UNIVERSIDAD OBERTA DE CATALUÑA

ESTUDIOS DE INFORMÁTICA, MULTIMEDIA Y COMUNICACIÓN

PROYECTO FIN DE CARRERA

MODELANDO CONOCIMIENTO PARA PLANIFICAR ASIGNATURAS EN LA UOC

Olga Alemán López

Junio 2014

Directores:
M. Antonia Sánchez Huertas
M. Elena Rodríguez González

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ABSTRACT

La extracción y representación del conocimiento es una tarea muy importante dentro de la inteligencia artificial y la ingeniería del conocimiento, ya que permite automatizar y simplificar los procesos de adquisición del conocimiento, automatizándolos y facilitando de este modo su tratamiento.

El conocimiento a representar en el presente trabajo es la planificación de una asignatura de la UOC. Dicha planificación, se realizará desde una doble perspectiva: se realizará una primera planificación estándar de la asignatura, realizada por el profesor responsable de la misma y en la que se incluyen todos los ítems y fechas claves. Una segunda planificación será realizada por el estudiante y partirá de esta primera, introduciendo las restricciones temporales personales del estudiante, y sus limitaciones horarias.

Previamente a la representación, se estudian tres modelos candidatos a tal efecto: las bases de datos orientadas a grafos, las ontologías, y la lógica temporal. Tras descubrir sus bondades y las aportaciones que estos modelos pueden aportar al problema objeto de estudio, se hace una propuesta de representación en la que se hace uso de los tres modelos: el modelo ontológico se usará para la especificación en la etapa de diseño conceptual; el modelo de base de datos orientado a grafos será de gran utilidad para estructurar y almacenar los datos, y por último, la lógica temporal tiene su cabida en la definición de las restricciones temporales.

Finalmente, se hará un pequeño ensayo de representación de un caso concreto: la programación del proyecto fin de carrera. Para ello, se seguirá el esquema teórico propuesto en la discusión del modelo.

I. DEFINICIÓN DEL PROBLEMA Y PLANIFICACIÓN

1. DEFINICIÓN DEL PROBLEMA

- Buscar un modelo de representación del conocimiento que represente de manera eficaz la programación de una asignatura, teniendo en cuenta tanto la carga lectiva como las fechas clave de entrega.
- Definición de todos los elementos que intervienen al cursar una asignatura. Elementos temporales que aparecen en instantes distintos de tiempo y tienen una duración temporal. Definición y clasificación de estos elementos temporales. Para ello, seguiré un razonamiento inductivo; desde la observación de los objetos, hasta la obtención de conclusiones. Haré uso de la siguiente matriz de inducción:

	Asignatura	Recursos Necesarios	Peso	Fecha Publicación	Fecha Entrega	Fecha Solución	Fecha Calificación	Calificación
Unidad didáctica								
PEC								
PRAC								
PF								

Ilustración 1. Matriz principales elementos

- Adquisición de los conocimientos necesarios para la realización de las distintas pruebas cumpliendo las fechas de entrega propuestas en la asignatura.
- Planificación del tiempo del estudiante. En esta planificación intervienen los elementos temporales antes descritos, el tiempo que el estudiante deberá dedicar a la asignatura, así como los recursos que el alumno deberá ir adquiriendo para la superación de la asignatura (conocimientos de otras materias por ejemplo). Esta planificación debe ser lo suficientemente flexible, con el fin de dar cabida a las distintas contingencias que puedan surgir al estudiante, así como la compatibilización de la asignatura con las actividades del estudiante al margen de la UOC.
- Preparación de la prueba final de evaluación

2. OBJETIVOS DESEABLES

- Proponer un modelo de representación del conocimiento eficaz e implementable para representar una planificación óptima de las asignaturas, compatible con el resto de actividades del estudiante.
- Mejorar la gestión del tiempo del alumno que cursa una determinada asignatura.

3. PLANIFICACIÓN

3.1. PEC1: definición del proyecto, y planificación del mismo
3.2. PEC2: partiendo del problema expuesto, seleccionar tres posibles modelos que representen fidedignamente el problema. En este punto quiero buscar el equilibrio entre lo que se puede expresar, los requisitos mínimos que me permitan definir la

