individuo (sujeto) que va a estar conectado mediante una propiedad a un literal u otro individuo o va a ser miembro de una clase va a ser siempre el mismo. Por ejemplo, en la red del ejemplo

anterior el sujeto tenía que ser forzosamente una actividad. La actividad es la que tiene que estar relacionada con el lugar en el que sucede, ya que no se podría acceder a esta información de ninguna otra forma. Lo mismo es aplicable para la hora.

- No existe la posibilidad de utilizar un contexto que límite el empleo de la red para individuos que cumplan unas ciertas condiciones.
- Todos los nodos han de devolver una aserción probabilística.
- No es posible extender la funcionalidad básica.

Estas limitaciones se reflejan en el siguiente ejemplo. Supóngase que se desea modelar la probabilidad de que John tome mal el medicamento. La ingesta de cada medicamento se recoge mediante instancias de la clase *Ingesta* que tiene de propiedades el medicamento que se toma, la hora a la que hay que realizarla y el paciente que tiene que medicarse. La red propuesta para calcular la probabilidad de que un paciente bajo vigilancia ingiera mal un medicamento se refleja en la figura 4.3. En ella se describe que la probabilidad de que alguien tome mal una pastilla viene condicionada por la hora del día, por la existencia de dislexia por parte del paciente y por la similitud del medicamento a otros existentes.

De la definición de este ejemplo se puede llegar a la conclusión de que esta iteración no ofrece la expresividad suficiente para llevar a cabo el modelado del problema. Por una parte, los nodos no están relacionados con el mismo individuo. Ejemplo de ello es que el nodo ”es similar” estaría relacionado con el medicamento y el nodo ”paciente disléxico” con el paciente. Por otro lado, no es posible definir que esa red bayesiana solo se aplica a los pacientes no autónomos. Además, tampoco se puede evaluar si existen medicamentos similares, ya que solo se soportan comparaciones de los valores de propiedad y para este caso lo que se necesita es saber si existen o no algún medicamente similar. No solo el nodo ”es similar” requiere del uso de una función no soportada para valorar el estado, sino que no se desea extraer una aserción probabilística del nodo. Finalmente es importante remarcar que aunque la red pudiese ser modelada, esta seguiría presentando los problemas de formato.

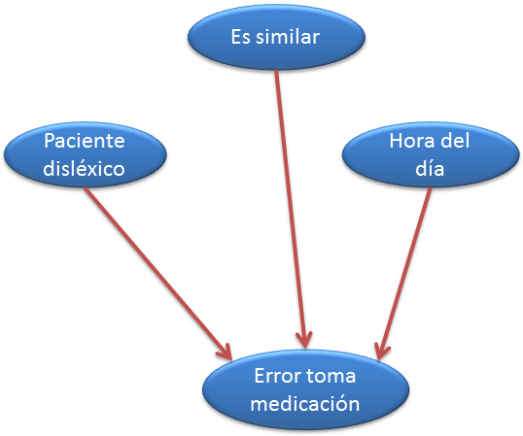


Figura 4.3: Ejemplo de red bayesiana que describe la probabilidad de que un individuo se haya tomado mal un medicamento.

Con el objetivo de solventar estos inconvenientes se desarrolló la segunda versión de Turambar.

4.2 Segunda iteración: separación del modelo probabilístico del vocabulario del dominio

La segunda versión sienta las bases para la tercera y definitiva versión de Turambar. En esta versión se relaciona por primera vez las redes bayesianas con el concepto de contexto, el cual es similar al de PR-OWL (Costa, 2005). Por otra parte, se define también por primera vez el concepto de ontología probabilística en Turambar y se establece la separación entre el conocimiento tradicional y la definición de la red bayesiana mediante el uso de dos documentos distintos.

En esta versión, la ontología probabilística está formada por una ontología OWL 2 DL ordinaria y una segunda ontología que describe las dependencias entre clases y propiedades (la red bayesiana), denominada ontología de dependencias. La ontología ordinaria se relaciona con la ontología de dependencias mediante el uso de anotaciones definidas en la ontología de anotaciones de Turambar:

- *turambarOntology* se usa únicamente una vez en la ontología ordinaria y sirve para anotar la propia ontología. El valor de la anotación debe de ser la ubicación de la ontología de dependencias.

- *turambarClass* identifica los nodos referidos a clases. Sirve para indicar la probabilidad de que un individuo se clasifique como miembro de una clase.

- *turambarProperty* identifica los nodos referidos a una propiedad. Se utiliza para indicar que un individuo puede estar relacionado con una serie de valores con una probabilidad asociada.

Tanto el valor de las anotaciones *turambarClass* y *turambarProperty* se corresponden al identificador de un individuo en la ontología de dependencias que es miembro de la clase Node. En el caso de que el identificador no sea una IRI valida, se creará una concatenando el valor de la anotación con la IRI de la ontología de dependencias.

Cada clase y propiedad anotada está asociada con un nodo de la red bayesiana descrita en la ontología de dependencias. En esta ontología se pueden diferenciar dos tipos de nodos: conocidos y perdidos. Los nodos conocidos son los nodos que pueden generar axiomas de Turambar y están referenciados desde la ontología ordinaria. Por su parte, los perdidos son aquellos que no están relacionados con ninguna propiedad, pero que son necesarias para el modelado de la red.

Los nodos conocidos agrupan a dos conjuntos de nodos que se diferencian en función de si están relacionados con propiedades no funcionales:

- Multi nodos. Son aquellos nodos asociados a propiedades no funcionales. Se les conoce como multi nodos porque cada estado de este equivaldrá a un nodo en la red bayesiana con dos estados. Estos indican si el individuo posee una propiedad declarada en la ontología con esos valores. Por ejemplo, si el nodo final A se define con los estado c: 0.5, d:0.3, e:0.1 que indican las probabilidades de que un individuo esté relacionado con la propiedad asociada al nodo con los valores c, d y e. El multi nodo se descompondría en 3 nodos de la red bayesiana (Ac, Ac y Ad) con dos estados cada uno: Ac(verdadero=0.5, falso= 0.5), Ad(verdadero=0.3, falso=0.7) y Ae(verdadero=0.1, falso=0.9). Nótese que las probabilidades de los estados del nodo A no tienen que sumar uno ya que el nodo se divide en otros nodos cuya probabilidad es la del estado definido en A para los casos que sea cierto y uno menos dicha probabilidad para los casos falsos. Si existiese una dependencia entre el nodo anterior y

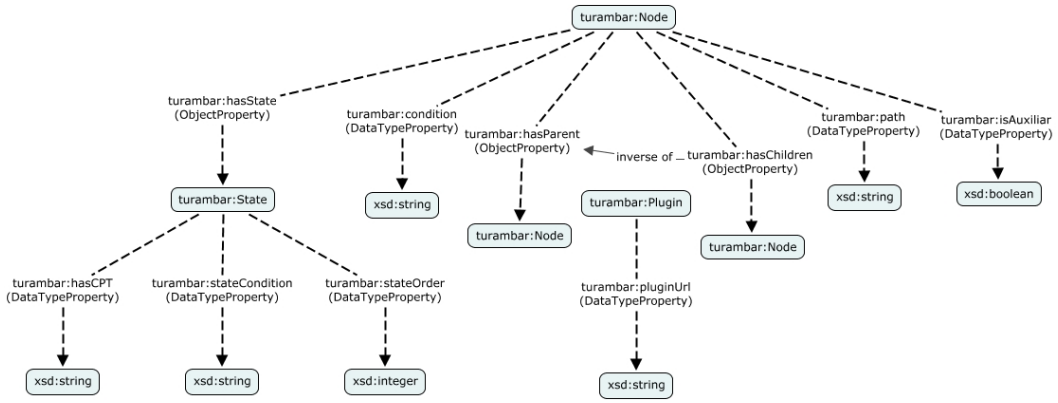


Figura 4.4: Ontología de Turambar de la segunda versión de nuestra propuesta

un hipotético nodo B ordinario tal que A está conectado por un arco dirigido con B se traduciría como Ac ->B, Ad->B y Ae->B. Si la relación fuese tal que B->A, el grafo resultante sería B->Ac, B->Ad y B->Ae.

- Nodos ordinarios. Son aquellos que se relacionan con una propiedad funcional o clase. Estos nodos no se descomponen en otros subnodos, por lo que equivalen a un único nodo de la red bayesiana.

Otra posible clasificación para los nodos es su división entre nodos finales, aquellos que generan aserciones, y finales, no generan aserciones. Todos los nodos finales son conocidos, pero no todos los nodos conocidos son finales.

Las relaciones entre nodos de la red definida en la ontología de dependencias se expresan mediante el vocabulario de la ontología Turambar. Esta ontología define las clases y propiedades que muestra la figura 4.4:

- Clase *Node*. Representa una variable aleatoria de la red bayesiana. Sus propiedades son las siguientes:
  - *hasState* define los estados de un nodo.
  - *path* selecciona el individuo adecuado para el nodo. Todos los nodos relacionados de manera jerárquica tienen que tener un nexo de unión: un individuo común con el cual están todos relacionados. La propiedad es opcional si solo se requiere un individuo en todos los nodos de la red.

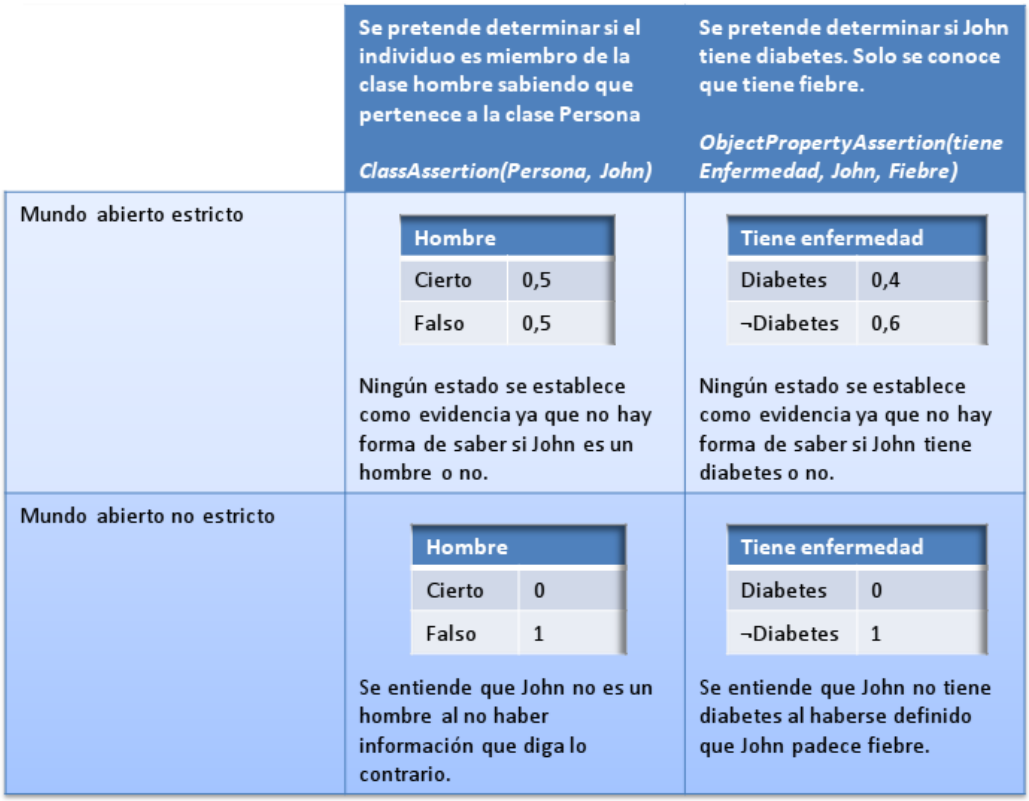


Figura 4.5: Diferencias entre el modo de mundo abierto estricto y no estricto en la segunda versión de Turambar

En caso contrario habrá que buscar el nexo de unión. Por ejemplo, si el nodo A, que hace referencia a la propiedad *esDeLaClaseSocial*,tiene la propiedad *path* y se pide la probabilidad de que el individuo I tenga esa propiedad, entonces se debe de buscar el nexo común del nodo A con los otros nodos de la red para poder propagar las evidencias. En caso de conocer el nexo común al evaluar los estados de ese nodo habría que hallar el individuo correcto para el nodo mediante la expresión definida por el valor de esta propiedad. El valor de esta propiedad es una expresión que sigue el formato *selected (<- propiedad de objeto)+ <- ?*, donde selected hace referencia al individuo requerido por el nodo actual, *(<- propiedad de objeto)+* al conjunto de propiedades de objeto a través de los que se relaciona el individuo del nodo actual con el nexo

común y el símbolo de interrogación identifica al nexo común.

- *condition* establece la condición que ha de cumplir el individuo del cual se van a evaluar sus propiedades en los estados del nodo. Es decir la condición necesaria que ha de cumplir el individuo del nodo actual para poder calcular la aserción probabilística. En el núcleo de Turambar solo se incluía un built-in para evaluar si el individuo pertenecía a una clase. No obstante se pueden desarrollar más built-ins implementando la interfaz ConditionBuiltIn y registrándolos en el razonador mediante los individuos de la clase Plugin.
- *isAuxiliar* indica si un nodo puede devolver aserciones probabilísticas o no.
- *hasParent* sirve para definir que nodos son padre del nodo actual. La propiedad inversa es *hasChildren*.
- *hasChildren* sirve para indicar que nodos son hijo de un nodo. La propiedad inversa es hasParent. El orden de los padres de un nodo es inverso al orden alfabético. Esto es importante tenerlo en cuenta de cara a la generación de las tablas de probabilidad condicionada. or ejemplo si el nodo B y C son padres de A y todos los nodos tienes dos estados (T y F), a la hora de definir el CPT de A sería tal como muestra la tabla 4.1:

- Clase *State*. Define el estado de un nodo, así como sus probabilidades. Los individuos de esta clase se relacionan con las siguientes propiedades:
  - *hasCPT*. Indica las probabilidades para el estado en concreto en función de los posibles valores de los padres. Por ejemplo en el caso anterior el estado T del nodo A tendría el siguiente valor: 0, 0.5, 0.2, 0.4;.
  - *stateOrder*. Define el orden del estado dentro de la tabla de probabilidad condicionada. El valor debe de ser un número entre 0 y n-1, donde n es el número de estado del nodo.
  - *stateCondition*:. Expresa la condición que se ha de cumplir para tomar como cierto el estado actual. La condición viene definida como una función de evaluación que comprueba si el nodo cumple la condición

C	*B*	*A (T)*	*A (F)*
T	*T*	0	1
T	*F*	0.5	0.5
F	*T*	0.2	0.8
F	*F*	0.4	0.6

Cuadro 4.1: Tabla de probabilidad condicionada de la segunda versión de Turambar para un nodo A que tiene de padres el nodo C y B, teniendo todos ellos dos estados verdadero (T) y falso (F).

definida en la función y retorna la aserción correspondiente a la que posteriormente se le asignará una probabilidad. Estas funciones se las conoce como built-ins de estado.

- Clase Plugin. Sirve para importar las librerías requeridas por la ontología probabilística. La ubicación de las librerías se define mediante el uso de la propiedad *pluginUrl*. Los plugins añaden nuevos built-ins para estados y/o condiciones. Por defecto, en esta versión solo se incluían los siguientes built-ins de estado:
  - *BooleanComparisonBuiltIn* compara si dos valores booleanos son iguales.
  - *IndividualComparisonBuiltIn* comprueba si dos individuos son iguales.
  - *NumberComparisonBuiltIn* comprueba si dos valores numéricos son iguales.
  - *RangeNumberComparisonBuiltIn* comprueba que un número este dentro de un rango númerico.
  - *StringComparisonBuiltIn* compara dos cadenas de texto.
  - *SubClassBuiltIn* comprueba si un individuo es miembro de una clase.

  De los anteriores built-ins solo el último no genera aserciones.

En esta versión también se introducen tres tipos de aserciones probabilísticas:

- Aserciones probabilísticas de valor único. Son aserciones de pertenencia de un individuo a una clase o que definen la relación entre un individuo y otro individuo o literal. Por ejemplo, las siguientes aserciones definen la probabilidad de que John sea un paciente y viva en Bilbao.

$$\begin{aligned} Paciente(John)\,0{,}5 \\ viveEn(John, Bilbao)\,0{,}8 \end{aligned} \tag{4.1}$$

- Aserciones probabilísticas de rango. Son aserciones que indican que el valor de una propiedad de datos se encuentra dentro de un rango de valores, pero no se puede determinar el valor exacto. Por ejemplo, John tiene una edad entre 65 y 70 años.

$$edad(John, [65, 80])0{,}8 \tag{4.2}$$

- Aserciones probabilísticas de rango abierto. Son iguales que las aserciones probabilísticas de rango con la salvedad que uno de sus valores límite es el infinito negativo o positivo.

$$tieneDinero(John, [65, \infty])0{,}8 \tag{4.3}$$

Finalmente, se introducen dos métodos diferentes de calcular las propiedades de los axiomas: el modo de mundo abierto estricto y el modo no estricto de mundo abierto. En el primer modo se marcarán como evidencias los estados cuya función de evaluación se cumple basándose exclusivamente en las evidencias de la ontología. Se diferencia del modo no estricto de mundo abierto, en que en este último cuando un individuo esté relacionado con al menos un valor a través de una propiedad no funcional se marcan como falso los estados de los nodos cuyo valor de propiedad no se cumpla y que si un individuo no está definido como miembro de una clase, entonces se asume que no es miembro. La figura 4.5 muestra un ejemplo del funcionamiento de ambos modos.

Con esta iteración se solventan la mayoría de problemas presentes en el ejemplo de la primera iteración. Por una parte, cada nodo puede estar relacionado con un individuo diferente. Por otra, existe la posibilidad de definir un contexto. Además,

con la incorporación de las extensiones se podría implementar una extensión propia que comprobase la existencia de nodos similares en la ontología e incluso que preguntase a un servicio externo. También es posible definir que el nodo ”es similar” no debe de devolver una aserción. En cuanto, a la definición del contexto ahora es posible, aunque su expresividad es limitada. Por ejemplo, no se podría definir en el contexto que el paciente tiene que vivir en una vivienda de un tipo determinado. Finalmente, este ejemplo sigue presentando algunas limitaciones a la hora de definir las redes bayesianas, que hacen que la modificación de valores y dependencias sea complicada.

Además de los inconvenientes extraídos del ejemplo, existen algunos nuevos que se han introducido en esta iteración. Todos ellos se recogen en la siguiente lista:

- Las propiedades no funcionales presentan el inconveniente de que no está claro como modelar la relación de dependencia entre diferentes valores de una propiedad. Por ejemplo, si el nodo Vehículo hace referencia a la propiedad *poseeUnVehículo* y tiene los estados Moto, Coche y Bicicleta puede darse el caso en el que tener un coche haga más probable que el usuario también tenga una bicicleta. Para ello habría que anotar múltiple veces la misma propiedad haciendo referencia a distintos nodos, lo cual no es una solución demasiado limpia ni clara a priori. Lo mismo ocurre si se quiere modelar que el tener un coche o una moto influye al gasto de energía.

- La representación de las tablas de probabilidad condicionada sigue siendo poco clara. Depender del orden alfabético para una correcta construcción de las tablas no es una solución muy extensible, ni mantenible.

- Las diferencias entre los modos de funcionamiento de mundo abierto estricto y mundo abierto no estricto son difíciles de entender por los usuarios.

- La definición del contexto entre los nodos es compleja. Se hace uso de un lenguaje propio que puede resultar confuso. Además, el contexto se debe de definir por cada nodo de la red indicando siempre el punto de enlace común entre los nodos. Por otra parte, su definición tendría que ser más expresiva permitiendo condicionar individuos que no estén relacionados con los nodos.

Por ejemplo, indiciar que uno de los individuos que sirven de enlace entre el individuo del nodo y el nexo de unión es miembro de una clase.

Con el objetivo de solucionar estos problemas se desarrolló la última y definitiva versión de Turambar: la versión tres.

4.3 Tercera iteración: definición final del sistema de representación y de la inferencia a realizar

La tercera versión de Turambar propone una solución a los problemas presentes en las anteriores versiones. Además, supone una simplificación del funcionamiento de Turambar al abandonar los dos modos de funcionamiento y decantarse por el modo estricto; así como por la simplificación de tipos de nodos. La decisión del modo estricto como único modo de funcionamiento responde a que este se identifica más con el paradigma de mundo abierto de OWL, por lo que es más apropiado.

4.3.1 Objetivo

El objetivo de esta nueva versión es extender las aserciones positivas de propiedades de objeto (*positive object property assertion*), las aserciones negativas de propiedades de objeto (*negative object property assertion*), las aserciones de clases (*class assertion*), las aserciones positivas de propiedades de datos (*positive data property assertion*) y las aserciones negativas de propiedades de datos (*negative data property assertion*) de OWL 2, ver figura 4.6. La extensión de OWL 2 dota al sistema de modelado una mayor integración con OWL 2, al mismo tiempo que clarifica el objetivo del sistema mismo.

Estas aserciones probabilísticas no pueden ser definidas de manera explícita, por lo que solo pueden ser inferidas. Por otra parte, estas aserciones presentan una serie de limitaciones respecto a las aserciones que extienden:

- La expresión de clase empleada en las aserciones probabilísticas de clase debe de ser una clase.

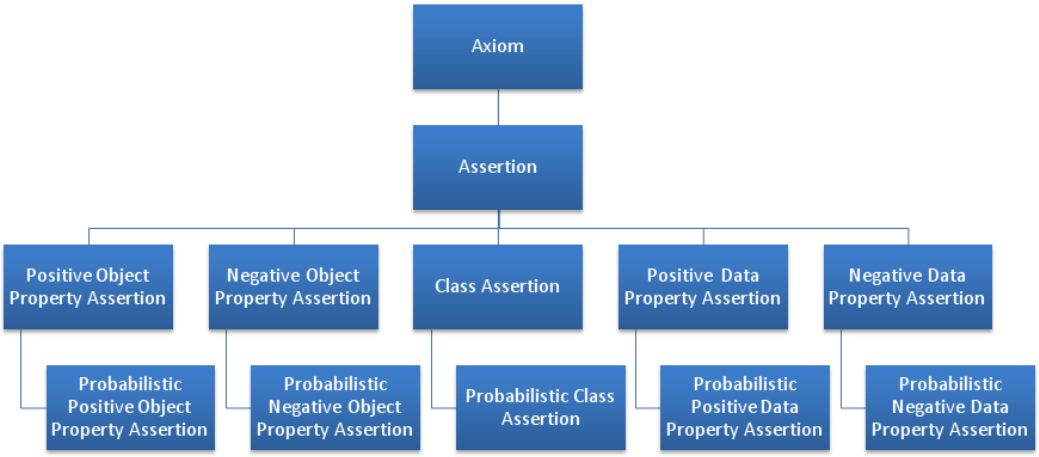


Figura 4.6: Aserciones probabilísticas disponibles en la tercera versión de Turambar

- La expresión de propiedad de objeto empleada debe de ser una propiedad de objeto.

Estas limitaciones pueden ser superadas mediante el uso de equivalencias, por lo que su existencia no es demasiado importante.

4.3.2 La ontología probabilística de Turambar

La definición de ontología probabilística en Turambar es la misma que en la versión anterior.

Definición 1 *Una ontología probabilística es una ontología ordinaria OWL 2 DL vinculada a través de las anotaciones de Turambar con una segunda ontología, en la que se describe mediante una red bayesiana las relaciones de dependencias entre aserciones de clases y propiedades.*

La figura 4.7 muestra las relaciones en la ontología OWL 2 DL y la ontología de dependencias a la cual está vinculada. Esta vinculación se produce a través de la ontología de anotaciones de Turambar que define las siguientes anotaciones:

- *turambarOntology* define la ubicación de la ontología de dependencias mediante una IRI.

- *turambarProperty* empareja directa o indirectamente una propiedad de objeto o propiedad de dato con un nodo de la red bayesiana.

- *turambarClass* establece una relación directa entre una clase y un nodo de la red bayesiana.

El valor de las anotaciones *turambarClass* y *turambarProperty* es una IRI que identifica a un individuo de la clase Node en la ontología de dependencias. Por su parte, la ontología de dependencias importa la ontología de Turambar para definir la red bayesiana.

4.3.3 Funcionamiento básico

El funcionamiento de Turambar es similar al de la versión 2: relaciona una ontología OWL 2 DL con un modelo probabilístico a través del cual calcular la probabilidad asociada a las aserciones probabilísticas. Para ello, se asocian las clases, propiedades de datos y propiedades de objetos con nodos de una red bayesiana. La parte del razonamiento tradicional de OWL se realizará mediante Pellet o HermiT. La elección de uno u otro es configurable.

La representación de las aserciones probabilísticas se puede considerar una extensión de la sintaxis funcional de OWL 2 (Patel-Schneider *et al.*, 2012b):

$$ProbabilisticClassAssertion(Clase\ Sujeto\ Probabilidad) \quad (4.4)$$

$$ProbabilisticObjectPropertyAssertion(PropiedadDeObjetoSujeto\ Individuo\ Probabilidad) \quad (4.5)$$

$$ProbabilisticNegativeObjectPropertyAssertion(PropiedadDeObjeto\ Sujeto\ Individuo\ Probabilidad) \quad (4.6)$$

$$ProbabilisticDataPropertyAssertion(PropiedadDeDatos\ Sujeto\ Literal\ Probabilidad) \quad (4.7)$$

$$ProbabilisticNegativeDataPropertyAssertion(PropiedadDeDatos\ Sujeto\ Literal\ Probabilidad) \quad (4.8)$$

En estas aserciones $Clase$ representa a una clase; $PropiedadDeObjeto$, a una propiedad de objeto; $PropiedadDeDatos$, a una propiedad de datos; $Sujeto$ e $Individuo$, a un individuo; $Literal$, a un literal y $Probabilidad$, a un valor comprendido en el rango entre cero y uno. Por su parte la aserción 4.4 es un aserción probabilística de clase; 4.5, una aserción probabilística positiva de propiedad de objeto; 4.6, una aserción probabilística negativa de propiedad de objeto; 4.7, una aserción probabilística positiva de propiedad de datos; y 4.8, una aserción probabilística negativa de propiedad de datos.

Los estados de los nodos de la red representan el conjunto de valores que una propiedad puede tener o la clase a la que un individuo puede pertenecer o no. Estos valores se definen mediante el uso de funciones que calculan el valor final de la aserción y comprueban si la condición representada por un estado existe en la ontología. Por ejemplo, una función podría ser ”$>$5”que comprueba si un individuo tiene una propiedad de datos con un valor superior a cinco. Algunas de estas funciones generan valores que pueden ser usados para la construcción de aserciones probabilísticas y otras no, como el caso anterior.

Todos los nodos de la red bayesiana están asociados a una clase o propiedad, en cambio una clase o propiedad puede estar asociada a más de un nodo de la red bayesiana. Esto es posible gracias a la definición del contexto y la existencia de propiedades no funcionales cuyos valores pueden estar condicionados por la existencia de esa misma propiedad con otros valores. El contexto representa la relación de los diferentes nodos de la red entre ellos y sirve para seleccionar que red bayesiana es la más adecuada para un individuo o si las redes disponibles son válidas para un individuo. Por ejemplo, el contexto podría definir que una red es válida para aquellos individuos que sean miembros de la clase *Paciente* y tengan un móvil. Para su definición se utiliza el lenguaje de consultas SPARQL-DL (Sirin & Parsia, 2007). El resultado de la consulta debe de ser univoco es decir el individuo por el que se pregunta no puede aparecer dos veces en los resultados como respuesta a una variable. Siguiendo el ejemplo, si la consulta retornase un individuo John y un móvil LG, entonces se podría proceder al cálculo de la probabilidad. Por el contrario, si tuviese dos móviles no. La relación de los nodos con el contexto se hace relacionando los nodos de la red con una de las variables de respuesta de la consulta SPARQL-DL, ver capítulo cinco.

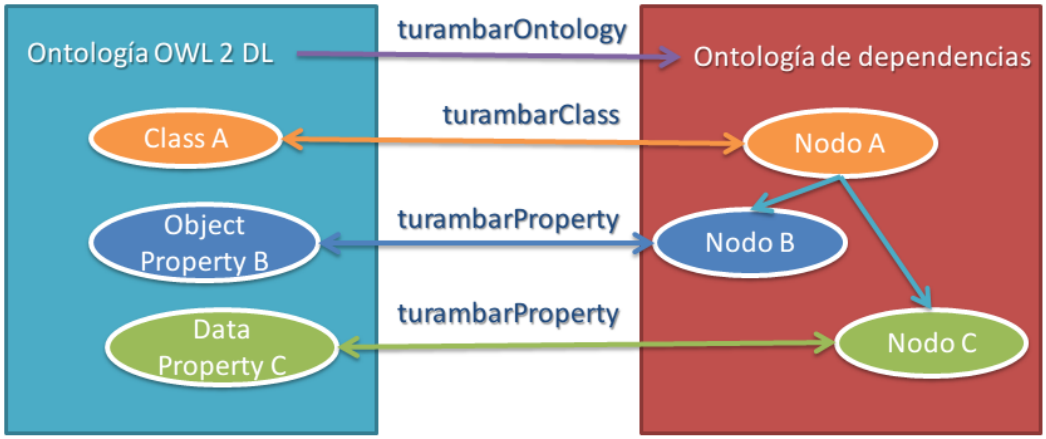


Figura 4.7: Relación entre una ontología OWL 2 DL vinculada a una ontología de dependencias en la tercera versión de Turambar.

A la hora de calcular la probabilidad de una aserción se pueden dar cuatro casos:

- Que la propiedad o clase esté relacionada con un nodo de la red bayesiana y exista una aserción en la ontología. En ese caso la probabilidad será 1.
- Que la propiedad o clase esté relacionada con un nodo de la red bayesiana y no exista una aserción en la ontología. Se deberá proceder al cálculo de la probabilidad y los valores probables según la red bayesiana.
- Que la propiedad o clase no esté relacionada con un nodo de la red bayesiana y exista una aserción en la ontología. En ese caso la probabilidad será 1.
- Que la propiedad o clase no esté relacionada con un nodo de la red bayesiana y no exista una aserción en la ontología. Se considera desconocido cualquier tipo de aserción.

La suma de las probabilidades de los estados de un nodo será siempre 1, siendo la probabilidad de cada estado un valor entre 1 y 0. No obstante, la suma de las aserciones probabilísticas para una propiedad puede ser mayor que uno debido a existencia de probabilidades no funcionales que están relacionadas con más de un nodo de la red al mismo tiempo, a pesar de compartir contexto.

Por último, se considera que una ontología es consistente si la ontología OWL 2 DL es consistente y las probabilidades de los nodos de la red bayesiana están

comprendidas entre 0 y 1, a la vez que la suma de las probabilidades de los estados de cada nodo de la red suma 1.

4.3.4 Modelado de la probabilidad

Una vez anotadas las clases y propiedades con los nombres de los nodos con los que se relacionan en la red bayesiana, se procede al modelado de la misma. Para expresar las relaciones de dependencia entre clases y propiedades se usa el vocabulario definido por la ontología de Turambar, ver figura 4.8. Por tanto, esta ontología debe de ser importada de manera obligatoria por la ontología de dependencias. Esta nueva versión es bastante similar a la de la versión 2. No obstante, ofrece notables mejoras respecto a la definición de las tablas de probabilidad condicionada y a la definición del contexto de los nodos.

Las clases y propiedades definidas por la ontología son las siguientes:

- *Node* representa un nodo de la red bayesiana. Sus instancias se definen como auxiliares a través de la propiedad *isAuxiliar*. Se relacionan con su tabla de probabilidad condicionada mediante la propiedad de objeto *hasProbability-Distibution*. Los posibles valores que puede tomar la aserción probabilística (los estados del nodo) se definen mediante la probabilidad de objeto *hasS-tate*. Por su parte, la jerarquía de la red se expresa mediante el uso de las propiedades de objeto *hasParent* y su inversa *hasChildren*. Estas describen los nodos padres de un nodo y sus hijos, respectivamente. El contexto de un nodo se define mediante el uso de las propiedades de objeto *hasContext*, para identificar la expresión de contexto; y *hasVariable*, para relacionar el nodo con una variable del contexto. No todos los nodos han de tener contexto, por lo que puede no aparecer.

- *MetaNode* es un tipo especial de nodo creado especialmente para las propiedades no funcionales. Esta clase agrupa a diversas instancias de la clase *Node* mediante el uso de la propiedad *compriseNode*. Su objetivo es facilitar el modelado de relaciones de dependencia entre valores de una propiedad. Por ejemplo, si existiese una propiedad de objeto no funcional *tienenEnfermedad* y se quisiese expresar que aquellos individuos con la enfermedad *A*

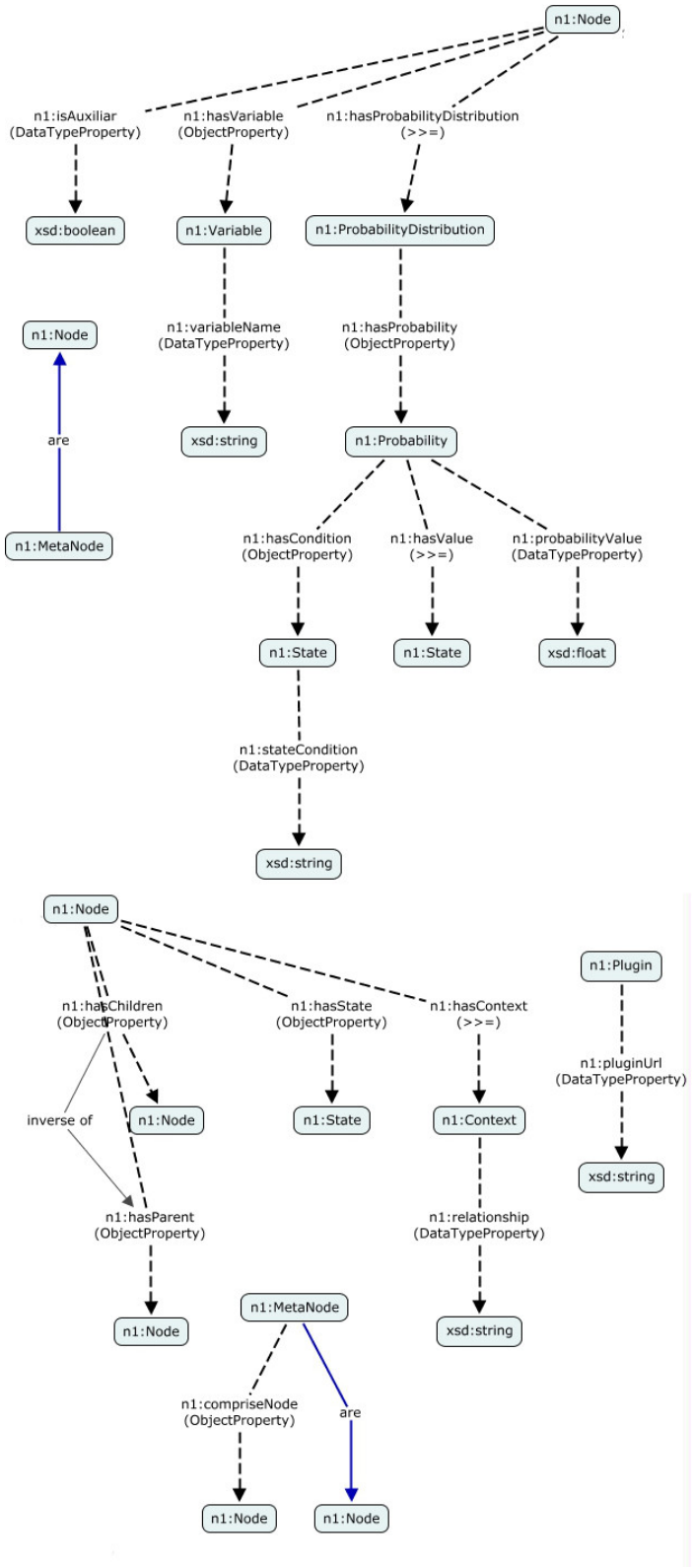


Figura 4.8: Ontología de Turambar en la tercera iteración de Turambar

son más propensos a tener la enfermedad *B*, entonces esto se podría modelar como un *MetaNode* compuesto por dos individuos de la clase *Node* (asumiendo que no existen más enfermedades posibles). Este nodo es el causante de que cuando se hable de que la anotación *turambarProperty* pueda relacionar indirectamente un nodo de la red bayesiana con una propiedad de objeto o de datos.

- *State* representa un valor del dominio de un nodo de la red bayesiana. Mediante el uso de la propiedad de datos *stateCondition* se define la función correspondiente que calcula el valor de la aserción correspondiente, a la vez que evalúa si el individuo ya está relacionado con ese valor.

- *ProbabilityDistribution* define una distribución de probabilidad en forma de tabla de probabilidad condicionada. Se asocia con las celdas de la tabla mediante el uso de la propiedad de objeto *hasProbability*.

- *Probability* representa una celda de la tabla de probabilidad condicionada. Se relaciona con el estado cuya probabilidad se quiere expresar mediante la propiedad de objeto *hasValue* y con los estados de los cuales depende a través de la propiedad de objeto *hasCondition*. La probabilidad se define mediante el uso de la propiedad de datos *hasProbabilityValue*.

- *Context* establece la relación que tienen en común los diferentes nodos de la red bayesiana. La relación se expresa mediante una consulta SPARQL-DL por medio de la propiedad de datos *relationship*.

- *Variable* define una variable utilizada en una consulta SPARQL-DL a la hora de establecer el contexto de una red bayesiana. Se utiliza la propiedad de datos *variableName* para identificar la variable de la consulta.

- *Plugin* es la clase que representa la necesidad de cargar librerías adicionales que provean funciones adicionales utilizadas en la definición de los estados. La propiedad de datos *pluginUrl* es utilizada para expresar la localización de la librería utilizada.

4.3.5 Funciones de estado y tipos de nodos

En esta versión los tipos de nodos han sido reducidos a dos grupos:

- Finales son aquellos nodos relacionados con propiedades o clases que retornan una aserción probabilística. Todos los nodos relacionados con clases son finales.
- Auxiliares son aquellos nodos relacionados con propiedades que no devuelven una aserción probabilística.

Las funciones de estado son las encargadas de generar el valor de una aserción probabilística y comprobar que el estado de un nodo sea cierto o no. No todas las funciones tienen que devolver un valor válido que se pueda utilizar para las aserciones probabilísticas. Las funciones de estado consideradas como parte de la especificación de Turambar son las siguientes:

1. Comparaciones booleanas se utilizan para comprobar si dos valores booleanos son el mismo.
2. Comparaciones numéricas comprueban si dos valores numéricos son iguales.
3. Comparaciones de cadenas de texto examinan si dos cadenas de texto son idénticas.
4. Comparaciones de rango numérico comprueban que un valor numérico se encuentre en un rango o que sea menor, menor o igual, mayor o igual o mayor que un determinado número.
5. Comparaciones de rango de cadenas de texto comprueba si una cadena de texto está en un rango, es mayor, mayor o igual, menor o igual o menor que una cadena de texto.
6. Comparación de individuos comparan si dos individuos son iguales.
7. Comprobación de membresía de clase examina si un individuo es miembro de una clase.

8. Aserción nula indica que se desconocen aserciones. Aunque no genera aserciones su uso, no se considera que su aparición convierta a un nodo en auxiliar.

Todas las funciones, a excepción de la función aserción nula, tienen su inversa. Las funciones inversas comprueban que no tienen un valor que cumpla la condición de la función normal. No obstante solo las funciones 1, 2, 3 y 6 generan aserciones probabilísticas.

Si uno de los estados de los nodos hace uso de una función que no produce aserción, entonces todo el nodo se considera auxiliar. Por el contrario, un nodo también se considera auxiliar si se ha definido como tal mediante el uso de la propiedad *isAuxiliar*.

4.3.6 Diferencias con la versión 2

Las principales diferencias con la anterior versión son:

- Eliminación de los modos de mundo abierto estricto y no estricto. Ahora solo existe el primero.
- Mejora en la definición de tablas de probabilidad condicional mediante un modelo más flexible y más simple de leer y comprender.
- Sustitución del lenguaje propio para definir el contexto por el lenguaje de consultas SPARQL-DL. De esta manera, se ofrece una mayor expresividad para su definición.
- Se ha clarificado el modelado de las dependencias entre valores de una misma propiedad no funcional.
- Las aserciones probabilísticas de rango y rango abierto ya no forman parte de la especificación. Es decir, ningún resultado a una consulta puede devolver una aserción de esos tipos.

Estas diferencias permiten modelar el ejemplo de la toma de medicación. Por una parte, se superan los problemas de definición de contexto existentes en la versión anterior gracias al uso de SPARQL-DL. Ello permite definir contextos más expresivos. Por otra parte, la definición de redes y probabilidades es más clara y sencilla tanto de describir como de actualizar.

4.4 Conclusiones

En este capítulo se ha presentado el proceso evolutivo que ha sufrido Turambar a lo largo del tiempo hasta llegar a su versión definitiva. Este proceso es importante ya que ayuda a entender las decisiones tomadas a lo largo de este trabajo. Además, se presenta el método de representación de incertidumbre propuesto, así como el tipo de inferencia que se puede esperar de este. Este sistema ofrece una solución para la representación de la incertidumbre en entornos inteligentes que combine aserciones probabilísticas de clases y propiedades. Por otra parte, cumple con los objetivos de IRI única y aislamiento del modelo probabilístico del conocimiento tradicional. Siendo estas dos últimas características propias de esta propuesta. Hasta donde se tiene conocimiento, no existe ninguna otra propuesta que tenga en cuenta esto.

Aunque en este capítulo se ha explicado el funcionamiento básico que se puede esperar de la propuesta, queda por definir cómo consultar el conocimiento probabilístico. El lenguaje de consultas que se presenta en el siguiente capítulo es importante ya que ofrece los métodos necesarios para acceder al conocimiento probabilístico.

Cuando la situación es adversa y la esperanza poca, las determinaciones drásticas son las más seguras.

Tito Livio

CAPÍTULO

5

Consultando las ontologías probabilísticas

A lo largo del viaje el explorador consultará los mapas y anotaciones que preparó antes de partir a la aventura. Tanto los mapas como las anotaciones están cuidadosamente ordenados para facilitar la búsqueda de la información. De poco le sirve al explorador disponer de la información si no es capaz de acceder a ella.

En este capítulo se describe como Turambar responde a una consulta sobre una aserción probabilística. Además, se define el lenguaje de consultas declarativo creado para el razonador. Este lenguaje es una extensión de SPARQL-DL, cuyo objetivo es facilitar la consulta de ontologías probabilísticas. La principal contribución de este lenguaje es ofrecer un método de consulta declarativa para ontologías probabilísticas que permita hacer consultas complejas, y cuya sintaxis resulte familiar a los desarrolladores. Una ventaja de las consultas SPARQL-DL es que las consultas pueden ser reutilizadas entre diferentes implementaciones del lenguaje de consultas. Es decir, la misma consulta podría ser utilizada en un motor de consultas en C++ y en Java sin modificarla, y el resultado sería el mismo. En cambio, si se ofreciese únicamente una API las consultas complejas deberían de ser realizadas mediante múltiples llamadas a métodos de la API, además de tener que procesar

los resultados de esas llamadas. Por otra parte, si se quisiese implementar Turambar en otro lenguaje de programación, como C++, esas consultas deberían de ser traducidas a llamadas a la nueva API de C++.

5.1 Respondiendo una consulta probabilística

A la hora de responder una consulta sobre una aserción probabilística se seguirán los siguientes pasos que muestra la figura 5.1:

1. Comprobar que la consulta no se haya realizado antes y el resultado esté calculado ya.

2. Comprobar que el sujeto existe en la ontología OWL 2 DL.

3. A partir de este punto el camino se divide en dos dependiendo de si se desea conocer las clases a las que pertenece un individuo o el valor de una propiedad.

   (a) Propiedades

      i. Se procede a comprobar si el individuo cumple con el contexto. En el caso de que hubiese varios nodos asociados con una propiedad y con diferentes contextos válidos para el nodo, se escogería siempre el contexto más restrictivo. Se asume que el contexto más restrictivo es aquel con mayor número de variables implicadas. En caso de igualdad de variables, se escoge el primero que sea válido.

      ii. Una vez escogido el nodo, se sustituye la variable asociada al nodo por el individuo retornado por la consulta de contexto. En el caso de los meta nodos, esta operación se realiza por cada uno de los nodos que conforman el meta nodo. Repitiendo los siguientes pasos por cada nodo del meta nodo.

      iii. A continuación se comprueba que los estados del nodo, estableciendo como evidencia aquel estado que cumpla. Una vez comprobadas las evidencias para un nodo, este se marca como revisado.

iv. Acto seguido se procede a analizar el resto de nodos en la jerarquía de la red, sustituyendo las variables por individuos y marcando las evidencias.

v. Comprobar que los nodos pendientes del paso (ii) se hayan revisado también. En caso de no haber sido revisados volver al paso (iii).

vi. Se calcula la probabilidad de los nodos relacionados con la consulta. Se retornan las aserciones generadas por los estados de los nodos y su probabilidad asociada. Las aserciones y sus probabilidades se añaden como resultados a la consulta.

(b) Clases. Este caso es particular debido a que se proveen dos formas de calcular la probabilidad de la pertenencia a una clase. La primera que es la más sencilla es cuando se pregunta por una clase en concreto, por ejemplo la probabilidad de que John sea un paciente. En estos casos los pasos seguidos son los mismos que para las propiedades. Sin embargo, existe la posibilidad de que sea una consulta de tipo y se pregunte por todas las clases a las que pertenece John, en cuyo caso el método de cálculo es ligeramente diferente. Este método se describe a continuación.

i. Se buscan todas las clases relacionadas con un nodo.

ii. Por cada clase se aplica el mismo proceso que para una propiedad.

4. Retorno de la lista de resultados. Si no se han hallado aserciones probabilísticas el resultado será vació.

El atento lector habrá observado que si se preguntase por el valor de una propiedad o la membresía a una clase que no estuviera relacionado con un nodo, el resultado será vacío cuando debería de ser la aserción existente en la ontología asociada a una probabilidad con valor 1. La razón por la que este algoritmo no refleja esos casos es porque se da por hecho que este se llama solo si la información existente en la ontología no se ha cumplido.

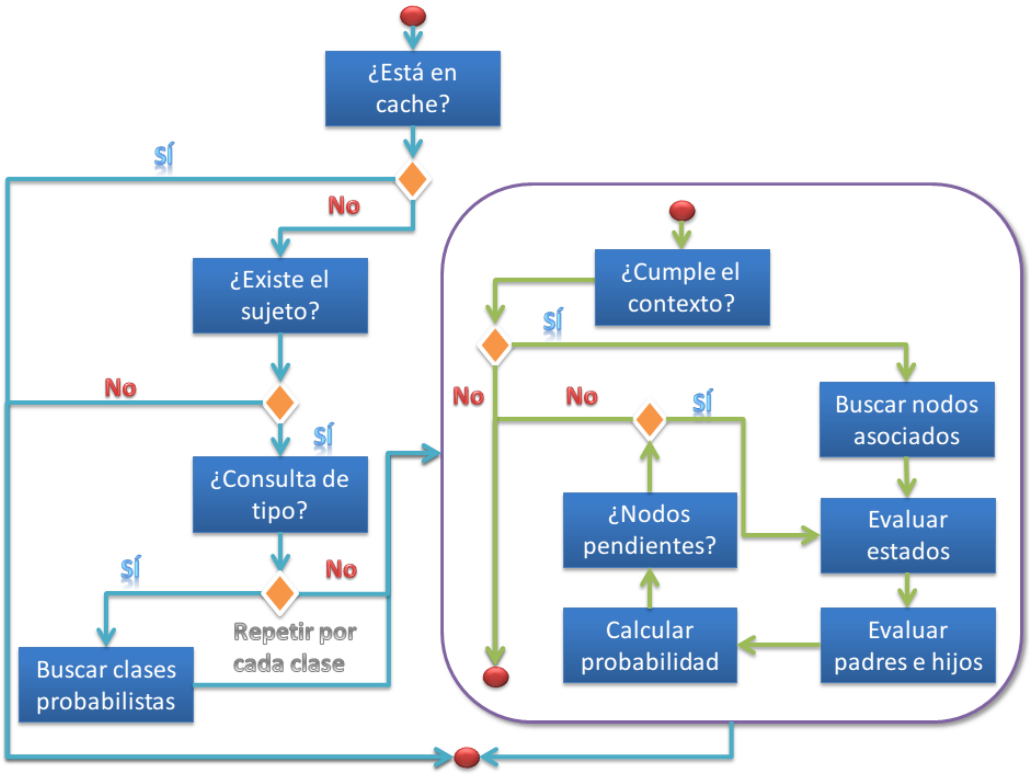


Figura 5.1: Algoritmo que muestra los pasos que se siguen a la hora de calcular una aserción probabilística

5.1.1 Lenguaje de consultas en Turambar

En Turambar existen dos formas de consultar el conocimiento probabilístico: mediante una API y mediante un lenguaje de consultas declarativo creado a partir de SPARQL-DL. La razón principal de incluir un lenguaje declarativo es que este se viese sujeto a menos cambios entre versiones y que pudiese ser implementado por terceros para sus propios sistemas. Las razones por las que se eligió extender este lenguaje y no SPARQL son:

- La creación de *query atoms* era más sencilla desde el punto técnico.
- El uso de *query atoms* evita errores que se podrían producir al añadir un cuarto miembro a los patrones de tripleta. Por ejemplo, que existiese un patrón que hiciese referencia a una consulta probabilística no soportada.
- Hasta donde conozco no existe una manera nativa de integrar SPARQL con OWL API (Clark & Parsia, 2014).

5.1.2 Introducción a SPARQL-DL

SPARQL-DL (Sirin & Parsia, 2007) tiene como principal objetivo ofrecer un lenguaje de consulta lo suficiente expresivo y versátil para las ontologías OWL DL, que permita consultar tanto TBox, RBox como ABox. SPARQL-DL es un subconjunto de SPARQL (Prud'hommeaux *et al.*, 2013) que es cubierto mediante los servicios de razonamiento estándar de un razonador OWL DL.

En (Sirin & Parsia, 2007) se describe el vocabulario de una ontología OWL DL como $\mathcal{V}_{\mathcal{O}} = (\mathcal{V}_{cls}, \mathcal{V}_{op}, \mathcal{V}_{dp}, \mathcal{V}_{ap}, \mathcal{V}_{ind}, \mathcal{V}_{\mathcal{D}}, \mathcal{V}_{lit})$; $\mathcal{V}_{bnode}$, el conjunto de identificadores de bnodes; y $\mathcal{V}_{var}$; donde $\mathcal{V}_{cls}$ es el conjunto de clases denotadas por URIs ; $\mathcal{V}_{op}$, el conjunto de propiedades de objetos identificadas por URIs; $\mathcal{V}_{dp}$, el conjunto de propiedades de datos definidas mediante una URI; $\mathcal{V}_{ind}$, el conjunto de URIs que nombran individuos ; $\mathcal{V}_{ap}$, el conjunto de anotaciones identificadas por una URI; $\mathcal{V}_{\mathcal{D}}$, el conjunto de tipos de datos definidos mediante URIs; $\mathcal{V}_{uri}$, el conjunto URIs; S_c el conjunto de las clases OWL DL; y $\mathcal{V}_{lit}$, el conjunto de literales bien formados. Del mismo artículo, se deprende que un *atom* de una consulta SPARQL-DL es de la siguiente forma:

$$q \leftarrow Type(a, C)|PropertyValue(a, p, v)|SameAs(a, b)|DifferentFrom(a, b)$$
$$EquivalentClass(C_1, C_2)|SubClassOf(C_1, C_2)|DisjointWith(C_1, C_2)$$
$$ComplementOf(C_1, C_2)|EquivalentProperty(p_1, p_2)|SubProperyOf(p_1, p_2)$$
$$InverseOf(p_1, p_2)|ObjectProperty(p)|DatatypeProperty(p)|Functional(p)$$
$$InverseFunctional(p)|Transitive(p)|Symmetric(p)|Annotations(s, p_a, o)$$

En el cual $a, b \in \mathcal{V}_{uri} \cup \mathcal{V}_{var} \cup \mathcal{V}_{bnode}$, $v \in \mathcal{V}_{uri} \cup \mathcal{V}_{lit} \cup \mathcal{V}_{var} \cup \mathcal{V}_{bnode}$. $p, q \in \mathcal{V}_{uri} \cup \mathcal{V}_{var}$; $C, D \in S_c \cup \mathcal{V}_{var}$, $s \in \mathcal{V}_{uri}$, $p_a \in \mathcal{V}_{ap}$, $o \in \mathcal{V}_{uri} \cup \mathcal{V}_{var}$. La consulta SPARQL-DL sería un conjunto finito de estos *atoms* y es interpretada como la conjunción de estos.

La función de evaluación $\sigma : \mathcal{V}_{ind} \cup \mathcal{V}_{bnode} \cup \mathcal{V}_{lit} \rightarrow \Delta^{\mathcal{I}}$ empareja nombres de individuos, bnodes y literales utilizados en la consulta con elementos del dominio de la interpretación $\Delta^{\mathcal{I}}$ con el requisito de que si $\sigma(a) = a^{\mathcal{I}}$ si $a \in \mathcal{V}_{ind}$ o $a \in \mathcal{V}_{lit}$. Por su parte, una interpretación $\mathcal{I}$ satisface a un *semi ground atom* q ($\mathcal{I} \models_\sigma q$) de una consulta si es compatible con el vocabulario de la ontología y cumple las condiciones impuestas en la tabla 5.1.

La interpretación $\mathcal{I}$ satisface a una consulta $\mathcal{Q} = q_1 \wedge q_2 .. \wedge q_n$ si y solo si $\mathcal{I} \models_\sigma q_i$ por cada atom de la consulta. Finalmente se dice que $\mathcal{Q}$ es una consecuencia lógica de la ontología si la consulta es satisfacible por todos los modelos de la ontología.

Una solución para una consulta SPARQL-DL $\mathcal{Q} = q_1 \wedge q_2 .. \wedge q_n$ para una ontología OWL DL es un emparejamiento de variable $\mu : \mathcal{V}_{var} \rightarrow \mathcal{V}_{uri} \cup \mathcal{V}_{lit}$ tal que cuando todas las variables en $\mathcal{Q}$ se hayan sustituido con el valor correspondiente de μ obteniendo una consulta *semi-ground* $\mu(\mathcal{Q})$ compatible con el vocabulario de la ontología y tal que $\mathcal{O} \models \mu(\mathcal{Q})$. El conjunto de soluciones para una consulta $\mathcal{Q}$ es el conjunto de todas las soluciones.

5.1.3 Extensión de Derivo

El lenguaje de consultas presentado es una extensión de la variante de SPARQL-DL de Derivo (Derivo, 2014). Esta variante está completamente alineada con OWL 2 e incluye los siguientes *atoms*:

Atom de la consulta	*Condición en la interpretación*
Type(a,C)	$\sigma(a) \in C^{\mathcal{I}}$
PropertyValue(a,p,v)	$\langle \sigma(a), \sigma(v) \rangle \in p^{\mathcal{I}}$
SameAs(a,b)	$\sigma(a) = \sigma(b)$
DifferentFrom(a,b)	$\sigma(a) \neq \sigma(b)$
EquivalentClass(C_1, C_2)	${C_1}^{\mathcal{I}} = {C_2}^{\mathcal{I}}$
SubClassOf(C_1, C_2)	${C_1}^{\mathcal{I}} \subseteq {C_2}^{\mathcal{I}}$
DisjointWith(C_1, C_2)	${C_1}^{\mathcal{I}} \cap {C_2}^{\mathcal{I}} =$
ComplementOf(C_1, C_2)	${C_1}^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus {C_2}^{\mathcal{I}}$
EquivalentProperty(p_1, p_2)	$p^{\mathcal{I}} = q^{\mathcal{I}}$
SubProperyOf(p_1, p_2)	$p^{\mathcal{I}} \subseteq q^{\mathcal{I}}$
ObjectProperty(p)	siempre que sea compatible con $\mathcal{V}_{\mathcal{O}}$ será satisfacible
DatatypeProperty(p)	siempre que sea compatible con $\mathcal{V}_{\mathcal{O}}$ será satisfacible
Functional(p)	$\langle a, b \rangle \in p^{\mathcal{I}}$ y $\langle a, c \rangle \in p^{\mathcal{I}}$ entonces $c = b$
InverseFunctional(p)	$\langle a, b \rangle \in p^{\mathcal{I}}$ y $\langle c, a \rangle \in p^{\mathcal{I}}$ entonces $c = b$
Transitive(p)	$\langle x, y \rangle \in p^{\mathcal{I}}$ y $\langle y, z \rangle \in p^{\mathcal{I}}$ entonces $\langle x, z \rangle \in p^{\mathcal{I}}$
Symmetric(p)	$\langle x, y \rangle \in p^{\mathcal{I}}$ entonces $\langle y, z \rangle \in p^{\mathcal{I}}$
Annotations(s,p_a,o)	$\langle s, o \rangle \in {p_a}^{\mathcal{I}}$

Cuadro 5.1: Satisfacción de un atom de consulta respecto una interpretación. Tabla extraída de (Sirin & Parsia, 2007).

- *Class(a)* comprueba que *a* es una clase.
- *Property(a)* comprueba que *a* es una propiedad.
- *Individual(a)* comprueba que *a* es un individuo.
- *Type(a, b)* comprueba que *a* es un miembro de la clase *b*.
- *PropertyValue(a, b, c*) comprueba que *a* está relacionado a través de la propiedad *b* con *c*.
- *EquivalentClass(a, b*) comprueba si dos clases son equivalentes.
- *SubClassOf(a, b*) comprueba si *a* es subclase de *b*.
- *EquivalentProperty(a, b)* comprueba si dos propiedades son equivalentes.
- *SubPropertyOf(a, b)* comprueba si *a* es una subpropiedad de *b*.
- *InverseOf(a, b)* comprueba si la propiedad *a* es la inversa de *b*.
- *ObjectProperty(a)* comprueba si *a* es una propiedad de objeto.
- *DataProperty(a)* comprueba si la propiedad *a* es una propiedad de dato.
- *Functional(a)* comprueba que la propiedad a sea funcional.
- *InverseFunctional(a)* comprueba que la inversa de la propiedad *a* sea funcional.
- *Transitive(a)* comprueba que la propiedad *a* sea transitiva.
- *Symmetric(a)* comprueba que la propiedad *a* sea simétrica.
- *Reflexive(a)* comprueba que la propiedad *a* sea reflexiva.
- *Irreflexive(a)* comprueba que la propiedad a sea irreflexiva.
- *SameAs(a, b)* comprueba que *a* y *b* sean el mismo individuo.
- *DisjointWith(a, b)* comprueba que la clase *a* sea disjunta con *b*.

- *DifferentFrom(a, b)* comprueba que dos individuos son distintos.
- *ComplementOf(a, b)* indica que la clase *a* es la completo de *b*.
- *Annotation(a, b, c)* comprueba que el valor de la anotación *b* para la URI *a* es *c*.
- *StrictSubClassOf(a, b)* comprueba que *a* es una subclase estricta de *b*.
- *DirectSubClassOf(a, b)* comprueba que *a* es una subclase directa de *b* en el árbol de jerarquías.
- *DirectType(a, b)* comprueba que *a* es miembro directo de la clase *b*.
- *StrictSubPropertyOf(a, b)* comprueba que *a* es una subpropiedad estricta de *b*.
- *DirectSubPropertyOf(a, b)* comprueba que *a* es una subpropiedad directa de *b* en el árbol de jerarquías.

Esta extensión soporta dos tipos de consultas:

- ASK devuelve un valor booleano que informa de si la consulta tiene o no solución. Su sintaxis es la siguiente:

$$ASK\,\{\,atom_1,\,atom_2,\,....atom_n\,\}$$

- SELECT realiza una consulta y devuelve como resultado los posibles valores para las variables de la consulta. Su sintaxis es la siguiente;

$$\begin{aligned} SELECT\,[DISTINCT]\,?var_1\,?var_2\,...\,?var_n \\ [WHERE]\,\{\,atom_1,\,atom_2,\,....\,atom_n\,\} \\ [ORWHERE\,\{\,atom_1,\,atom_2,\,....\,atom_n\}]\,* \end{aligned}$$

También, se permite el uso de prefijos añadiendo al principio de la consulta la lista de prefijos y la URI a la que representan, tal que:

$$PREFIX\,prefijo:\,URI \tag{5.1}$$

5.1.4 Extensión propuesta

Partiendo de la implementación de Derivo, se han añadido dos nuevos *atoms* para consultar las ontologías de Turambar:

- *ProbType(?individual, ?class, ?probability)* es una extensión del atom *Type*. Sirve para obtener la probabilidad de que un individuo sea miembro de una clase.

- *ProbPropertyValue(?individual, ?property, ?value, ?probability)* es una extensión del atom *PropertyValue*. Determina la probabilidad de que exista una aserción probabilística que asocie un individuo con otro individuo o literal a través de una propiedad.

En ambos casos la propiedad se expresa como un valor entre 0 y 1, que puede ir precedido por los símbolos mayor que (>), menor que (<), mayor o igual que (>=) o menor o igual que (<=) para determinar el umbral de probabilidad mínimo o máximo aceptado. Esto puede ser muy útil para determinar las aserciones que se encuentran entre un rango de probabilidad determinado.

5.1.4.1 Ejemplos de consultas

La siguiente consulta muestra cómo recuperar la tarea actual que está haciendo una persona y la probabilidad de que esté realizándola.

```
PREFIX ex: <http://www.example.org/example.owl#>
SELECT ?task ?prob ?person WHERE {
        ProbType(?task, ex:TomarMedicina , ?prob),
        PropertyValue(?person, ex:tarea, ?task)
}
```

Si se quisiese determinar que las actividades del tipo *TomarMedicina* con una probabilidad mayor al ochenta por ciento son las que se deben de retornar, la consulta sería la siguiente:

```
PREFIX ex: <http://www.example.org/example.owl#>
SELECT ?task ?prob ?person WHERE {
```

```
        ProbType(?task, ex:TomarMedicina , ?prob),
        PropertyValue(?person, ex:tarea, ?task),
        ProbType(?task, ex:TomarMedicina , >0.8)
}
```

Si por el contrario, se desease recuperar las actividades del tipo *TomarMedicina* que se producen con un rango de probabilidad entre el cincuenta por cierto y el ochenta por ciento, entonces la consulta sería la siguiente:

```
PREFIX ex: <http://www.example.org/example.owl#>
SELECT ?task ?proba ?person WHERE {
        ProbType(?task, ex:TomarMedicina , ?proba),
        PropertyValue(?person, ex:tarea, ?task)
        ProbType(?task, ex:TomarMedicina , >=0.5),
        ProbType(?task, ex:TomarMedicina , <=0.8)
}
```

Si se quisiese conocer la probabilidad con la que una ingesta de medicina ha sido correcta, la consulta se describiría tal que:

```
PREFIX ex: <http://www.example.org/example.owl#>
SELECT   ?person ?ingesta ?prob WHERE {
        PropertyValue(?person, ex:ingesta, ?ingesta),
        Type(?ingesta, ex:Ingesta),
        ProbPropertyValue(?ingesta, ex:esCorrecta,
        "true", ?prob)
}
```

Al igual que con las propiedades, también se puede especificar que la probabilidad mínima con la que las ingestas se consideran correctas. Por ejemplo, ochenta por ciento:

```
PREFIX ex: <http://www.example.org/example.owl#>
SELECT   ?person ?ingesta ?prob WHERE {
        PropertyValue(?person, ex:ingesta, ?ingesta),
        Type(?ingesta, ex:Ingesta),
```

```
        ProbPropertyValue(?ingesta, ex:esCorrecta,
        "true", ?prob),
        ProbPropertyValue(?ingesta, ex:esCorrecta,
         "true", >0.8)
}
```

O también se podría indicar que solo interesan aquellas ingestas correctas con una probabilidad entre el ochenta y cincuenta por ciento:

```
PREFIX ex: <http://www.example.org/example.owl#>
SELECT   ?person ?ingesta ?prob WHERE {
        PropertyValue(?person, ex:ingesta, ?ingesta),
        Type(?ingesta, ex:Ingesta),
        ProbPropertyValue(?ingesta, ex:esCorrecta,
        "true", ?prob),
        ProbPropertyValue(?ingesta, ex:esCorrecta,
         "true", >0.5),
        ProbPropertyValue(?ingesta, ex:esCorrecta,
        "true", >0.8)
}
```

Los atoms ProbPropertyValue y ProbType pueden aparecer combinados en una misma consulta. Por ejemplo, para calcular la probabilidad de que una persona esté tomando la medicación con una probabilidad de entre el cincuenta y ochenta por ciento y la ingesta sea correcta con una probabilidad entre el cincuenta y el ochenta por ciento.

```
PREFIX ex: <http://www.example.org/example.owl#>
SELECT   ?person ?ingesta ?prob ?task ?proba WHERE {
        PropertyValue(?person, ex:ingesta, ?ingesta),
        Type(?ingesta, ex:Ingesta),
        ProbPropertyValue(?ingesta, ex:esCorrecta,
         "true",?prob),
        ProbPropertyValue(?ingesta, ex:esCorrecta,
        "true", >0.5),
        ProbPropertyValue(?ingesta, ex:esCorrecta,
```

```
        "true", >0.8),
        PropertyValue(?task, ex:tieneIngesta,
         ?ingesta),
        ProbType(?task, ex:TomarMedicina , ?proba),
        PropertyValue(?person, ex:tarea, ?task)
        ProbType(?task, ex:TomarMedicina , >=0.5),
        ProbType(?task, ex:TomarMedicina , <=0.8)

}
```

5.2 Conclusiones

En este capítulo se ha descrito el lenguaje declarativo de consultas que se usa para definir el contexto, así como la extensión realizada para consultar las ontologías probabilísticas. Además, se han ofrecido una serie de ejemplos que explican el funcionamiento básico del lenguaje de consultas. Así mismo, se ha explicado el algoritmo que sigue Turambar para calcular una consulta probabilística.

El lenguaje probabilístico presentado ofrece la principal ventaja de que permite realizar consultas complejas que combinen conocimiento probabilístico con el conocimiento de la ontología de una manera declarativa, abstrayendo al programador de la API del razonador. Esto permite que si algún otro razonador decidiese implementar el lenguaje de consultas propuesto, los desarrolladores pudiesen migrar las consultas de manera directa a este. Por otra parte, se simplifica la lógica de las aplicaciones que hagan uso de Turambar, ya que se delega a este las consultas complejas.

Explicado los componentes principales del trabajo realizado, el siguiente paso es evaluarlo. Por ello, en el siguiente capítulo se describen los experimentos que se han realizado con el fin de evaluar Turambar y la extensión del lenguaje de consultas SPARQL-DL.

No te pongas en el lado malo de un argumento simplemente porque tu oponente se ha puesto en el lado correcto

Baltasar Gracián

CAPÍTULO

6

Evaluación

Cada etapa del viaje a completar entraña peligros diferentes que el explorador deberá de evaluar durante el viaje. Los accidentes geográficos no siempre son como él esperaba y habrá que sopesar la situación y plantearse nuevas opciones, si fuera necesario. Este proceso es vital para poder afrontar el viaje con la seguridad suficiente de que va a ser posible completarlo. Si la etapa no se puede proseguir por el camino decidido de antemano, entonces habrá que tomar un desvío, desandar parte del camino o asumir los riesgos que suponga el continuar.

En este capítulo se exponen las diferentes evaluaciones que se han llevado a cabo durante el desarrollo de la tesis. Por una parte, se presentan una serie de experimentos que se realizaron para determinar qué razonador OWL 2 y framework semántico utilizar. Por la otra, se realizó una evaluación del razonador de Turambar y la extensión probabilística de SPARQL-DL en la que se evalua un caso de uso de ejemplo. Este enfoque ha sido elegido debido a la metodología iterativa elegida.

6.1 Evaluación de razonadores y frameworks semánticos

Una de las primeras preguntas que había que responder durante el desarrollo de Turambar era cuál era el framework semántico y razonador más apropiado para su desarrollo. Aunque generalmente se suele utilizar el *benchmark* de la universidad de Lehigh (LUBM) (Guo *et al.*, 2005), se decidió hacer uso de una batería propia de pruebas de rendimiento. Esta decisión fue motivada por la existencia de autores que sostienen que estos test no son suficientes para la elección de un razonador(Weithöner *et al.*, 2006), ya que no ofrecen información relevante que ayude a la elección de un razonador para las aplicaciones del mundo real. Como consecuencia, se plantearon unos experimentos propios que facilitaran la elección del razonador y framework.

6.1.1 Preparación del experimento

Para este experimento seleccionamos los siguientes frameworks:

- OWL API (Bechhofer *et al.*, 2003) nace del esfuerzo de intentar ofrecer una componente reusable para la construcción de diferentes aplicaciones basadas en OWL
- Jena es un framework para el desarrollo de aplicaciones semánticas (Carroll *et al.*, 2004). Soporta tanto Resource Description Framework (RDF) (Schreiber & Raimond, 2014), RDF Schema (RDFS) (Guha & Brickley, 2014) y OWL. Además, ofrece un motor de consultas SPARQL.
- OWLlink API (Noppens *et al.*, 2010)es una implementación del protocolo OWLlink (Liebig *et al.*, 2011), que sirve para el intercambio de información entre aplicaciones que hagan uso de OWL 2. Permite que el razonador y la aplicación que hace uso de este, se encuentren en máquinas diferentes.

En cuanto a los razonadores, se hizo uso del razonador de Jena, HermiT (Shearer *et al.*, 2008) y Pellet (Sirin *et al.*, 2007). Las elecciones de los razonadores

vienen motivadas por ser software de código abierto, así como por la posibilidad de poderse integrar junto con los frameworks anteriores.

En el experimento se registraban las actividades diarias que realiza una persona. Estas actividades se extraen del *data set* de Huynh *et al.* (2008), al cual se le ha añadido la actividad de tomar medicina. A partir de estas actividades se puede inferir si el usuario ha tomado las medicinas o si se han tomado cuando no se debía. Las reglas son las encargadas de recordar al usuario que debe de comer y tomar las medicinas. Las reglas fueron implementadas tanto en SWRL (Horrocks *et al.*, 2004), código Java, como en reglas Jena. Este experimento es descrito en (Ausín *et al.*, 2012a; Ausín & Castanedo, 2012).

El experimento fue ejecutado en un portátil Acer Aspire 4830TG con las siguientes características:

- Procesador: Intel Core i5-2410M.
- RAM disponible: 8 GB.
- Sistema operativo: Ubuntu 11.10 para 64 bits.
- Versión de Java: 1.6.0_23 de OpenJDK Runtime Environment (IcedTea6 1.11pre)

El programa carga las ontologías en el razonador y a continuación empieza a actualizar la ontología con la información suministrada por el *data set*. La tabla 6.1 resumen las combinaciones de frameworks y razonadores que se llevaron a cabo durante el experimento.

	OWLlink API 1.2.2	*OWL API 3.1.0*	*Jena 2.6.4*
Jena 2.6.4	-	-	(1) Reglas Jena
HermiT 1.3.5	(2) Reglas Java	(3) Reglas Java	-
Pellet 2.3.0	(4) Reglas Java rules	(5) Reglas SWRL	(6) Jena y (7) SWRL

Cuadro 6.1: Diferentes combinaciones de razonadores y frameworks semánticos utilizados para simular la toma de medicinas de un usuario.

La ontología cargada importa una ontología llamada ”talismán extension”que importa a su vez la ontología ”talismán core”. Según los resultados devueltos por OWL API, se cargan 123 axiomas pertenecientes al ABox; 305, al TBox; y 2, al RBox. De entre las clases definidas en esta ontología, las más importantes son las siguientes:

- *DisabledPerson* identifica a las personas que necesitan de cuidados.
- *MedicalTreatment* representa un tratamiento médico.
- Las subclases de la clase *Activity* describen las posibles actividades que el usuario puede realizar. Las principales subclases se pueden ver en las figuras 6.1, 6.2 y 6.3.

Para relacionar un miembro de la clase *DisabledPerson* con un miembro de la clase *MedicalTreatment*, es decir expresar que un usuario toma una medicación, se hace uso de la propiedad *medicalTreatment*. Por otra parte, para expresar que un usuario está realizando una actividad se hace uso de la propiedad *isDoing*. En cuanto al número de reglas, existen 16 reglas SWRL y otras 16 reglas Jena, además de su traducción a Java.

Durante la ejecución del experimento la base de conocimiento se irá actualizando con la información de las actividades proporcionadas por el *data set* lo que se traduce en múltiples actualizaciones de la misma, que implican volver a ejecutar las reglas.

6.1.2 Resultados del experimento

El experimento se ejecutó diez veces para determinar el tiempo medio de ejecución por cada combinación de razonador y framework. Otras tantas fueron necesarias para determinar el consumo de memoria. Se midieron la memoria entregada (*committed memory*) y la memoria usada. La memoria entregada es la memoria que se garantiza estar disponible para ser usada por la máquina virtual de Java. Por su parte, la memoria usada es la memoria ocupada por los objetos tanto referenciados como desreferenciados. Los resultados son los mostrados por las figuras 6.4, 6.5 y 6.6.

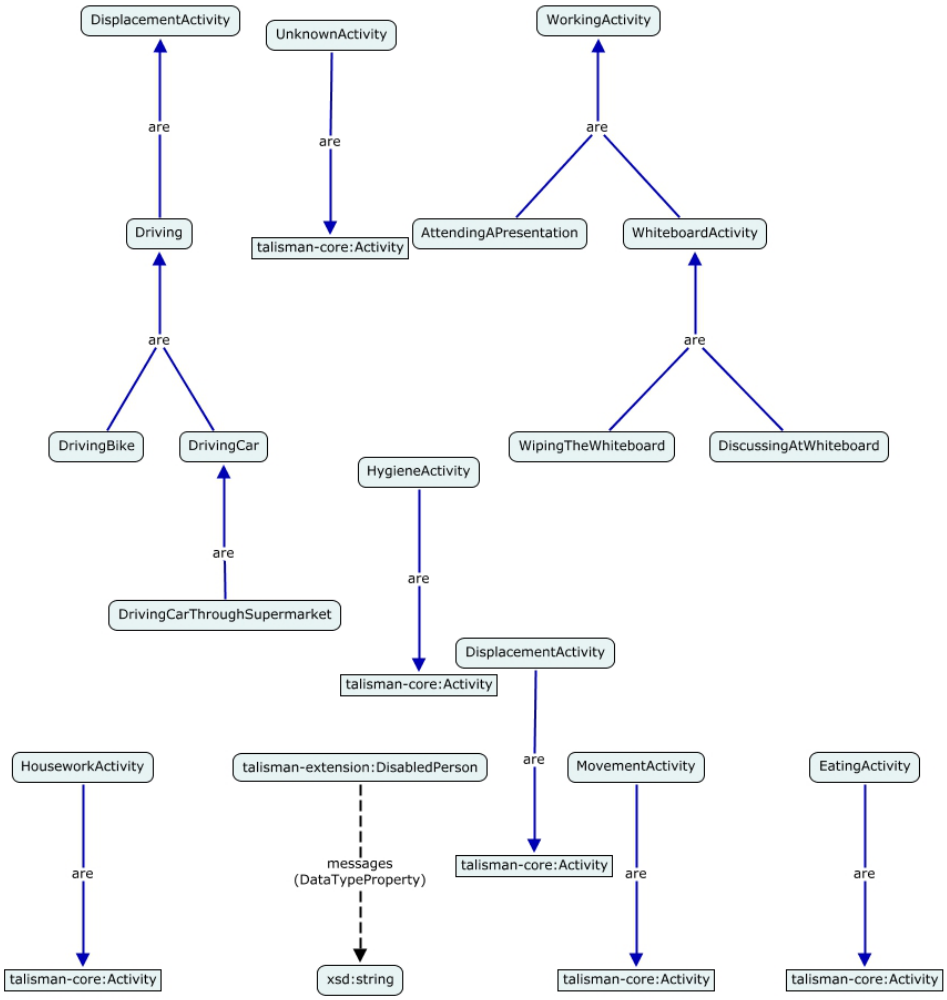


Figura 6.1: Parte de las principales subclases de *Activity* definidas en la ontología utilizada para determinar cuándo un usuario debe de tomar la medicación (1 de 3)

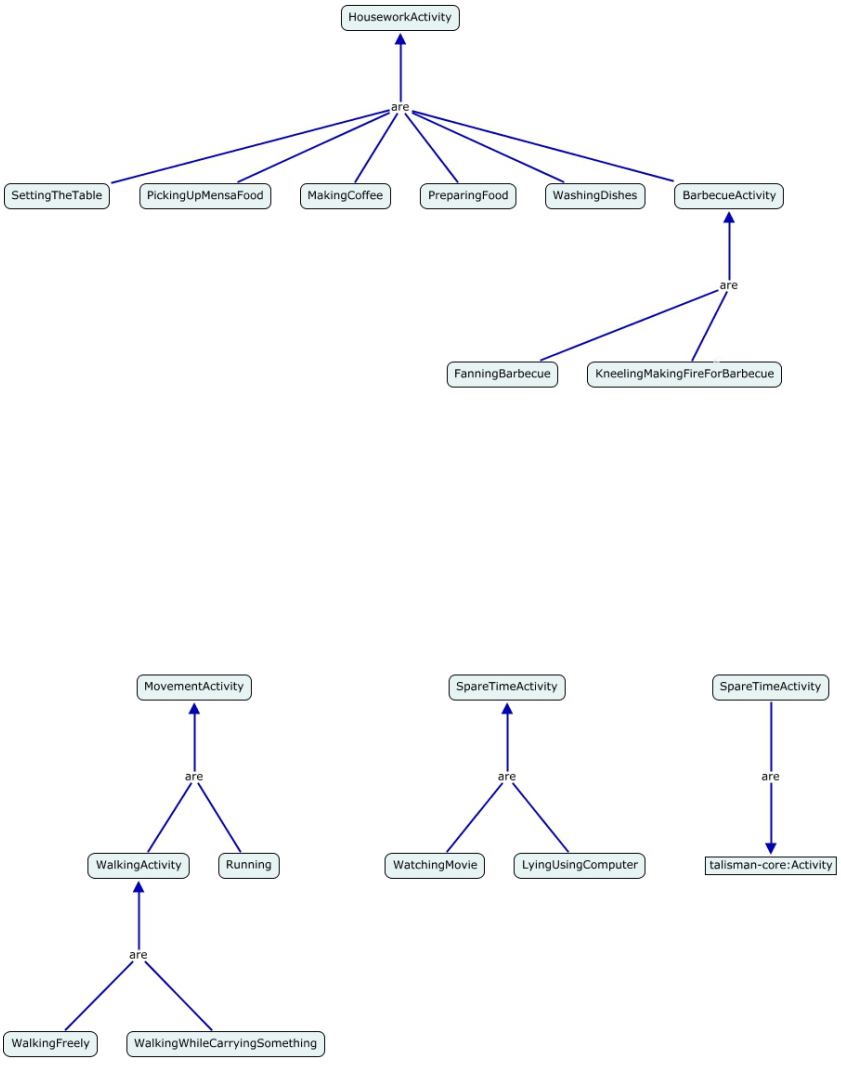


Figura 6.2: Parte de las principales subclases de *Activity* definidas en la ontología utilizada para determinar cuándo un usuario debe de tomar la medicación (2 de 3)

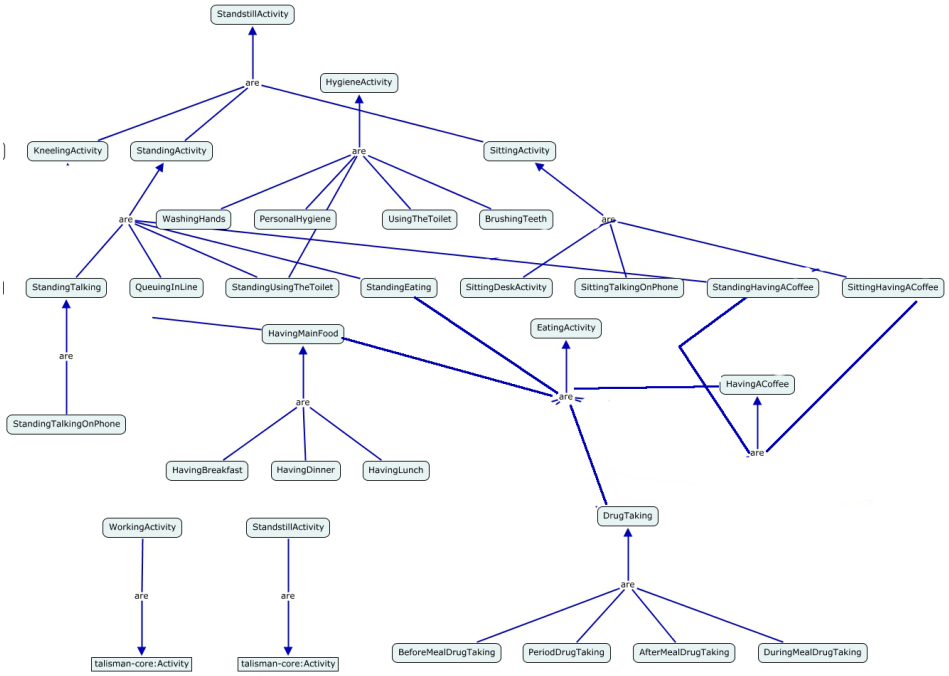


Figura 6.3: Parte de las principales subclases de *Activity* definidas en la ontología utilizada para determinar cuándo un usuario debe de tomar la medicación (3 de 3)

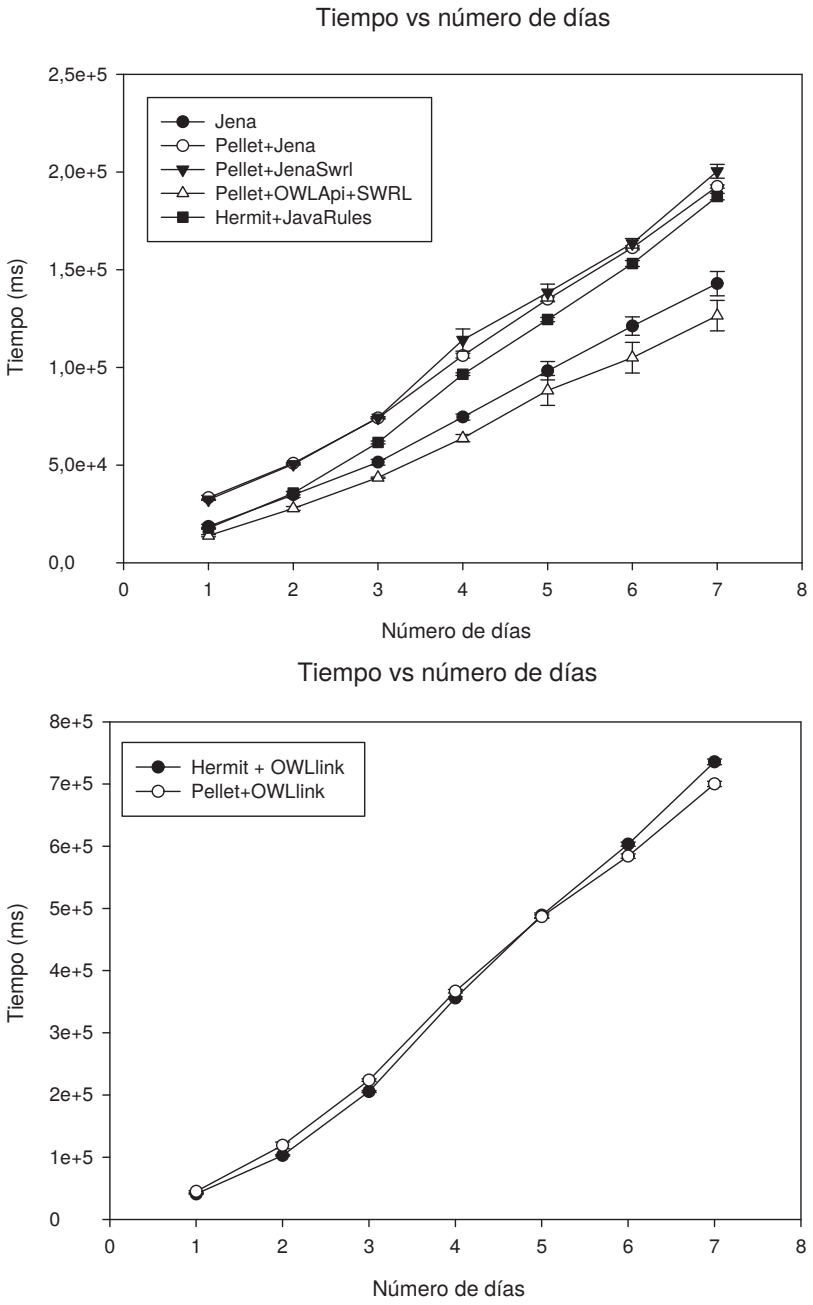


Figura 6.4: Tiempo requerido para procesar las actividadees del experimento de las medicinas a lo largo de siete días. El gráfico superior muestra los resultados de ejecutar los razonadores como librerías y el inferior el de ejecutarlos con OWLlink API en modo cliente/servidor.

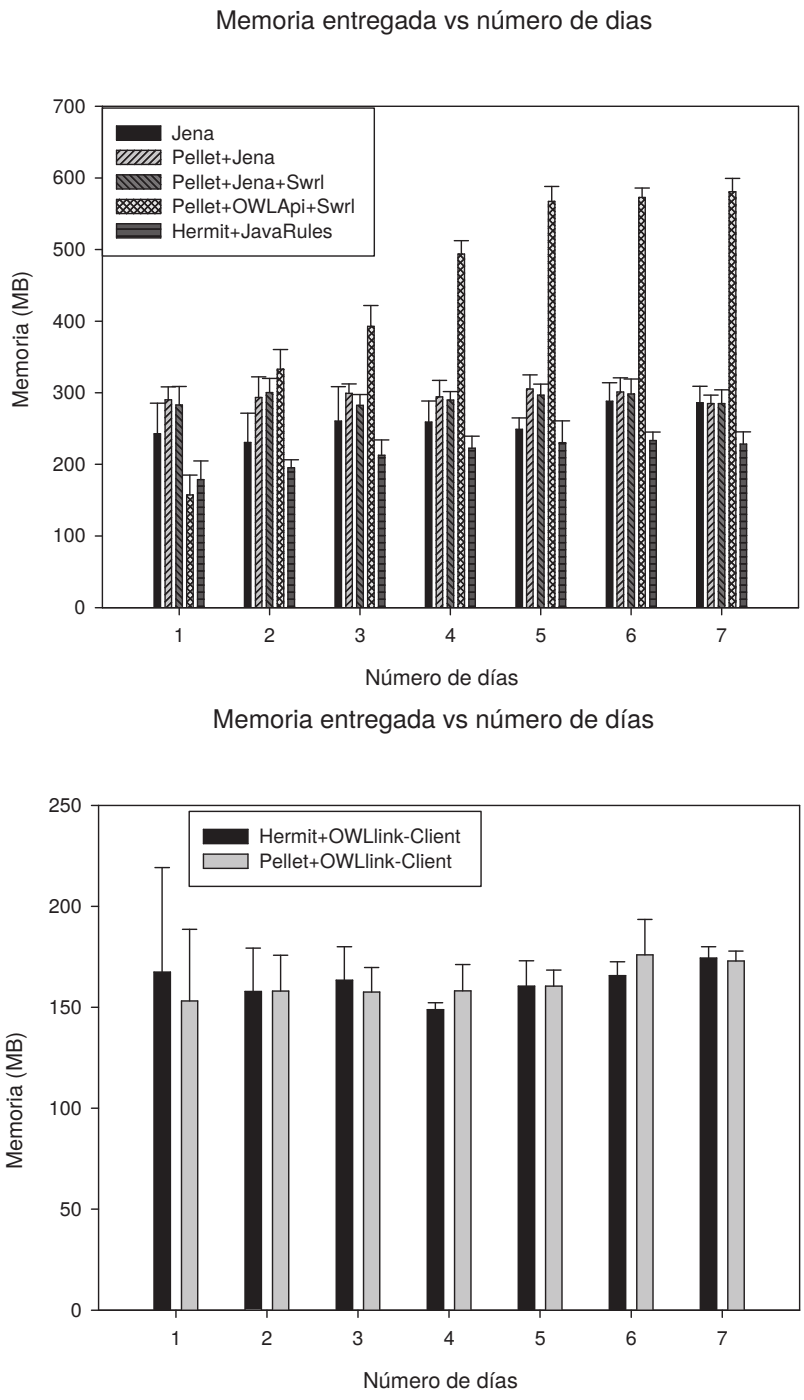


Figura 6.5: Resultados de la memoria entregada (*committed memory*) cuando se incrementa el número de días (el volumen de datos crece) La imagen superior muestra la memoria utilizada por el uso de los razonadores como librerías y la inferior, los resultados de la parte cliente de las combinaciones con OWLlink API

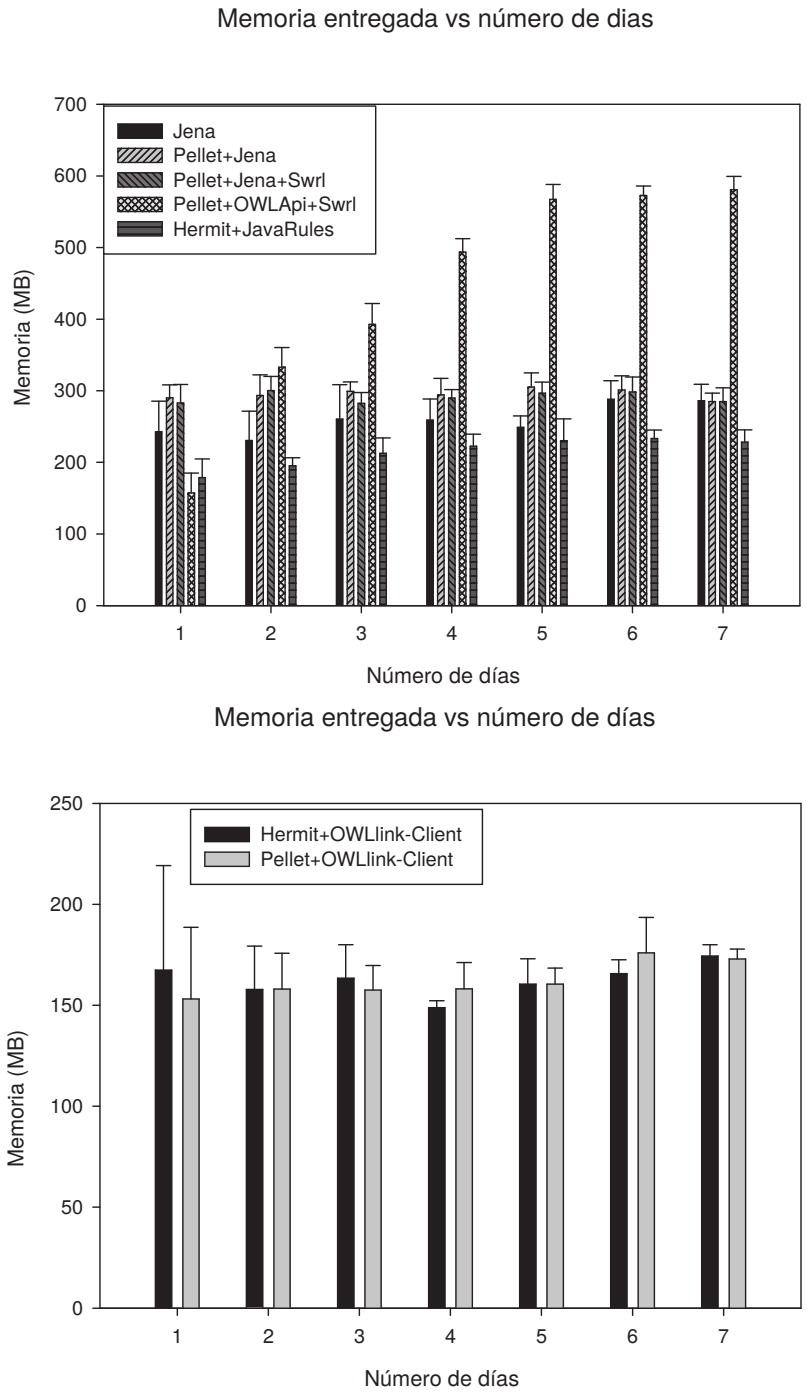


Figura 6.6: Resultados de la memoria usada cuando se incrementa el número de días (el volumen de datos crece). La imagen superior muestra la memoria utilizada por el uso de los razonadores como librerías y la inferior, los resultados de la parte cliente de las combinaciones con OWLlink API

La combinación de framework y razonador más rápido es la de OWL API con el razonador Pellet. Esta configuración resultó hasta un 12 % más rápida que la segunda configuración más rápida, como se puede apreciar en la figura 6.4. Por contra, también era la que más memoria requería, ver figuras 6.5 y 6.6 . Otra conclusión que se puede extraer es que en general el uso del framework Jena con otro razonador que no fuese Jena, es más lento que cualquier otra combinación. La razón de que Jena con su propio razonador sea más rápido que con otros razonadores puede deberse a que no hace un razonamiento de OWL completo. En cuanto a la memoria, HermiT combinado con OWL API es la combinación que menos memoria requiere sin hacer uso de OWLlink.

En cuanto a las combinaciones con el protocolo OWLlink, se puede apreciar un alto incremento en el tiempo de ejecución, seguramente debido al uso de una nueva capa de abstracción. En cuanto al uso de memoria en la parte cliente, esta es parecida con cualquiera de las combinaciones, tal como se esperaba.

6.1.3 Conclusiones del experimento

A partir de los resultados obtenidos se decidió hacer uso de OWL API como framework semántico, ya que la combinación de OWL API y Pellet resultó la más rápida; y HermiT con OWL API, la que menos memoria necesitaba. El hacer uso de OWL API permite utilizar de manera casi transparente cualquier razonador compatible con esta API. Por ello, en Turambar se eligió hacer uso de ella, ya que permite hacer uso tanto de HermiT, para aquellos casos en los que la memoria fuese un requisito importante; como de Pellet, cuando el tiempo de ejecución fuese el requisito más importante.

El uso de OWLLink como protocolo fue rechazado debido a que el tiempo de ejecución era más del doble que en aquellos casos en los que se usaba OWL API y Jena. Este hecho no compensaba la ventaja que pueda aportar un enfoque cliente/servidor. Además su uso significaba tener que extender el protocolo para poder soportar las consultas probabilística, lo que requiere un esfuerzo adicional.

6.2 Experimento de Turambar

Con el objetivo de evaluar Turambar se plantea un caso de uso en el que se pretende determinar la posición de un usuario, John. Dicha información podrá ser consultada por los familiares o cuidadores del usuario mediante el uso de una aplicación. La casa modelada en el ejemplo cuenta con dos dormitorios, un baño, un salón y una cocina. En el caso de este escenario, se pretende evaluar el tiempo de respuesta del sistema cuando la información del sensor ”vestible”de localización falla o no está disponible.

Los sensores ”vestibles” utilizados en entornos inteligentes suelen depender de baterías para su funcionamiento y de una conexión inalámbrica para comunicarse con el resto del sistema. Esto conlleva que la información del sensor pueda no estar disponible debido a problemas de conectividad entre el dispositivo y el sistema central o a que este se haya quedado sin batería. Otro riesgo que se corre con estos sistemas es que el usuario se los quite por alguna razón y se le olvide ponérselos de nuevo. Todos estos inconvenientes deben de ser tenidos en cuenta por un sistema de monitorización.

Un ejemplo de los problemas anteriormente descritos son los sensores de localización. Imagínese que John se olvidase de recargar el sensor localización o que no se lo pusiese. Este hecho provocaría que no fuese posible determinar su posición. En sistemas tradicionales se podría asumir que la activación de ciertos sensores por parte de John están vinculados con su localización. Por ejemplo, se podría definir que si la televisión está activada, John debería de estar en el salón viéndola. También se podría establecer que si se ha activado la vitrocerámica, John debería de estar en la cocina. Con el objetivo de facilitar el modelado se podría optar por combinar OWL con reglas SWRL. Por ejemplo, para describir que un usuario está en el salón y no en la cocina cuando la televisión está encendida y la vitrocerámica,

apagada se podría utilizar una regla tal que:

$$Usuario(?usuario) \wedge tieneVitroceramica(?usuario, ?vitro) \wedge$$
$$tieneTelevision(?usuario, ?tele) \wedge estaEncendida(?tele, ?teleOn)$$
$$\wedge estaEncendida(?vitro, ?vitroOn) \wedge swrlb : equal(?teleOn, true)$$
$$\wedge swrlb : equal(?vitroOn, false) \wedge estaEn(?tele, ?telePos)$$
$$\longrightarrow estaEn(?usuario, ?telePos)$$

El problema es que ambos sensores pueden estar activados al mismo tiempo como son la vitrocerámica o la televisión. En estos casos, el desafío que se plantea es cómo modelar los hábitos del usuario de una manera flexible y que pueda ser definida tanto por expertos como sistemas de aprendizaje.

Por otra parte, se asume que esto va a ser siempre cierto cuando no existe una evidencia de que esto pueda ser siempre así. Por ejemplo, podría ser que la televisión estuviese encendida y la vitrocerámica, apagada y el usuario no estuviese en la cocina. Por lo que verdaderamente se estaría diciendo que algo es probable que suceda si se cumplen una serie de hechos, ya que se carece de certeza debido a la falta de información concluyente. Es claro por tanto que las ontologías no son suficientes para estos casos, debido a la incertidumbre presente y a que no permite representar la noción de que un hecho sea probable. Además, este tipo de información es claramente probabilística. Las relaciones entre las activaciones de los sensores, su duración o cualquier otra información relevante para determinar la posición de una persona siempre darán como resultado una estimación de los lugares en los que el usuario puede estar y la probabilidad de que esta sea cierta.

Debido a todas las razones anteriormente expuestas, modelar este conocimiento en Turambar es una opción más acertada, que hacerlo simplemente con ontologías. Por ejemplo, se podría modelar la localización del usuario tal como muestra la figura 6.7. Esta red bayesina es la utilizada para evaluar los tiempos de consulta, consistencia, carga y tiempo de inserciones. Tanto las probabilidades asignadas y las relaciones entre nodos se han definido de manera no empírica, debido a que el interés del experimento no está en detectar la posición del usuario, sino evaluar el comportamiento de Turambar. No obstante, la red podría ser construida por un experto o mediante el uso de sistemas de aprendizaje.

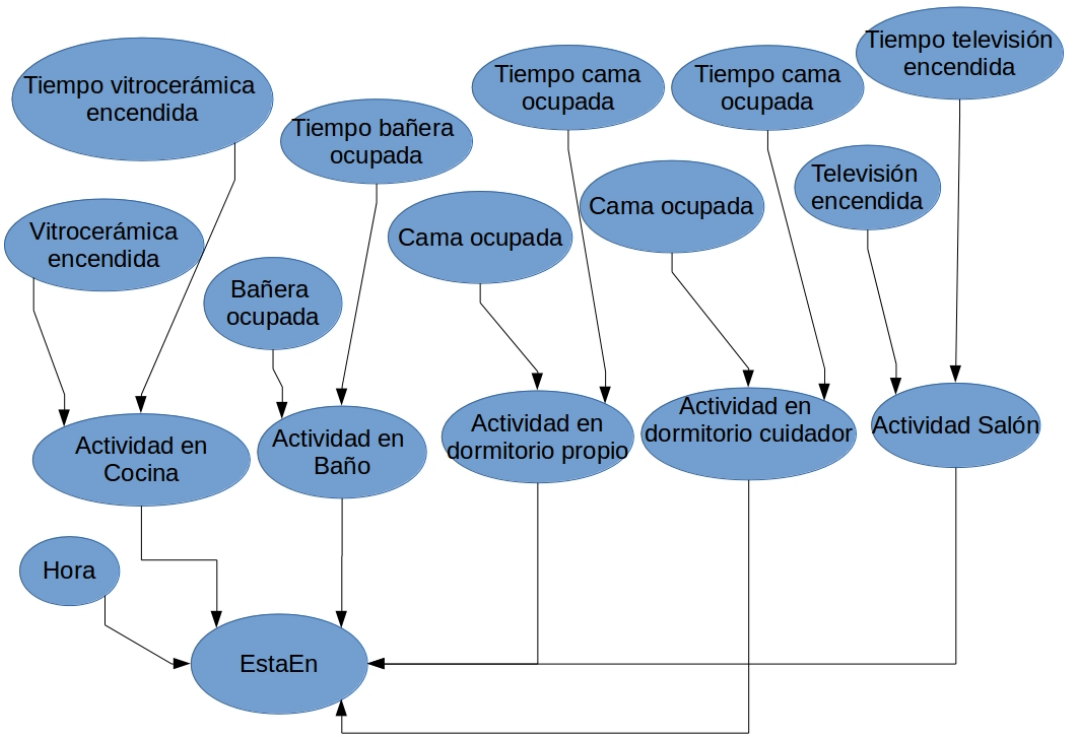


Figura 6.7: Red bayesiana de ejemplo modelada en Turambar para estimar dónde se encuentra un usuario.

6.2.1 Definición de la ontología de Turambar

A modo de ilustración se va a explicar brevemente cómo se han modelado algunas de las relaciones entre los nodos de la red bayesiana y las propiedad y clases de OWL.

Para modelar la ontología de dependencias lo primero es asociar la ontología OWL 2 DL con ésta. Para ello hay que importar la ontología de anotaciones de Turambar:

```
Import(<http://www.morelab.deusto.es/talisman/
    turambarAnnotations.owl>)
```

A continuación definir la localización de la ontología de dependencias mediante la anotación de la ontología:

```
Annotation(<http://www.morelab.deusto.es/talisman/
    turambarAnnotations.owl#turambarOntology> "http://
    www.morelab.deusto.es/talisman/evaluationSupport.owl
    ")
```

A continuación, se deben de anotar las propiedades y clases relacionadas con un nodo de la ontología de dependencias. Por ejemplo, el nodo de la imagen definido como "actividad en cocina" es el individuo `http://www.morelab.deusto.es/talisman/evaluationSupport.owl#KitchenActivity` en la ontología de dependencias y está asociado a la clase *:KitchenActivity*.

```
AnnotationAssertion(<http://www.morelab.deusto.es/
    talisman/turambarAnnotations.owl\#turambarClass> :
    KitchenActivity "http://www.morelab.deusto.es/
    talisman/evaluationSupport.owl\#KitchenActivity")
```

En el caso de la propiedad "vitrocerámica encendida" está representado por el individuo "`http://www.morelab.deusto.es/talisman/evaluationSupport.owl#CooktopisSwitched`" en la ontología de dependencias, el cual a su vez está emparejado con la propiedad *:isSwitched* en la ontología OWL 2.

```
AnnotationAssertion(<http://www.morelab.deusto.es/
    talisman/turambarAnnotations.owl#turambarProperty> :
    isSwitched "http://www.morelab.deusto.es/talisman/
    evaluationSupport.owl#CooktopisSwitched")
```

En este ejemplo, dicha propiedad es utilizada para definir si la televisión está encendida; por lo que también estará emparejada con el individuo que represente al nodo ”televisión encendida”. Es decir, por cada nodo con el que se empareja una propiedad o clase, este deberá de ser anotado nuevamente indicando el nodo con el que se le empareja.

Una vez emparejadas las clases y propiedades con los individuos que representan los nodos de la red, entonces se comienza con la definición de la red en la ontología de dependencias. Las relaciones de dependencias entre nodos se declaran con la propiedad *hasParent*. Por ejemplo, para definir que el individuo *:KitchenActivity* depende del individuo *isIn*.

```
ObjectPropertyAssertion(<http://www.morelab.deusto.es/
    talisman/turambar.owl#hasParent> <http://www.morelab
    .deusto.es/talisman/evaluationSupport.owl#isIn> <
    http://www.morelab.deusto.es/talisman/
    evaluationSupport.owl#KitchenActivity>)
```

A continuación hay que definir los estados de cada nodo. Para ello, hay que relacionar mediante *hasState* cada nodo con un conjunto de individuos de la clase *State* y asociar a estos individuos con la función correspondiente con la propiedad *stateCondition*. Por ejemplo, el nodo *:KitchenActivity* tiene dos estados: uno identificando que el individuo pertenece a esa clase y otro que no pertenece a una clase disjunta.

```
ObjectPropertyAssertion(<http://www.morelab.deusto.es/
    talisman/turambar.owl#hasState> <http://www.morelab.
    deusto.es/talisman/evaluationSupport.owl#
    KitchenActivity> <http://www.morelab.deusto.es/
    talisman/evaluationSupport.owl#ReadState30>)
ObjectPropertyAssertion(<http://www.morelab.deusto.es/
    talisman/turambar.owl#hasState> <http://www.morelab.
    deusto.es/talisman/evaluationSupport.owl#
    KitchenActivity> <http://www.morelab.deusto.es/
    talisman/evaluationSupport.owl#ReadState31>)

ClassAssertion(<http://www.morelab.deusto.es/talisman/
    turambar.owl#State> <http://www.morelab.deusto.es/
    talisman/evaluationSupport.owl#ReadState30>)
```

```
DataPropertyAssertion(<http://www.morelab.deusto.es/
    talisman/turambar.owl#stateCondition>
 <http://www.morelab.deusto.es/talisman/
    evaluationSupport.owl#ReadState30> "http://www.
    semanticweb.org/david/ontologies/2014/5/untitled-
    ontology-62#KitchenActivity"^^xsd:string)

 ClassAssertion(<http://www.morelab.deusto.es/talisman
    /turambar.owl#State> <http://www.morelab.deusto.es/
    talisman/evaluationSupport.owl#ReadState31>)
 DataPropertyAssertion(<http://www.morelab.deusto.es/
    talisman/turambar.owl#stateCondition> <http://www.
    morelab.deusto.es/talisman/evaluationSupport.owl#
    ReadState31> "!http://www.semanticweb.org/david/
    ontologies/2014/5/untitled-ontology-62#
    KitchenActivity"^^xsd:string)
```

En este experimento se hace uso de funciones en los estados que no pertenecen al núcleo de Turambar por lo que hay que definir la localización de la librería que las provee. Estas funciones son usadas en el nodo *isIn*. El siguiente código muestra cómo declarar la dependencia de librerías externas y cómo hace uso de sus funciones un estado del nodo *isIn*.

```
ClassAssertion(<http://www.morelab.deusto.es/talisman/
    turambar.owl#Plugin> <http://www.morelab.deusto.es/
    talisman/evaluationSupport.owl#Plugin>)
DataPropertyAssertion(<http://www.morelab.deusto.es/
    talisman/turambar.owl#pluginUrl> <http://www.morelab
    .deusto.es/talisman/evaluationSupport.owl#Plugin> "
    file:resources/evaluation/Evaluation.jar"^^xsd:
    string)

DataPropertyAssertion(<http://www.morelab.deusto.es/
    talisman/turambar.owl#stateCondition> <http://www.
    morelab.deusto.es/talisman/evaluationSupport.owl#
    ReadState34> "get(http://www.semanticweb.org/david/
    ontologies/2014/5/untitled-ontology-62#LivingRoom)
    "^^xsd:string)
```

La función usada es la función *get* que retorna la habitación del tipo *LivingRoom* para un usuario.

Una vez definidos los estados hay que definir si el nodo es auxiliar o no. Como el nodo *:KitchenActivity* no es auxiliar, entonces se describiría tal que:

```
DataPropertyAssertion(<http://www.morelab.deusto.es/
    talisman/turambar.owl#isAuxiliar> <http://www.
    morelab.deusto.es/talisman/evaluationSupport.owl#
    KitchenActivity> "false"^^xsd:boolean)
```

El siguiente paso es definir el contexto que comparten los nodos de la red y relacionarles con la variable de contexto correspondiente. En el caso del nodo *:KitchenActivity* la variable es *activity* y se define tal que:

```
ClassAssertion(<http://www.morelab.deusto.es/talisman/
    turambar.owl#Context> <http://www.morelab.deusto.es/
    talisman/evaluationSupport.owl#Base>)
DataPropertyAssertion(<http://www.morelab.deusto.es/
    talisman/turambar.owl#relationship> <http://www.
    morelab.deusto.es/talisman/evaluationSupport.owl#
    Base> "PREFIX t: <http://www.semanticweb.org/david/
    ontologies/2014/5/untitled-ontology-62#> SELECT ?
    activity ?user ?cooktop ?bath ?bedcaregiver ?bed ?
    tv ?activity WHERE { Type(?user, t:User),
    PropertyValue(?user, t:isDoing, ?activity),
    PropertyValue(?user, t:isOwner, ?kitchen), Type(?
    kitchen, t:Kitchen), PropertyValue(?kitchen, t:
    hasDevice, ?cooktop), Type(?cooktop, t:Cooktop),
    PropertyValue(?user, t:isOwner, ?bathroom), Type(?
    bathroom, t:Bathroom),PropertyValue(?bathroom, t:
    hasDevice, ?bath), Type(?bath, t:Bath),PropertyValue
    (?user, t:isOwner, ?guestroom), Type(?guestroom, t:
    GuestRoom),PropertyValue(?guestroom, t:hasFurniture,
     ?bedcaregiver), Type(?bedcaregiver, t:Bed),
    PropertyValue(?user, t:isOwner, ?livingroom), Type(?
    livingroom, t:LivingRoom),PropertyValue(?livingroom,
     t:hasDevice, ?tv), Type(?tv, t:Television),
    PropertyValue(?user, t:isOwner, ?bedroom), Type(?
    bedroom, t:Bedroom), PropertyValue(?bedroom, t:
    hasFurniture, ?bed), Type(?bed, t:Bed)}
"^^xsd:string)

ObjectPropertyAssertion(<http://www.morelab.deusto.es/
    talisman/turambar.owl#hasContext> <http://www.
```

```
    morelab.deusto.es/talisman/evaluationSupport.owl#
    KitchenActivity> <http://www.morelab.deusto.es/
    talisman/evaluationSupport.owl#Base>)

ClassAssertion(<http://www.morelab.deusto.es/talisman/
    turambar.owl#Variable> <http://www.morelab.deusto.es
    /talisman/evaluationSupport.owl#activity>)
DataPropertyAssertion(<http://www.morelab.deusto.es/
    talisman/turambar.owl#variableName> <http://www.
    morelab.deusto.es/talisman/evaluationSupport.owl#
    activity> "activity"^^xsd:string)

ObjectPropertyAssertion(<http://www.morelab.deusto.es/
    talisman/turambar.owl#hasVariable> <http://www.
    morelab.deusto.es/talisman/evaluationSupport.owl#
    KitchenActivity> <http://www.morelab.deusto.es/
    talisman/evaluationSupport.owl#activity>)
```

Posteriormente, se asocian los nodos con sus tablas de probabilidad condicionada mediante la propiedad *hasProbabilityDistribution*. Para el nodo *KitchenActivity* se haría de la siguiente forma:

```
ClassAssertion(<http://www.morelab.deusto.es/talisman/
    turambar.owl#ProbabilityDistribution> <http://www.
    morelab.deusto.es/talisman/evaluationSupport.owl#
    KitchenActivityDistribution>)

ObjectPropertyAssertion(<http://www.morelab.deusto.es/
    talisman/turambar.owl#hasProbabilityDistribution> <
    http://www.morelab.deusto.es/talisman/
    evaluationSupport.owl#KitchenActivity> <http://www.
    morelab.deusto.es/talisman/evaluationSupport.owl#
    KitchenActivityDistribution>)
```

Finalmente, queda asociar cada celda de la tabla de probabilidad condicionada con esta mediante la propiedad de objeto *hasProbability*. Las celdas se definen mediante la clase *Probability* y los estados de los nodos de los que dependen se asocian mediante *hasCondition*; el estado propio al que hace referencia la celda, vía la propiedad de objeto *hasValue*; y la probabilidad, mediante probabilidad de dato *probabilityValue*. Por ejemplo, una celda de la tabla de probabilidad condicionada

de *KitchenActivity* se definiría tal que:

```
ClassAssertion(<http://www.morelab.deusto.es/talisman/
    turambar.owl#Probability> <http://www.morelab.deusto
    .es/talisman/evaluationSupport.owl#
    KitchenActivityProbability6>)
ObjectPropertyAssertion(<http://www.morelab.deusto.es/
    talisman/turambar.owl#hasProbability> <http://www.
    morelab.deusto.es/talisman/evaluationSupport.owl#
    KitchenActivityDistribution> <http://www.morelab.
    deusto.es/talisman/evaluationSupport.owl#
    KitchenActivityProbability6>)

ObjectPropertyAssertion(<http://www.morelab.deusto.es/
    talisman/turambar.owl#hasCondition> <http://www.
    morelab.deusto.es/talisman/evaluationSupport.owl#
    KitchenActivityProbability6> <http://www.morelab.
    deusto.es/talisman/evaluationSupport.owl#ReadState26
    >)
ObjectPropertyAssertion(<http://www.morelab.deusto.es/
    talisman/turambar.owl#hasCondition> <http://www.
    morelab.deusto.es/talisman/evaluationSupport.owl#
    KitchenActivityProbability6> <http://www.morelab.
    deusto.es/talisman/evaluationSupport.owl#ReadState14
    >)

ObjectPropertyAssertion(<http://www.morelab.deusto.es/
    talisman/turambar.owl#hasValue> <http://www.morelab.
    deusto.es/talisman/evaluationSupport.owl#
    KitchenActivityProbability6> <http://www.morelab.
    deusto.es/talisman/evaluationSupport.owl#ReadState30
    >)
DataPropertyAssertion(<http://www.morelab.deusto.es/
    talisman/turambar.owl#probabilityValue> <http://www.
    morelab.deusto.es/talisman/evaluationSupport.owl#
    KitchenActivityProbability6> "0.87358785"^^xsd:float
    )
```

Para este experimento no ha sido necesario el uso de *MetaNode*s. No obstante la única diferencia hubiese radicado en que al emparejar nodos de la red con propiedades se hubiese hecho referencia a un individuo de la clase *MetaNode* y no

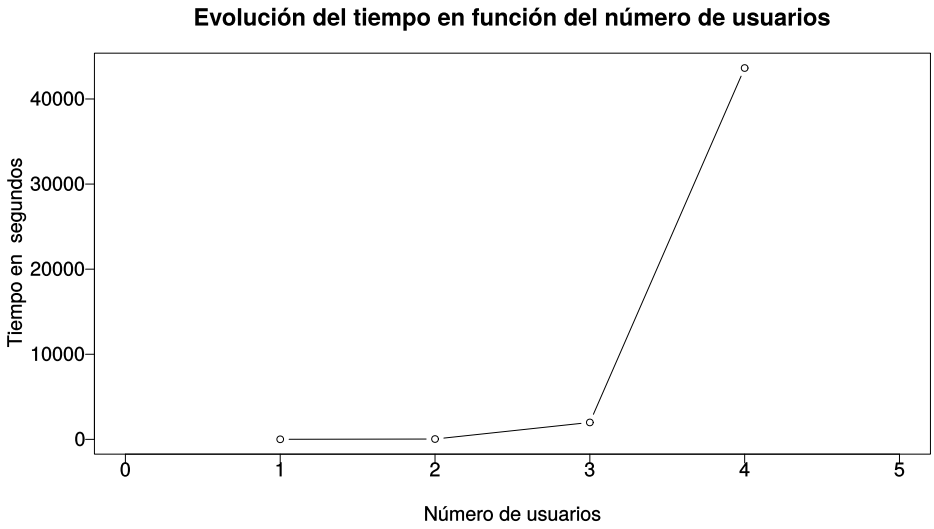


Figura 6.8: Tiempo de ejecución total a medida que se incrementan el número de usuarios.

de *Node*, que estaría relacionado finalmente con los nodos de la red mediante la propiedad *compriseNode*.

6.2.2 Ejecución del experimento

Una vez generada la ontología de Turambar, se crearon cuatro copias diferentes de ella. La diferencia entre las copias se basaba en la información del número de usuarios que contenían. Todos los usuarios poseían una casa con igual número de habitaciones, mobiliario y electrodomésticos, que representaban la información necesaria descrita por la red definida en la figura 6.7. La primera ontología contenía un único usuario; la segunda, dos; la tercera, 3; y la cuarta, cuatro. Las ontologías son cargadas en el razonador y se ejecuta la primera prueba de consistencia. Tras esto se procede a la ejecución de diez etapas. Una etapa está formada por la ejecución de una serie de cambios aleatorios en la ontología, la comprobación de la consistencia y una consulta de la extensión de SPARQL-DL propuesta para determinar dónde se encuentran los usuarios.

Los cambios producidos durante una etapa son aleatorios y consiste en cambiar el estado de alguno de los sensores implicados en la red bayesiana. El cambio de

estado puede ser a un valor conocido o la eliminación del valor para indicar que es desconocido. También se modificará el tiempo que lleva con ese valor sin cambios. Por cada usuario se realizará un único cambio en cada etapa.

La consulta realizada para determinar dónde se encuentra el usuario es la siguiente:

```
PREFIX t: <http://www.semanticweb.org/david/ontologies
    /2014/5/untitled-ontology-62#>
SELECT ?user ?place ?prob  where { ProbPropertyValue(?
    user, t:isIn, ?place, ?prob) }
```

Las ontologías utilizadas, independientemente del número de usuarios, cuentan con un TBox de 314 axioma y RBox de 18 axiomas. En cuanto al tamaño del ABox, es de 63 axiomas para un usuario, 127 axiomas para dos usuarios, de 184 axiomas para tres usuarios y de 253 axiomas para cuatro usuarios. En cuanto a la ontología de dependencias, esta está formada por un ABox de 5537 axiomas y un TBox de 43 axiomas. Nótese que el valor para las propiedades de algunos individuos puede ser desconocido, también en las ontologías iniciales, debido a que se generaron bajo las mismas premisas con las que se realizan los cambios.

El experimento se ejecutó un total de diez veces por cada ontología creada en un equipo Acer Aspire 4830TG con las siguientes características:

- Procesador: Intel Core i5-2410M.
- RAM disponible: 8 GB.
- Sistema operativo: Ubuntu 14.04 para 64 bits.
- Versión de Java: 1.7.0_51

El experimento iba a ser ejecutado haciendo uso de los razonadores HermiT y Pellet. No obstante, no fue posible realizar el experimento con Pellet debido a que presentaba problemas en la gestión de memoria que impedían su ejecución. Por ello, solo se ha ejecutado con HermiT.

La figura 6.8 muestra el tiempo total de ejecución del experimento. En ella se puede apreciar que el tiempo total se incrementa desde 8.109521 segundos de media de tiempo total para un usuario a los 43638.06 segundos cuando hay cuatro

usuarios. Estos tiempos indican que a medida que el ABox crece, el rendimiento de Turambar decrece dramáticamente, ofreciendo tiempos de etapas superiores a los 30 minutos.

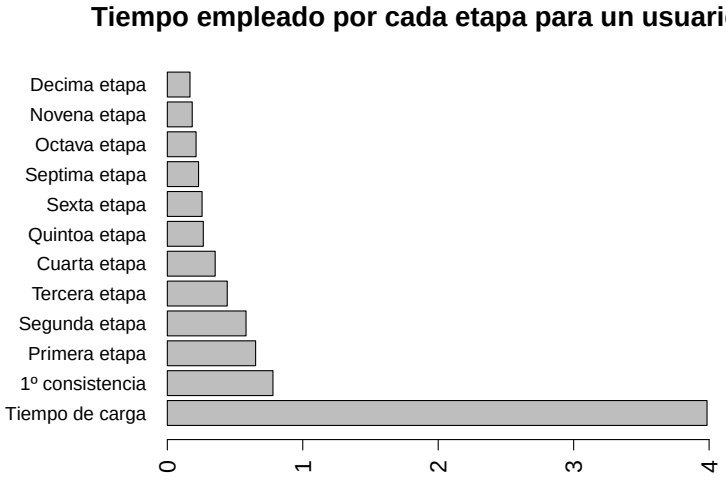
El tiempo medio de carga, ejecución de la primera consistencia y las etapas para las diferentes configuraciones de usuarios es mostrado en las imágenes 6.9, 6.10, 6.11 y 6.12. En ellas se puede apreciar que el tiempo en cada etapa es parecido y representan un porcentaje de tiempo similar respecto al tiempo final. Este resultado era esperable debido a que los cambios que se realizan en la ontología tienen implicaciones lógicas parecidas y a que el tamaño del ABox entre cambios va a ser parecido.

Una vez vistos los resultados es importante analizar dónde se producen las mayores pérdidas de tiempo para saber la razón. Por ello, se tomaron los tiempos de cada parte de una etapa por separado (cambios, consistencia y consulta). Los tiempos para los cambios se muestran en las figuras 6.13 y 6.14 ; los tiempos para las consistencias, en las figuras 6.15 y 6.16; y los tiempos de la consulta, en las figuras 6.17 y 6.18. De estos resultados se puede extraer que la mayoría del tiempo se dedica a determinar que la ontología sea consistente. Este hecho se agrava a medida que el número de usuarios se incrementa.

Estos resultados ponen en relevancia que existe un problema con el cálculo de la consistencia, por lo que se decide revisar el algoritmo para determinar dónde reside el problema. El primer paso es intentar aislar la causa del problema por lo que se decidió medir el tiempo que se necesita para responder la consulta del contexto de los nodos de la red.

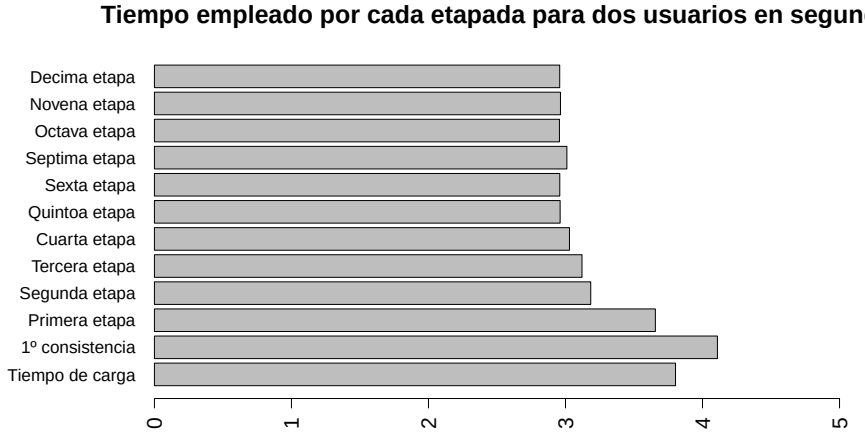
Con este objetivo, se ejecutó en un programa aparte la consulta SPARQL-DL diez veces para las ontologías de uno, dos, tres y cuatro usuarios. La media del tiempo de ejecución de la consulta de contexto, así como su comparación frente al tiempo medio de consistencia por etapa se puede ver en la tabla 6.2. Los resultados revelaron que el motor de consultas de SPARQL-DL no estaba lo suficientemente optimizado y se perdía en la consulta de contexto la práctica totalidad del tiempo.

Adicionalmente, se ejecutó el experimento cuatro veces más para medir el uso de la memoria. Los resultados pueden verse en la tabla 6.3. En principio, no parece que exista el mismo problema de memoria que en Pellet, ya que no se ha observado que en ningún momento la memoria usada, ni entregada, alcancen el máximo del



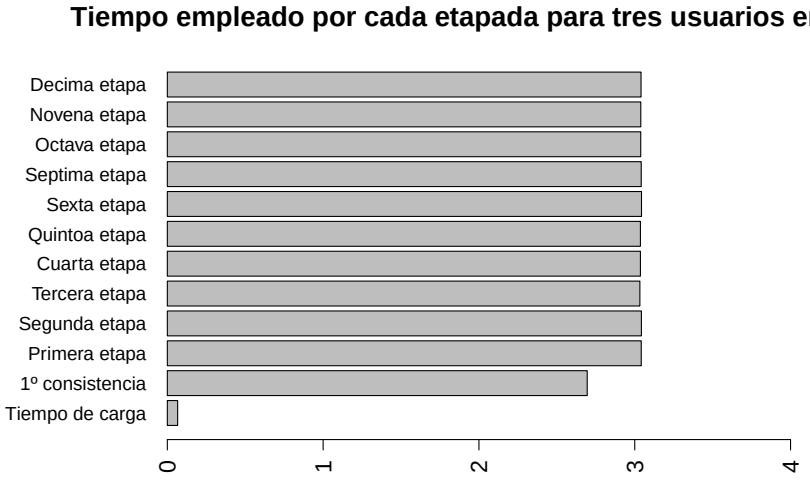
	Tiempo (sg)	*% del total*	*Error estándar*
Tiempo de carga	3.9847643	49.136863	0.127990422
Primera consistencia	0.7799358	9.617531	0.039260955
Primera etapa	0.6519345	8.039124	0.046574146
Segunda etapa	0.5810501	7.165036	0.023865119
Tercera etapa	0.4423801	5.455070	0.017451903
Cuarta etapa	0.3532610	4.356127	0.017846766
Quinta etapa	0.2655996	3.275157	0.008504996
Sexta etapa	0.2565495	3.163559	0.011861929
Séptima etapa	0.2305585	2.843060	0.014316764
Octava etapa	0.2122361	2.617122	0.010749106
Novena etapa	0.1839068	2.267789	0.005977306
Décima etapa	0.1673451	2.063563	0.005130737

Figura 6.9: La imagen superior muestra la media del tiempo empleado por etapa y la tabla inferior el porcentaje de tiempo de cada etapa respecto al tiempo total del experimento junto con la media y el error estándar. Ambas incluyen los tiempos de carga y de la primera consistencia. Los resultados son para un usuario.



	Tiempo (sg)	*% del total*	*Error estándar*
Tiempo de carga	3.802932	9.824680	0.075384752
Primera consistencia	4.108835	10.614965	0.135621964
Primera etapa	3.655884	9.444788	0.106187540
Segunda etapa	3.184271	8.226402	0.105049976
Tercera etapa	3.120500	8.061652	0.084584508
Cuarta etapa	3.029056	7.825410	0.083092616
Quinta etapa	2.960890	7.649308	0.082671006
Sexta etapa	2.958349	7.642743	0.084837997
Séptima etapa	3.010190	7.776671	0.086915486
Octava etapa	2.955820	7.636211	0.088488375
Novena etapa	2.963345	7.655651	0.082705634
Décima etapa	2.957875	7.641520	0.085345081

Figura 6.10: La imagen superior muestra la media del tiempo empleado por etapa y la tabla inferior el porcentaje de tiempo de cada etapa respecto al tiempo total del experimento junto con la media y el error estándar. Ambas incluyen los tiempos de carga y de la primera consistencia. Los resultados son para dos usuarios.



	Tiempo (sg)	*% del total*	*Error estándar*
Tiempo de carga	3.970265	0.1995856	0.127980975
Primera consistencia	161.696696	8.1285095	5.288051644
Primera etapa	182.505538	9.1745721	5.088211889
Segunda etapa	182.563657	9.1774937	5.212231295
Tercera etapa	181.980091	9.1481578	4.990590732
Cuarta etapa	182.185146	9.1584659	5.046679411
Quinta etapa	182.189208	9.1586702	5.084769577
Sexta etapa	182.603857	9.1795146	5.202160445
Séptima etapa	182.495671	9.1740761	5.118925374
Octava etapa	182.308553	9.1646696	5.134270178
Novena etapa	182.331264	9.1658113	5.052187147
Décima etapa	182.424007	9.1704735	5.256549031

Figura 6.11: La imagen superior muestra la media del tiempo empleado por etapa y la inferior el porcentaje de tiempo de cada etapa respecto al tiempo total del experimento junto con la media y el error estándar. Ambas incluyen los tiempos de carga y de la primera consistencia. Los resultados son para tres usuarios.

	Tiempo (sg)	*% del total*	*Error estándar*
Tiempo de carga	3.983739	0.009129047	0.138533288
Primera consistencia	3660.075606	8.387346889	114.184133633
Primera etapa	3995.186053	9.155278446	100.498268463
Segunda etapa	4005.093586	9.177982327	95.927361288
Tercera etapa	3998.365005	9.162563261	98.643663250
Cuarta etapa	3993.940377	9.152423883	101.105890125
Quinta etapa	3999.572874	9.165331187	99.136430393
Sexta etapa	4004.726825	9.177141866	98.119821866
Séptima etapa	3997.363724	9.160268749	98.197505161
Octava etapa	3984.949640	9.131820913	100.810601524
Novena etapa	3999.151822	9.164366315	98.001911429
Décima etapa	3995.652400	9.156347117	99.095734712

Figura 6.12: La imagen superior muestra la media del tiempo empleado por etapa y la tabla inferior el porcentaje de tiempo de cada etapa respecto al tiempo total del experimento junto con la media y el error estándar. Ambas incluyen los tiempos de carga y de la primera consistencia. Los resultados son para cuatro usuarios.

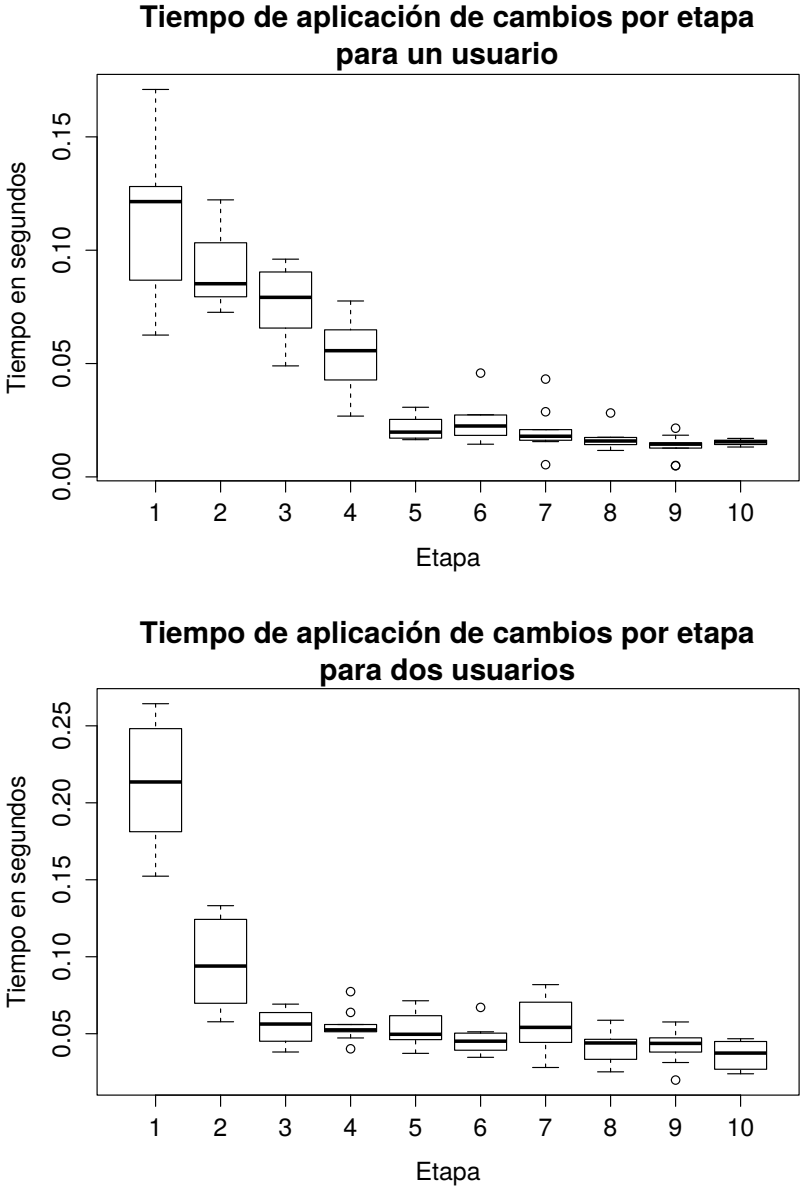


Figura 6.13: Tiempo que transcurre realizando cambios en la ontología para cada etapa cuando hay uno y dos usuarios.

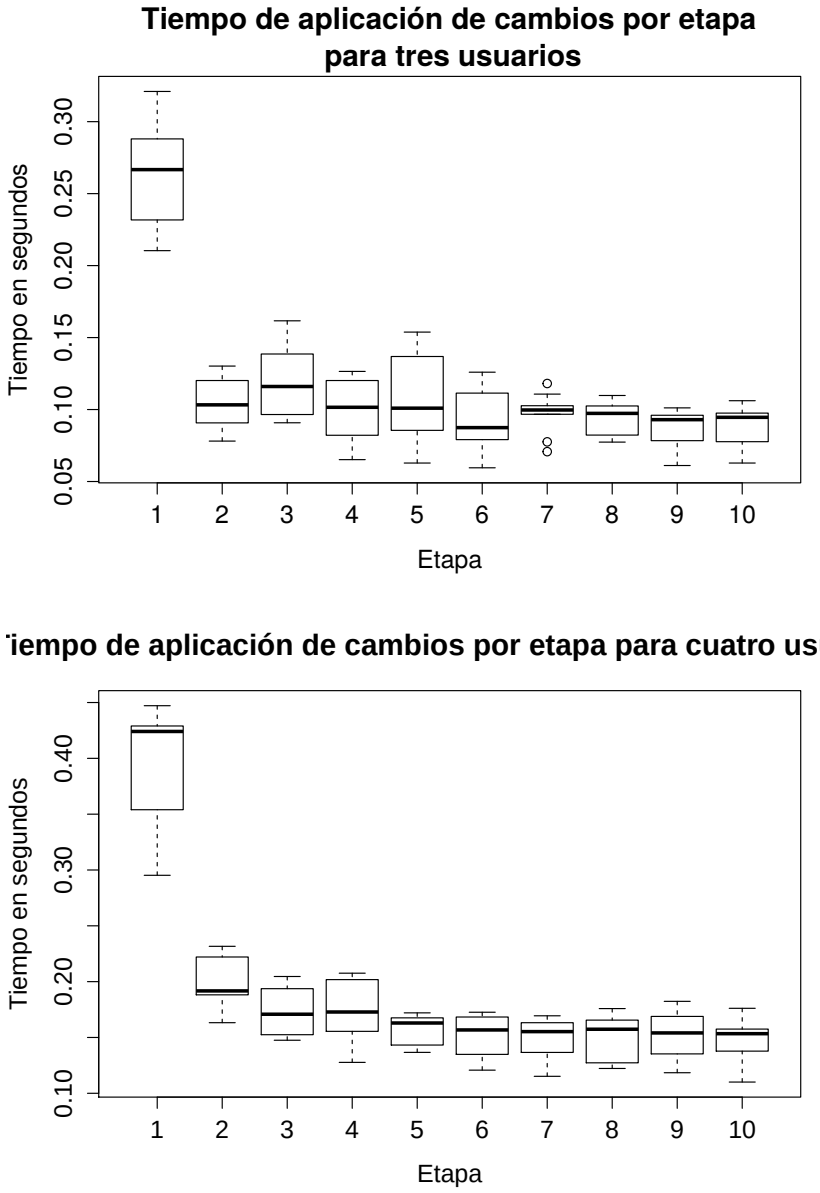


Figura 6.14: Tiempo que transcurre realizando cambios en la ontología para cada etapa cuando hay tres y cuatro usuarios.

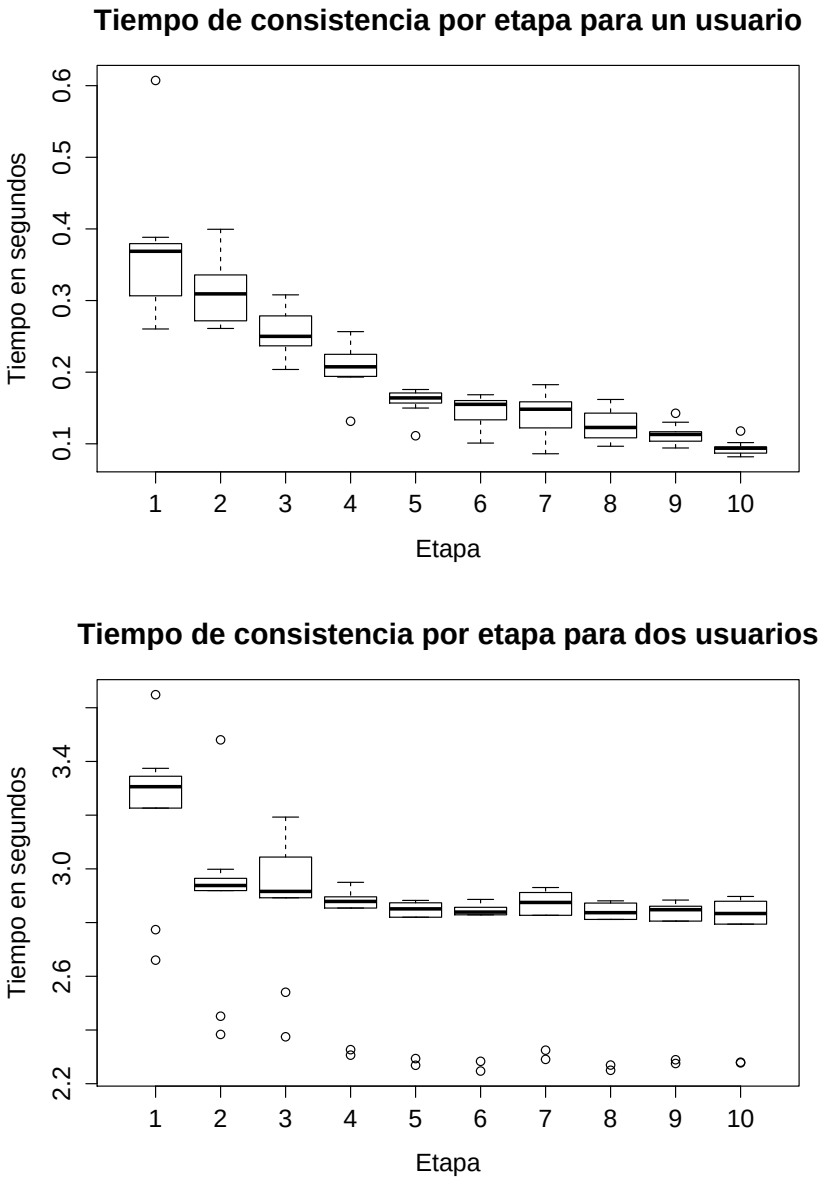


Figura 6.15: Tiempo transcurrido para determinar la consistencia de la ontología en cada etapa para uno y dos usuarios.

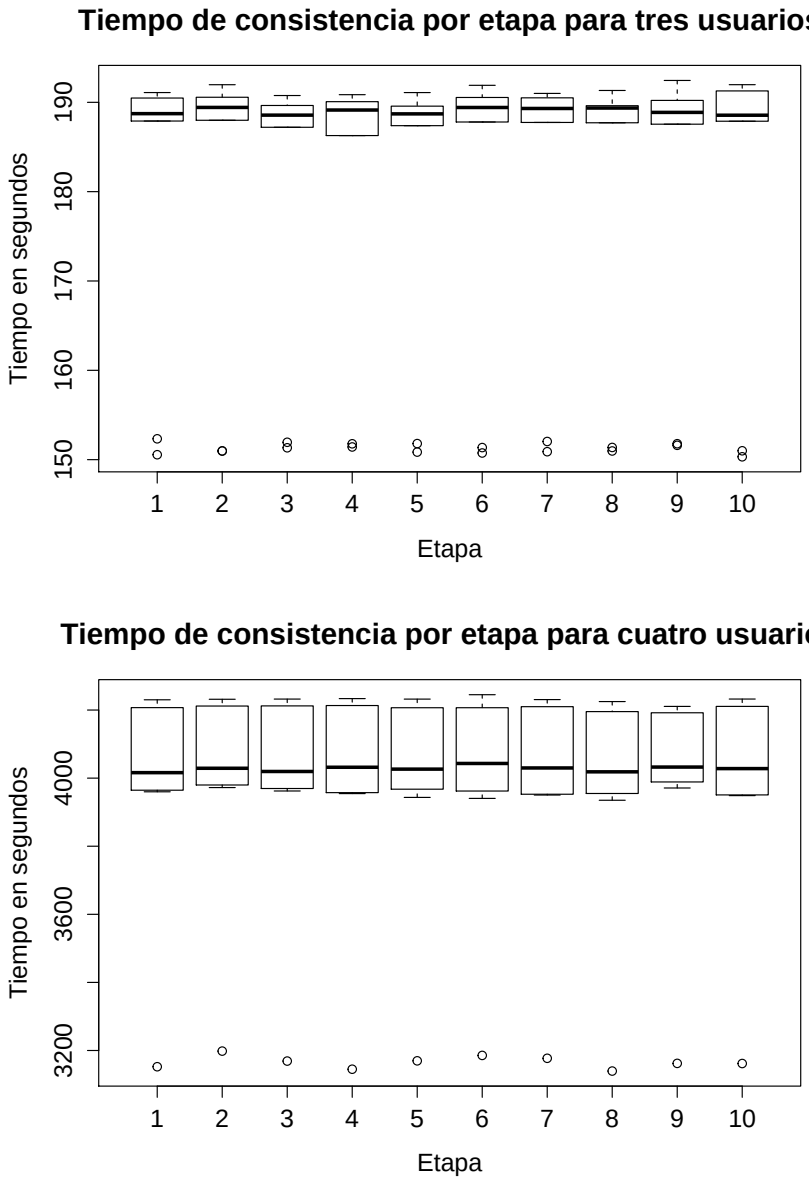


Figura 6.16: Tiempo transcurrido para determinar la consistencia de la ontología en cada etapa para tres y cuatro usuarios.

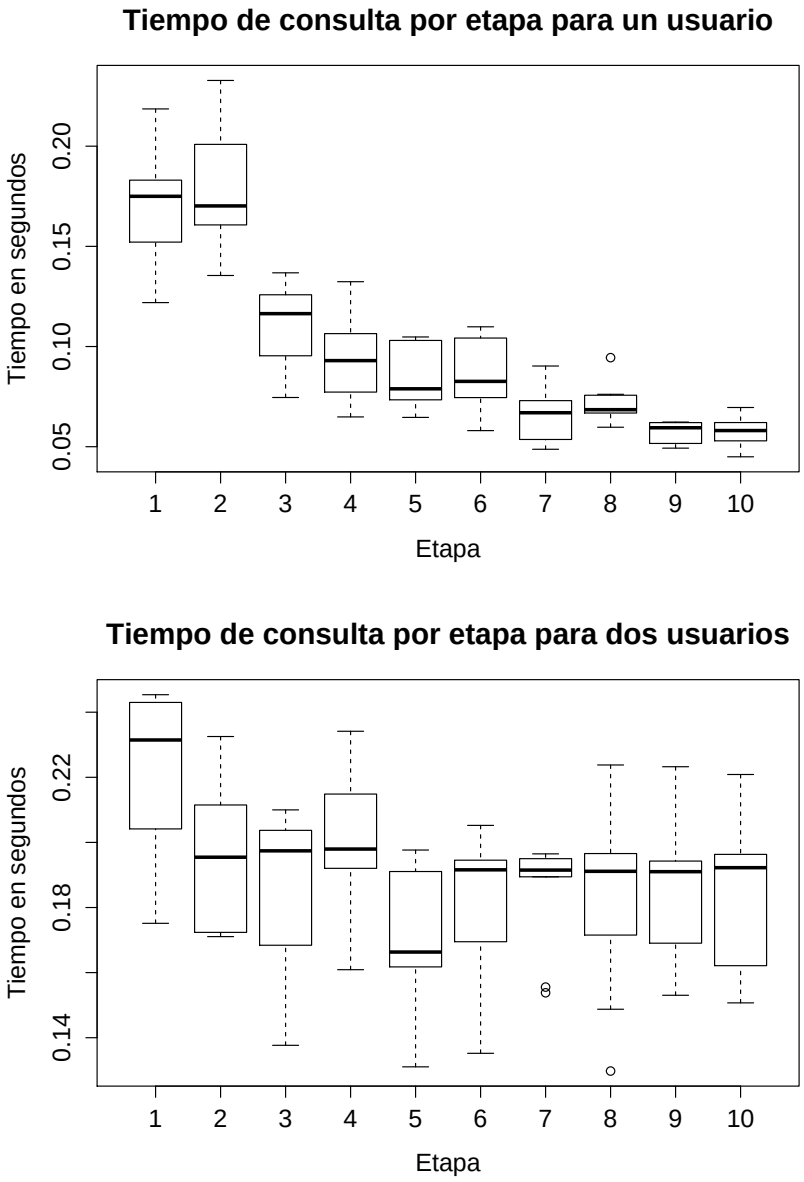


Figura 6.17: Tiempo empleado para responder una consulta probabilística para uno y dos usuarios.

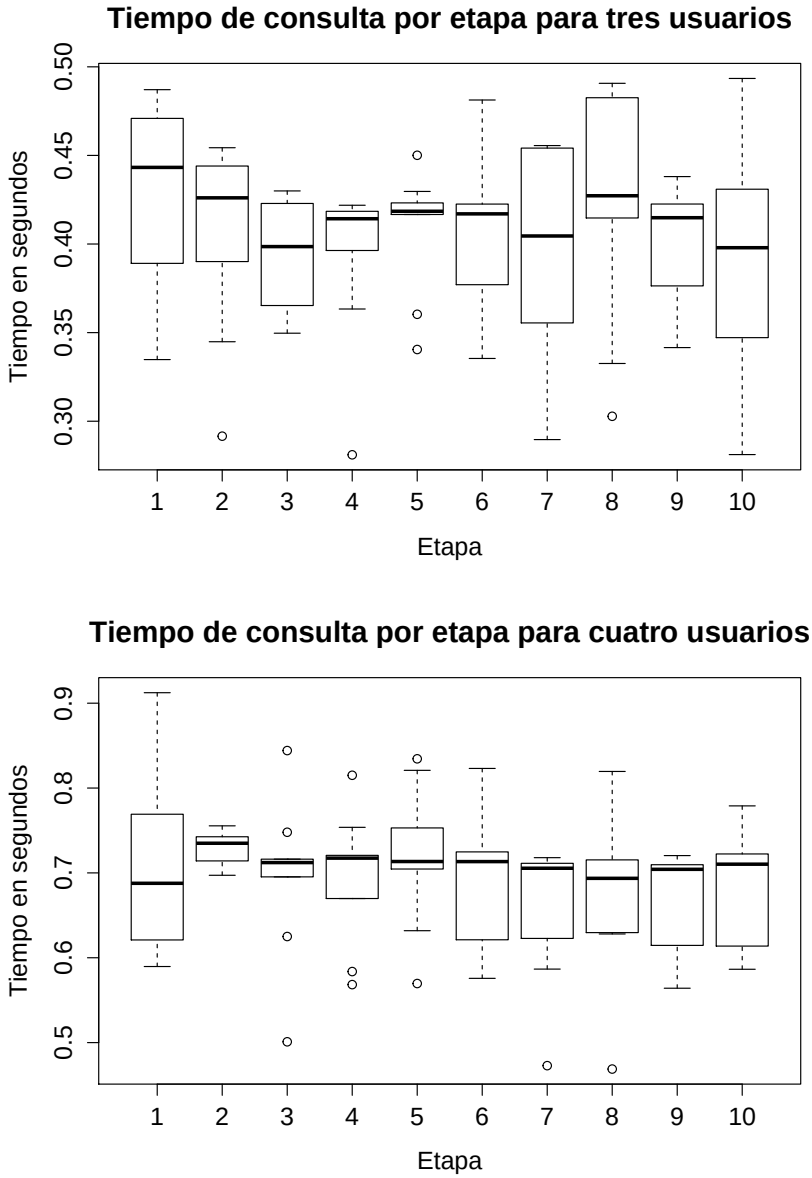


Figura 6.18: Tiempo empleado para responder una consulta probabilística para tres y cuatro usuarios.

Usuarios	*1*	*2*	*3*	*4*
Tiempo medio consistencia	0.3344821	3.079618	182.3587	3995.6524
Tiempo medio consulta	0.0161742	2.4841972	178.8057	3952.057
Diferencia de medias	0.3183079	0.5954208	3.5530	43.5954

Cuadro 6.2: Diferencia entre el tiempo medio de la consulta de contexto y el tiempo medio de consistencia en las etapas en segundos.

	1 Usuario	*2 Usuarios*	*3 Usuarios*	*4 Usuarios*
Memoria entregada	1000.5	1042.31	1073.74	1121.30
Memoria usada	86.23	190.55	231.86	300.11

Cuadro 6.3: Memoria en MB utilizada de media en el experimento Turambar.

tamaño de la pila (2 GB), siendo el pico 808.14 MB de memoria usada y 1320.5 MB, de memoria entregada. El programa fue ejecutado con un tamaño de pila mínima igual a 1000 MB y con el tamaño máximo por defecto, que en este caso fueron 2 GB.

6.2.3 Conclusiones del experimento

Para determinar el tiempo que se asume como aceptable para localizar un usuario se ha tenido en cuenta los tiempos de respuesta definidos por Jakob Nielsen (Nielsen, 1993b,a) para las intereaciones entre un usuario y un programa. Según este autor, los tiempos de respuesta iguales o inferiores a 0.1 segundos son aquellos que el usuario considera que el sistema reacciona instantáneamente. Los tiempos superiores a 0.1 segundos pero inferiores al segundo se encuentran en el límite del tiempo en el que el flujo de pensamiento del usuario puede estar ininterrumpido. Durante estos periodos de tiempo no es necesario el uso feedback adicional por parte del sistema. Por el contrario, 10 segundos es el límite de tiempo en el que se puede mantener la atención de un usuario enfocada en una interacción. Para demoras más largas, los usuarios querrán realizar otras tareas mientras se espera que el sistema termine, por lo que se debe dar retroalimentación que indique el sistema continúa trabajando.

Teniendo en cuenta estos valores, se puede afirmar que el rendimiento es bueno en aquellos escenarios mono usuarios, al estar el tiempo entre cambio de etapas por debajo del segundo. Con lo que se puede considerar validada la hipótesis de la tesis, para este caso que además es el caso más común, seguido de los entornos con dos usuarios. Por el contrario, el escenario de dos usuarios poseería un rendimiento adecuado para continuar manteniendo la atención del usuario. No obstante, conviene hacer una apreciación importante: los tiempos de etapas recogen el tiempo completo que se tarda en hacer un cambio en el estado de la ontología, el cálculo de la consistencia y la consulta. En otras palabras, representan el peor de los casos posibles de interacción con la aplicación. Es decir, cuando se intenta consultar el contexto y justo se ha producido un cambio. Por lo que en la mayoría de las veces, el tiempo de la interacción con la aplicación vendría marcado por la consulta en un entorno real. Si se tiene en cuenta solo el tiempo de la respuesta este se llega a encontrar en muchas consultas por debajo de los 0.1 segundos para un único usuario y por debajo de debajo de los 0.3 segundos en el caso de dos usuarios.

En cuanto a los escenarios de más de dos usuarios, la interacción con el sistema no es viable debido a los altos tiempos de cada etapa. No obstante, la mayor parte de los tiempos de la etapa se consumen en el cálculo de la consistencia. En cambio, el tiempo de la consulta sí está por debajo del segundo. Por lo que si se consiguiese reducir el tiempo de cálculo de la consistencia, la interacción entre usuarios y el sistema serían viables para este caso de uso. Otra conclusión que se puede extraer del experimento es que la implementación actual de SPARQL-DL que se utiliza no es la más idónea para su uso en Turambar y debería de mejorarse de cara a mejorar el rendimiento general. No obstante, el trabajo que esto implica queda fuera del ámbito de la tesis al ser un problema que la implementación arrastra desde antes de la extensión.

A pesar de los inconvenientes que SPARQL-DL presenta, sí parece claro que con ABoxes más pequeños si ofrece un rendimiento adecuado para entornos inteligentes, véase el caso de uno y dos usuarios. Por lo que el uso de Turambar podría ser el adecuado en aquellos casos en los que el tiempo de consulta de contexto no fuese demasiado elevado, bien debido a la complejidad de la consulta misma o la complejidad de la ontología. Esto mismo se aplicaría a las consultas probabilísticas que hicieran uso de la extensión de SPARQL-DL.

Por otra parte, la implementación de Turambar posiblemente podría mejorar el tiempo si se paralizase parte de las tareas. No obstante, esta tarea no es sencilla pues posiblemente habría que realizar algunos cambios en los razonadores para que soporten la concurrencia. Al menos, eso se pudo observar en una prueba de concepto que se realizó. Tal vez, también se pueda mejorar el tiempo mediante un mejor uso de las cachés. En su momento, la introducción de las cachés en Turambar significó una mejora del tiempo de ejecución bastante grande, de hasta un 40 % según las pruebas realizadas, por ello creo que una revisión de su uso podría mejorar algo el rendimiento.

Como dato positivo a nivel general, se puede extraer que no parece que exista un degradación en cuanto al tiempo de ejecución, consistencia y adición de nueva información; ya que no se observa un decaimiento en los tiempos de ejecución entre etapas.

Desde el punto de vista cualitativo, es claro que la introducción de la incertidumbre al modelado de contexto ofrece nuevas capacidades que no existían en OWL, como el ejemplo de este escenario demuestra. Sin la inclusión de la probabilidad no sería posible determinar la localización de un usuario cuando esta no esté disponible sin perder la noción de la certeza de un hecho. Este tipo de conocimiento no podría ser representado en OWL sin tener que recurrir a alguna solución que extienda OWL mediante algún sistema de representación propia de la incertidumbre.

Además, la extensión de un lenguaje probabilístico de consultas ofrece la oportunidad de realizar consultas complejas que permiten combinar axiomas tradicionales con las aserciones probabilísticas, dotando de una mayor expresividad a las consultas y precisión.

6.3 Conclusiones

En este capítulo se han presentado los experimentos que se realizaron para determinar qué herramientas se debían de elegir para el desarrollo del razonador con incertidumbre. Además, se ha presentado el experimento con el que se evalúa el rendimiento de Turambar. En él se explica cuándo debería de usarse y se extraen conclusiones sobre su rendimiento.

En el siguiente capítulo se exponen las conclusiones generales de la tesis y las líneas de investigación futuras.

Hay cosas que para saberlas no basta con haberlas aprendido.

Séneca

CAPÍTULO 7

Conclusiones

Completado el viaje, el aventurero hace balance de la aventura, mientras planea la siguiente. Las conclusiones de lo vivido le harán repetir la experiencia para buscar una ruta mejor o enfrentarse a nuevos desafíos.

Al igual que el aventurero, llegados a este punto es momento de recapitular y hacer balance. Por ello, en este capítulo se exponen las conclusiones generales de la tesis, así como las futuras líneas de investigación.

7.1 Conclusiones generales y contribuciones

En este trabajo se ha presentado el término de incertidumbre y se han explicado las acepciones más comunes del mismo, a parte de definir la acepción utilizada para el desarrollo de este trabajo. Según ella, la incertidumbre se produce por la falta de conocimiento. También se exponen los diferentes métodos para hacer frente a la incertidumbre y cuál es el elegido para este cometido: el enfoque probabilista. Además, se analizan las principales propuestas existentes para lidiar con la incertidumbre desde el ámbito de la probabilidad.

Dado que las propuestas analizadas no satisfacen las necesidades requeridas, se presenta un sistema propio de modelado de incertidumbre que combina las redes

bayesianas con las ontologías tradicionales y que además posee características diferenciales: uso de una única IRI para ser referida la ontología tanto para el uso con razonadores tradicionales, como con razonadores que implementen la especificación de Turambar; y que el sistema de funciones para evaluar estados de Turambar es extensible, obteniendo las dependencias de manera automática y transparente. Estas características son importantes porque permiten compartir ontologías probabilísticas con razonadores tradicionales de manera transparente y sin que tengan acceso al conocimiento probabilístico, evitando así modificaciones no deseadas y un incremento innecesario del número de axiomas. Además, ofrecen una solución sencilla para la introducción de nuevas características al razonamiento probabilístico mediante la introducción de extensiones que complemente el funcionamiento básico del razonador.

La definición de las redes bayesianas se realiza mediante el uso de una ontología que ha sido diseñada para la ocasión y que también permite definir las extensiones adicionales que se puedan necesitar para evaluar los estados de los nodos de la red. Además, existe una segunda ontología que permite relacionar de manera transparente la ontología que define la red con la ontología que describe el vocabulario del dominio.

En cuanto al razonamiento tradicional de OWL realizado por Turambar, este puede ser realizado por cualquiera de los razonadores soportados: HermiT y Pellet. La decisión depende del desarrollador y sus necesidades.

Adicionalmente, se ha extendido el lenguaje de consulta SPARQL-DL para ofrecer un lenguaje declarativo con el que consultar las ontologías de Turambar. SPARQL-DL fue elegido como alternativa a SPARQL debido a que no existían implementaciones de SPARQL para OWL API y a que el lenguaje de SPARQL-DL hace uso de query atoms, haciéndolo más propicio y lógico la extensión del mismo. De esta forma, se provee un método para realizar consultas complejas que hagan uso tanto de la probabilidad como del conocimiento tradicional de OWL. Además, este lenguaje presenta una alternativa al uso de la API de Turambar y podría ser adaptada por otros razonadores probabilistas con el objetivo de tener un lenguaje de consultas común entre razonadores.

También, se ha evaluado el razonador mediante un caso de uso en el que se pone de manifiesto el valor de Turambar para aquellas situaciones de los entornos

inteligentes en los que exista incertidumbre.

Turambar ofrece la ventaja de estar centrado en entornos inteligentes frente a sus competidores, por lo que las características del razonamiento en las que se centra están orientadas a este propósito. De la misma forma, cualquier cambio futuro estará dirigido a satisfacer las necesidades de los entornos inteligentes. De las tres soluciones analizadas de extensiones probabilísticas de OWL es la única que ofrece razonamiento probabilístico sobre aserciones de propiedades sobre clases y propiedades al mismo tiempo, junto con PR-OWL. Respecto a esta última se diferencia en que el modelado del conocimiento probabilístico queda aislado de la ontología que define el conocimiento del dominio. Además, la ontología probabilística se puede cargar en razonadores tradicionales como si fuese una ontología normal, sin que este tenga acceso a la definición del modelo probabilístico, cosa que en PR-OWL no es posible. Esto permite compartir las ontologías entre diferentes aplicaciones de manera más sencilla sin que esto suponga un esfuerzo extra en las tareas de razonamiento. Su desventaja frente a PR-OWL es que en el razonamiento se asume la representación atributo-valor, en los que cada tarea de razonamiento implica el mismo número de atributos. No obstante, este inconveniente puede ser solucionado en algunas ocasiones mediante un modelado inteligente del conocimiento. Además, se prefiere que el tamaño de la red bayesiana sea fija para controlar de una manera más efectiva el rendimiento del razonador, a diferencia de PR-OWL que hace uso de Situation Specific Bayesian Networks para consturir la red en función de la consulta. Ambas soluciones, presentan la desventaja de que la construcción de la red tiene que ser definida previamente. De las soluciones presentadas la única que ofrece esta característica es BayesOWL.

Si se atienden a los resultados obtenidos en la evaluación, se puede observar que e Turambar cumple la hipótesis de la tesis, ya que es capaz de ofrecer ayuda para la toma de decisiones cuando existe ausencia de información. Por otra parte, el rendimiento mostrado en la evaluación es bueno en entornos de uno y dos usuarios que son los escenarios más comunes. No obstante, en entornos de más de dos usuarios el rendimiento podría no ser el adecuado, debido en gran parte al tiempo empleado por la consulta de contexto. Como recomendación general para su uso, se debería de tener en cuenta el tamaño del ABox y la complejidad de las consultas

a realizar, especialmente la de contexto. No obstante, nada de esto es óbice para reconocer que una de las tareas futuras debe de ser buscar una mejora de rendimiento de la aplicación.

7.1.1 Publicaciones

Durante el desarrollo de este trabajo se han realizado las siguientes publicaciones:

- Como primer autor:
  - AUSÍN, D., LÓPEZ-DE-IPIÑA, D., BRAVO, J., VALERO, M.A. & FLÓREZ, F. (2011). TALISMAN+: Intelligent system for follow-up and promotion of personal autonomy. In J. Bravo, R. Hervás & V. Villarreal, eds., *Ambient Assisted Living*, no. 6693 in Lecture Notes in Computer Science, 187–191, Springer Berlin Heidelberg.
  - AUSÍN, D., CASTANEDO, F. & LÓPEZ-DE-IPIÑA, D. (2012a). Benchmarking results of semantic reasoners applied to an ambient assisted living environment. In M. Donnelly, C. Paggetti, C. Nugent & M. Mokhtari, eds., *Impact Analysis of Solutions for Chronic Disease Prevention and Management*, no. 7251 in Lecture Notes in Computer Science, 282–285, Springer Berlin Heidelberg.
  - AUSÍN, D. & CASTANEDO, D., FEDERICO ANDLÓPEZ-DEIPIÑA (2012). On the measurement of semantic reasoners in ambient assisted living environments. In *Intelligent Systems (IS), 2012 6th IEEE International Conference*, 082–087, Sofia, Bulgaria.
  - AUSÍN, D., CASTANEDO, F. & LÓPEZ-DE-IPIÑA, D. (2012b). TURAMBAR: An approach to deal with uncertainty in semantic environments. In J. Bravo, R. Hervás & M. Rodríguez, eds., *Ambient Assisted Living and Home Care*, no. 7657 in Lecture Notes in Computer Science, 329–337, Springer Berlin Heidelberg.
  - AUSÍN, D., LÓPEZ-DE-IPIÑA, D. & CASTANEDO, F. (2014). A probabilistic OWL reasoner for intelligent environments. In *URSW 2014 Uncertainty Reasoning for the Semantic Web*, vol. Vol-1259, Riva del Garda, Italy,.

- Como coautor:
  - GÓMEZ-GOIRI, A., ORDUÑA, P., AUSÍN, D., EMALDI, M. & LÓPEZ-DE-IPIÑA, D. (2011). Collaboration of sensors and actuators through triple spaces. In *2011 IEEE Sensors*, 651–654, IEEE, Limerick, Ireland.
  - FONTECHA, J., AUSÍN, D., CASTANEDO, F., LÓPEZ-DE-IPIÑA, D., HERVÁS, R. & BRAVO, J. (2012). A new approach to prevent cardiovascular diseases based on SCORE charts through reasoning methods and mobile monitoring. In J. Bravo, R. Hervás & M. Rodríguez, eds., *Ambient Assisted Living and Home Care*, no. 7657 in Lecture Notes in Computer Science, 17–24, Springer Berlin Heidelberg.
  - HERVÁS, R., FONTECHA, J., AUSÍN, D., CASTANEDO, F., BRAVO, J. & LÓPEZ-DE-IPIÑA, D. (2013). Mobile monitoring and reasoning methods to prevent cardiovascular diseases. *Sensors*, **13**, 6524–6541.

7.2 Trabajo Futuro

Las áreas de trabajo futuro que se han identificado son las siguientes:

- *Mejorar el rendimiento de Turambar*. Una de las futuras líneas de trabajo debería de ser la mejora en el tiempo necesario para determinar el contexto de una consulta probabíslitica. Esta mejora podría alcanzarse bien mejorando el rendimiento de la implementación de SPARQL-DL o bien creando un lenguaje propio. Otra mejora que podría reducir el tiempo es la paralelización de parte de las tareas de razonamiento. Sin embargo, esta no es trivial ya que no todos los razonadores OWL pueden ser accedidos de manera segura desde diferentes hilos de ejecución. Finalmente, una mejora clara sería la elaboración de un algoritmo incremental de consistencia, que permitiese recalcular solo aquellas aserciones probabilísticas que pudiesen verse afectadas por las alteraciones de la base de conocimiento. No obstante, las tareas para lograr este objetivo son complejas debido a la existencia de las consultas de contexto y la posibilidad de incluir extensiones propias en las ontologías de Turambar.

- *Desarrollo de un lenguaje de consultas declarativo universal para todos los razonadores*. El lenguaje de consultas presentado añade funcionalidades que solo son útiles para Turambar. No obstante otros razonadores podrían ofrecer información probabilística de distinta naturaleza a la presentada por Turambar, como por ejemplo la probabilidad de que dos individuos sean distintos. El objetivo de esta tarea futura sería el de determinar el lenguaje declarativo que pudiese cubrir todos los casos de interés práctico y definir el resto de lenguajes como subconjuntos de este.

- *Aprendizaje automático de las redes y probabilidades*. Uno de los objetivos futuros es dotar al razonador de la capacidad de aprender por si solo las relaciones entre los nodos, así como su distribución de probabilidad. El objetivo es evitar la mediación de un experto del dominio que defina el modelo y sus probabilidades; así como la actualización continua de las distribuciones de probabilidad y relaciones.

- *Distribuir el razonamiento entre diferentes máquinas*. Una de las características de los entornos inteligentes es que existen diferentes dispositivos desplegados en el entorno y algunos de ellos pueden poseer una cierta capacidad de cómputo que podría ser utilizado para dividir la tarea de inferencia entre diferentes máquinas.

- *Desarrollo de herramientas gráficas que faciliten el uso de Turambar*. Durante el desarrollo de esta tesis se ha desarrollado un prototipo de interfaz de usuario, que a pesar de no estar completada y ser del todo funcional, ayuda enormemente al desarrollo de ontologías de Turambar. Por ello, creo que el desarrollo de una versión completa y funcional de un programa gráfico que asista al ingeniero ontológico en la creación de ontologías de Turambar podría ser de gran ayuda para su popularización. La interfaz podría ser bien una aplicación propia o bien un plugin que se integrase con Protégé[1].

[1] `http://protege.stanford.edu/`

7.3 Comentarios finales

Con esta tesis expongo el trabajo realizado durante casi cinco años. En ella describo los avances alcanzados a la hora de modelar la incertidumbre en entornos inteligentes, así como los resultados de dichos logros con la esperanza de que ellos puedan servir de guía a futuros investigadores.

Bibliografía

ALMEIDA, A. & LÓPEZ-DE-IPIÑA, D. (2012). Assessing ambiguity of context data in intelligent environments: Towards a more reliable context managing system. *Sensors*, **12**, 4934–4951. 3

AUSÍN, D., LÓPEZ-DE-IPIÑA, D., BRAVO, J., VALERO, M.A. & FLÓREZ, F. (2011). TALISMAN+: Intelligent system for follow-up and promotion of personal autonomy. In J. Bravo, R. Hervás & V. Villarreal, eds., *Ambient Assisted Living*, no. 6693 in Lecture Notes in Computer Science, 187–191, Springer Berlin Heidelberg.

AUSÍN, D., CASTANEDO, F. & LÓPEZ-DE-IPIÑA, D. (2012a). Benchmarking results of semantic reasoners applied to an ambient assisted living environment. In M. Donnelly, C. Paggetti, C. Nugent & M. Mokhtari, eds., *Impact Analysis of Solutions for Chronic Disease Prevention and Management*, no. 7251 in Lecture Notes in Computer Science, 282–285, Springer Berlin Heidelberg. 91

AUSÍN, D., CASTANEDO, F. & LÓPEZ-DE-IPIÑA, D. (2012b). TURAMBAR: An approach to deal with uncertainty in semantic environments. In J. Bravo, R. Hervás & M. Rodríguez, eds., *Ambient Assisted Living and Home Care*, no. 7657 in Lecture Notes in Computer Science, 329–337, Springer Berlin Heidelberg.

AUSÍN, D. & CASTANEDO, D., FEDERICO ANDLÓPEZ-DEIPIÑA (2012). On the measurement of semantic reasoners in ambient assisted living environments. In *Intelligent Systems (IS), 2012 6th IEEE International Conference*, 082–087, Sofia, Bulgaria. 91

AUSÍN, D., LÓPEZ-DE-IPIÑA, D. & CASTANEDO, F. (2014). A probabilistic OWL reasoner for intelligent environments. In *URSW 2014 Uncertainty Reasoning for the Semantic Web*, vol. Vol-1259, Riva del Garda, Italy,.

BAADER, F. & NUTT, W. (2003). Basic description logics. In F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi & P.F. Patel-Schneider, eds., *The Description Logic Handbook: Theory, Implementation, and Applications*, 43–95, Cambridge University Press. 3, 23

BAADER, F., KÜSTERS, R. & WOLTER, F. (2003). The description logic handbook. chap. Extensions to description logics, 219–261, Cambridge University Press, New York, NY, USA. 4, 32

BECHHOFER, S., VOLZ, R. & LORD, P. (2003). Cooking the semantic web with the owl api. In D. Fensel, K. Sycara & J. Mylopoulos, eds., *The Semantic Web - ISWC 2003*, vol. 2870 of *Lecture Notes in Computer Science*, 659–675, Springer Berlin Heidelberg. 90

BECKETT, D. (2004). RDF/xml syntax specification (revised). W3C recommendation, W3C, http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/. 10

BECKETT, D., BERNERS-LEE, T., CAROTHERS, G. & PRUD'HOMMEAUX, E. (2014). RDF 1.1 turtle. W3C recommendation, W3C, http://www.w3.org/TR/2014/REC-turtle-20140225/. 10

BORGIDA, A. & PATEL-SCHNEIDER, P.F. (1994). A semantics and complete algorithm for subsumption in the classic description logic. *Journal of Artificial Intelligence Research*, **1**, 277–308. 38

CALVANESE, D., GIACOMO, G.D., LEMBO, D., LENZERINI, M. & ROSATI, R. (2005). Dl-lite: Tractable description logics for ontologies. In M.M. Veloso & S. Kambhampati, eds., *AAAI*, 602–607, AAAI Press / The MIT Press. 40

CARROLL, J.J., DICKINSON, I., DOLLIN, C., REYNOLDS, D., SEABORNE, A. & WILKINSON, K. (2004). Jena: Implementing the semantic web recommendations. In *Proceedings of the 13th International World Wide Web Conference*

on Alternate Track Papers &Amp; Posters, WWW Alt. '04, 74–83, ACM, New York, NY, USA. 90

CARVALHO, R., LASKEY, K. & COSTA, P. (2010). Pr-owl 2.0-bridging the gap to owl semantics. In *Proceedings of the 6th International Workshop on Uncertainty Reasoning for the Semantic Web (URSW 2010), collocated with the 9th International Semantic Web Conference (ISWC 2010)*, 73–84. 46

CARVALHO, R.N. (2011). *Probabilistic Ontology: Representation and Modeling Methodology*. Ph.D. thesis, George Mason University. 46

CEYLAN, I.I. & PENALOZA, R. (2014). Reasoning in the description logic bel using bayesian networks. In *Proc. 4th International Workshop on Statistical Relational AI (StarAI 2014)*. 38

CHEN, H., PERICH, F., FININ, T. & JOSHI, A. (2004). Soupa: standard ontology for ubiquitous and pervasive applications. In *Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on*, 258–267. 2

CHEN, L. & NUGENT, C. (2009). Ontology–based activity recognition in intelligent pervasive environments. *International Journal of Web Information Systems*, **5**, 410–430. 2

CLARK & PARSIA, L. (2014). How can i run sparql queries with owlapi? Http://clarkparsia.com/pellet/faq/owlapi-sparql/. 79

COEN, M.H. (1998). Design principles for intelligent environments. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, 547–554, American Association for Artificial Intelligence, Menlo Park, CA, USA. 1

COSTA, P. (2005). *Bayesian semantics for the Semantic Web*. Ph.D. thesis, George Mason University. 37, 44, 56

DEAN, M., CONNOLLY, D., VAN HARMELEN, F., HENDLER, J., HORROCKS, I., MCGUINNESS, D.L., PATEL-SCHNEIDER, P.F. & STEIN, L.A. (2002). Owl web ontology language 1.0 reference. Tech. rep., W3C Working Draft. 2

DERIVO (2014). Sparql-dl syntax. Http://www.derivo.de/en/resources/sparql-dl-api/sparql-dl-syntax.html. 80

DEY, A.K. (2001). Understanding and using context. *Personal Ubiquitous Comput.*, **5**, 4–7. 1

DING, Z. (2005). Bayesowl: A probabilistic framework for uncertainty in semantic web. 41, 48

DUERST, M. & SUIGNARD, M. (2005). RFC 3987: Internationalized Resource Identifiers (IRIs). RFC 3987 (Proposed Standard), see `http://www.ietf.org/rfc/rfc3987.txt`. 12

D'AMATO, C., FANIZZI, N. & LUKASIEWICZ, T. (2008). Tractable reasoning with bayesian description logics. In S. Greco & T. Lukasiewicz, eds., *Scalable Uncertainty Management*, vol. 5291 of *Lecture Notes in Computer Science*, 146–159, Springer Berlin Heidelberg. 37, 40

FONTECHA, J., AUSÍN, D., CASTANEDO, F., LÓPEZ-DE-IPIÑA, D., HERVÁS, R. & BRAVO, J. (2012). A new approach to prevent cardiovascular diseases based on SCORE charts through reasoning methods and mobile monitoring. In J. Bravo, R. Hervás & M. Rodríguez, eds., *Ambient Assisted Living and Home Care*, no. 7657 in Lecture Notes in Computer Science, 17–24, Springer Berlin Heidelberg.

FUKUSHIGE, Y. (2004). Representing probabilistic knowledge in the semantic web. In *Proceedings of the W3C Workshop on Semantic Web for Life Sciences*. 41

GAO, S.S., SPERBERG-MCQUEEN, C.M. & THOMPSON, H.S. (2009). W3C xml schema definition language (xsd) 1.1 part 1: Structures). W3C candidate recommendation, W3C, http://www.w3.org/TR/2009/CR-xmlschema11-1-20090430/. 10

GIANNAKOURIS, K. (2010). Regional population projections europop2008: Most eu regions face older population profile in 2030. Tech. rep., Eurostat. 1

GRAU, B.C., HORROCKS, I., MOTIK, B., PARSIA, B., PATEL-SCHNEIDER, P. & SATTLER, U. (2008). Owl 2: The next step for owl. *Web Semantics: Science, Services and Agents on the World Wide Web*, **6**, 309 – 322, semantic Web Challenge 2006/2007. 2

GRAU, B.C., PATEL-SCHNEIDER, P. & MOTIK, B. (2012). OWL 2 web ontology language direct semantics (second edition). W3C recommendation, W3C, http://www.w3.org/TR/2012/REC-owl2-direct-semantics-20121211/. 10

GROUP, W.O.W. (2009). OWL 2 Web Ontology Language Document Overview. Recommendation, W3C, `http://www.w3.org/TR/2009/REC-owl2-overview-20091027/`. 3

GROUP, W.O.W. (2012). Owl 2 web ontology language document overview (second edition). Tech. rep., http://www.w3.org/TR/owl2-overview/. 3, 9

GUHA, R. & BRICKLEY, D. (2014). RDF schema 1.1. W3C recommendation, W3C, http://www.w3.org/TR/2014/REC-rdf-schema-20140225/. 90

GUO, Y., PAN, Z. & HEFLIN, J. (2005). Lubm: A benchmark for owl knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, **3**, 158 – 182, selcted Papers from the International Semantic Web Conference, 2004 ISWC, 2004 3rd. International Semantic Web Conference, 2004. 90

GÓMEZ-GOIRI, A., ORDUÑA, P., AUSÍN, D., EMALDI, M. & LÓPEZ-DE-IPIÑA, D. (2011). Collaboration of sensors and actuators through triple spaces. In *2011 IEEE Sensors*, 651–654, IEEE, Limerick, Ireland.

HAYES, P. (2004). RDF semantics. W3C recommendation, W3C, http://www.w3.org/TR/2004/REC-rdf-mt-20040210/. 10

HEINSOHN, J. (1994). Probabilistic description logics. In *Proceedings of the Tenth International Conference on Uncertainty in Artificial Intelligence*, UAI'94, 311–318, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. 37

HERVÁS, R., FONTECHA, J., AUSÍN, D., CASTANEDO, F., BRAVO, J. & LÓPEZ-DE-IPIÑA, D. (2013). Mobile monitoring and reasoning methods to prevent cardiovascular diseases. *Sensors*, **13**, 6524–6541.

HORRIDGE, M. & PATEL-SCHNEIDER, P. (2012). OWL 2 web ontology language manchester syntax (second edition). W3C note, W3C, http://www.w3.org/TR/2012/NOTE-owl2-manchester-syntax-20121211/. 10

HORROCKS, I., PATEL-SCHNEIDER, P.F., BOLEY, H., TABET, S., GROSOF, B. & DEAN, M. (2004). Swrl: A semantic web rule language combining owl and ruleml. W3c member submission, World Wide Web Consortium. 91

HORROCKS, I., KUTZ, O. & SATTLER, U. (2006). The even more irresistible $\mathcal{SROIQ}$. In *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR2006)*, 57–67, 10th International Conference on Principles of Knowledge Representation and Reasoning, AAAI Press. 11, 23

HORROCKS, I., KRÖTZSCH, M., GLIMM, B. & SMITH, M. (2012). OWL 2 web ontology language conformance (second edition). W3C recommendation, W3C, http://www.w3.org/TR/2012/REC-owl2-conformance-20121211/. 10

HUYNH, T., FRITZ, M. & SCHIELE, B. (2008). Discovery of activity patterns using topic models. In *Proceedings of the 10th international conference on Ubiquitous computing*, 10–19, ACM. 91

JAEGER, M. *et al.* (1994). Probabilistic reasoning in terminological logics. *KR*, **94**, 305–316. 38

KLINOV, P. (2011). *Practical Reasoning in Probabilistic Description Logic*. Ph.D. thesis, The University of Manchester, Manchester, UK. 38, 39

KLINOV, P. & PARSIA, B. (2010). Pronto: A practical probabilistic description logic reasoner. In *International Workshop on Uncertainty in Description Logics*. 38

KOLLER, D. & FRIEDMAN, N. (2009). *Probabilistic Graphical Models - Principles and Techniques.*. MIT Press. 33

KOLLER, D., LEVY, A. & PFEFFER, A. (1997). P-classic: A tractable probabilistic description logic. In *Proceedings of AAAI-97*, 390–397. 38

KRÖTZSCH, M., SIMANCIK, F. & HORROCKS, I. (2012). A description logic primer. *arXiv preprint arXiv:1201.4089*. xvii, 23, 30, 31

KROTZSCH, M., SIMANCIK, F. & HORROCKS, I. (2014). Description logics. *Intelligent Systems, IEEE*, **29**, 12–19. 23, 32

LASKEY, K. (2008). Mebn: A language for first-order bayesian knowledge bases. *Artificial Intelligence*, **172**, 140–178. 44

LASKEY, K.J., LASKEY, K.B., COSTA, P.C.G., KOKAR, M.M., MARTIN, T. & LUKASIEWICZ, T. (2008). Uncertainty reasoning for the world wide web. Tech. rep., W3C, http://www.w3.org/2005/Incubator/urw3/XGR-urw3/. 3, 32

LASKEY, P.C.G.C..K.B. (2010). Pr-owl: A bayesian extension to the owl ontology language. 46

LIEBIG, T., LUTHER, M., NOPPENS, O. & WESSEL, M. (2011). OWLlink. *Semantic Web*, **2**, 23–32. 90

LUKASIEWICZ, T. & STRACCIA, U. (2008). Managing uncertainty and vagueness in description logics for the semantic web. *Web Semantics: Science, Services and Agents on the World Wide Web*, **6**, 291 – 308, semantic Web Challenge 2006/2007. 3, 4, 32, 37, 46

LUTZ, C. & SCHRÖDER, L. (2010). Probabilistic description logics for subjective uncertainty. In F. Lin, U. Sattler & M. Truszczynski, eds., *Principles of Knowledge Representation and Reasoning (KR 2010)*, 393–403, AAAI Press; Menlo Park, CA. 38

MITRA, P., NOY, N. & JAISWAL, A. (2005). Omen: A probabilistic ontology mapping tool. In Y. Gil, E. Motta, V. Benjamins & M. Musen, eds., *The Semantic Web – ISWC 2005*, vol. 3729 of *Lecture Notes in Computer Science*, 537–547, Springer Berlin Heidelberg. 41

MOTIK, B., GRAU, B.C., HORROCKS, I., FOKOUE, A. & WU, Z. (2012). OWL 2 web ontology language profiles (second edition). W3C recommendation, W3C, http://www.w3.org/TR/2012/REC-owl2-profiles-20121211/. 11

NIELSEN, J. (1993a). Response times: The 3 important limits. Http://www.nngroup.com/articles/response-times-3-important-limits/. 122

NIELSEN, J. (1993b). *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. 122

NOPPENS, O., LUTHER, M. & LIEBIG, T. (2010). The owllink api: Teaching owl components a common protocol. In *OWLED*. 90

PATEL-SCHNEIDER, P. & HORROCKS, I. (2006). OWL 1.1 Web Ontology Language overview. Tech. rep., World Wide Web Consortium, http://www.w3.org/Submission/owl11-overview/. 2

PATEL-SCHNEIDER, P., MOTIK, B. & PARSIA, B. (2012a). OWL 2 web ontology language XML serialization (second edition). W3C recommendation, W3C, http://www.w3.org/TR/2012/REC-owl2-xml-serialization-20121211/. 10

PATEL-SCHNEIDER, P., PARSIA, B. & MOTIK, B. (2012b). OWL 2 web ontology language structural specification and functional-style syntax (second edition). W3C recommendation, W3C, http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/. 10, 11, 12, 66

PEARL, J. (1986). Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, **29**, 241 – 288. 34

POOL, M. & AIKIN, J. (1994). Keeper and protégé: An elicitation environment for bayesian inference tools. In *Proceedings of the Workshop on Protégé and Reasoning, held at the 7th International Protégé Conference*. 41

PREDOIU, L. & STUCKENSCHMIDT, H. (2008). Probabilistic models for the semantic web: A survey. *The Semantic Web for Knowledge and Data Management: Technologies and Practices. Information Science Reference, Hershey, PA, USA*. 46

PRUD'HOMMEAUX, E., HARRIS, S. & SEABORNE, A. (2013). SPARQL 1.1 Query Language. Tech. rep., W3C, http://www.w3.org/TR/sparql11-query. 79

RIBONI, D. & BETTINI, C. (2011). {OWL} 2 modeling and reasoning with complex human activities. *Pervasive and Mobile Computing*, **7**, 379 – 395, knowledge-Driven Activity Recognition in Intelligent Environments. 3

RIGUZZI, F., BELLODI, E., LAMMA, E. & ZESE, R. (2013). Bundle: A reasoner for probabilistic ontologies. In W. Faber & D. Lembo, eds., *Web Reasoning and Rule Systems*, vol. 7994 of *Lecture Notes in Computer Science*, 183–197, Springer Berlin Heidelberg. 38

RIGUZZI, F., BELLODI, E., LAMMA, E. & ZESE, R. (2014). Probabilistic description logics under the distribution semantics. *Semantic Web – Interoperability, Usability, Applicability*, **To appear**. 38

SCHNEIDER, M. (2012). OWL 2 web ontology language RDF-based semantics (second edition). W3C recommendation, W3C, http://www.w3.org/TR/2012/REC-owl2-rdf-based-semantics-20121211/. 10

SCHREIBER, G. & RAIMOND, Y. (2014). RDF 1.1 primer. W3C note, W3C, http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/. 90

SCHREIBER, G., DEAN, M., BECHHOFER, S., VAN HARMELEN, F., HENDLER, J., HORROCKS, I., MCGUINNESS, D.L., PATEL-SCHNEIDER, P.F. & STEIN, L.A. (2004). OWL web ontology language reference. W3C recommendation, W3C, http://www.w3.org/TR/2004/REC-owl-ref-20040210/. 2

SHEARER, R., MOTIK, B. & HORROCKS, I. (2008). HermiT: A highly-efficient OWL reasoner. In *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2008)*, 26–27. 90

SIRIN, E. & PARSIA, B. (2007). Sparql-dl: Sparql query for owl-dl. In *OWLED*, vol. 258. xvii, 67, 79, 81

SIRIN, E., PARSIA, B., GRAU, B., KALYANPUR, A. & KATZ, Y. (2007). Pellet: A practical owl-dl reasoner. *Web Semantics: science, services and agents on the World Wide Web*, **5**, 51–53. 90

STRANG, T. & LINNHOFF-POPIEN, C. (2004). A context modeling survey. In *In: Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England*. 2

WANG, X., ZHANG, D.Q., GU, T. & PUNG, H. (2004). Ontology based context modeling and reasoning using owl. In *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*, 18–22. 2

WEITHÖNER, T., LIEBIG, T., LUTHER, M. & BÖHM, S. (2006). What's wrong with owl benchmarks. In *Proc. of the Second Int. Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2006)*, 101–114, Citeseer. 90

YANG, Y. & CALMET, J. (2005). Ontobayes: An ontology-driven uncertainty model. In *Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on*, vol. 1, 457–463, IEEE. 43

YUN PENG, Z.D. (2013). Bayesowl: Reference manual. Http://www.csee.umbc.edu/ ypeng/BayesOWL/manual/index.html. 43

Declaration

I herewith declare that I have produced this work without the prohibited assistance of third parties and without making use of aids other than those specified; notions taken over directly or indirectly from other sources have been identified as such. This work has not previously been presented in identical or similar form to any examination board.

The dissertation work was conducted from 2010 to 2015 under the supervision of Diego López de Ipiña and Federico Castanedo at the University of Deusto.

Bilbao,

This dissertation was finished writing in Bilbao on 24 de septiembre de 2015