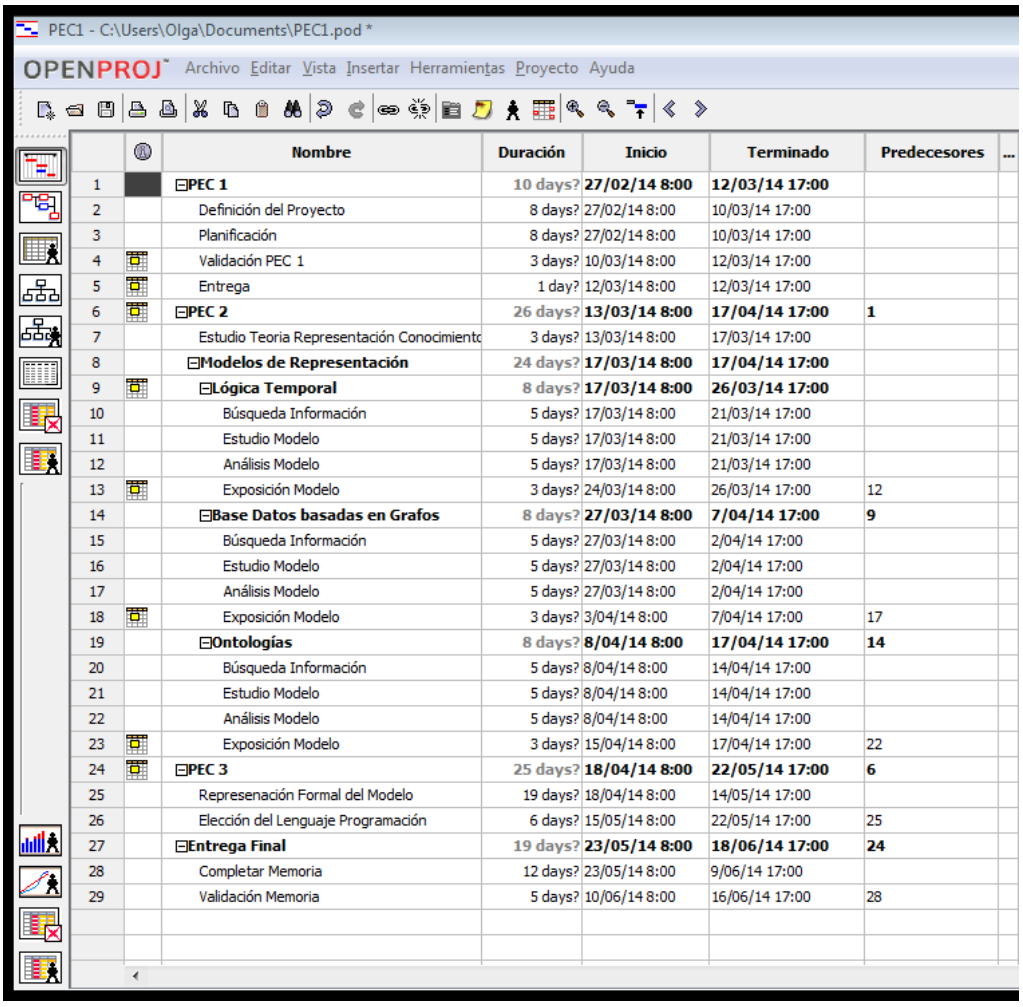
totalidad de mi escenario, y la eficiencia del modelo. Así mismo, realizaré un análisis detallado de las características de los tres modelos, sus pros y sus contras. Los modelos que estudiaré son:

a) Representación de bases de datos basadas en grafos
b) Ontologías
c) Lógica temporal

3.3. PEC3: Elección del modelo de representación de mi escenario.

a) Representación formal del modelo
b) Elección del lenguaje de programación

3.4. ENTREGA FINAL: entrega de la memoria del proyecto y presentación virtual



	Nombre	Duración	Inicio	Terminado	Predecesores
1	**PEC 1**	10 days?	**27/02/14 8:00**	**12/03/14 17:00**	
2	Definición del Proyecto	8 days?	27/02/14 8:00	10/03/14 17:00	
3	Planificación	8 days?	27/02/14 8:00	10/03/14 17:00	
4	Validación PEC 1	3 days?	10/03/14 8:00	12/03/14 17:00	
5	Entrega	1 day?	12/03/14 8:00	12/03/14 17:00	
6	**PEC 2**	26 days?	**13/03/14 8:00**	**17/04/14 17:00**	**1**
7	Estudio Teoria Representación Conocimiento	3 days?	13/03/14 8:00	17/03/14 17:00	
8	**Modelos de Representación**	24 days?	**17/03/14 8:00**	**17/04/14 17:00**	
9	**Lógica Temporal**	8 days?	**17/03/14 8:00**	**26/03/14 17:00**	
10	Búsqueda Información	5 days?	17/03/14 8:00	21/03/14 17:00	
11	Estudio Modelo	5 days?	17/03/14 8:00	21/03/14 17:00	
12	Análisis Modelo	5 days?	17/03/14 8:00	21/03/14 17:00	
13	Exposición Modelo	3 days?	24/03/14 8:00	26/03/14 17:00	12
14	**Base Datos basadas en Grafos**	8 days?	**27/03/14 8:00**	**7/04/14 17:00**	**9**
15	Búsqueda Información	5 days?	27/03/14 8:00	2/04/14 17:00	
16	Estudio Modelo	5 days?	27/03/14 8:00	2/04/14 17:00	
17	Análisis Modelo	5 days?	27/03/14 8:00	2/04/14 17:00	
18	Exposición Modelo	3 days?	3/04/14 8:00	7/04/14 17:00	17
19	**Ontologías**	8 days?	**8/04/14 8:00**	**17/04/14 17:00**	**14**
20	Búsqueda Información	5 days?	8/04/14 8:00	14/04/14 17:00	
21	Estudio Modelo	5 days?	8/04/14 8:00	14/04/14 17:00	
22	Análisis Modelo	5 days?	8/04/14 8:00	14/04/14 17:00	
23	Exposición Modelo	3 days?	15/04/14 8:00	17/04/14 17:00	22
24	**PEC 3**	25 days?	**18/04/14 8:00**	**22/05/14 17:00**	**6**
25	Represenación Formal del Modelo	19 days?	18/04/14 8:00	14/05/14 17:00	
26	Elección del Lenguaje Programación	6 days?	15/05/14 8:00	22/05/14 17:00	25
27	**Entrega Final**	19 days?	**23/05/14 8:00**	**18/06/14 17:00**	**24**
28	Completar Memoria	12 days?	23/05/14 8:00	9/06/14 17:00	
29	Validación Memoria	5 days?	10/06/14 8:00	16/06/14 17:00	28

Ilustración 2. Planificación del PFC

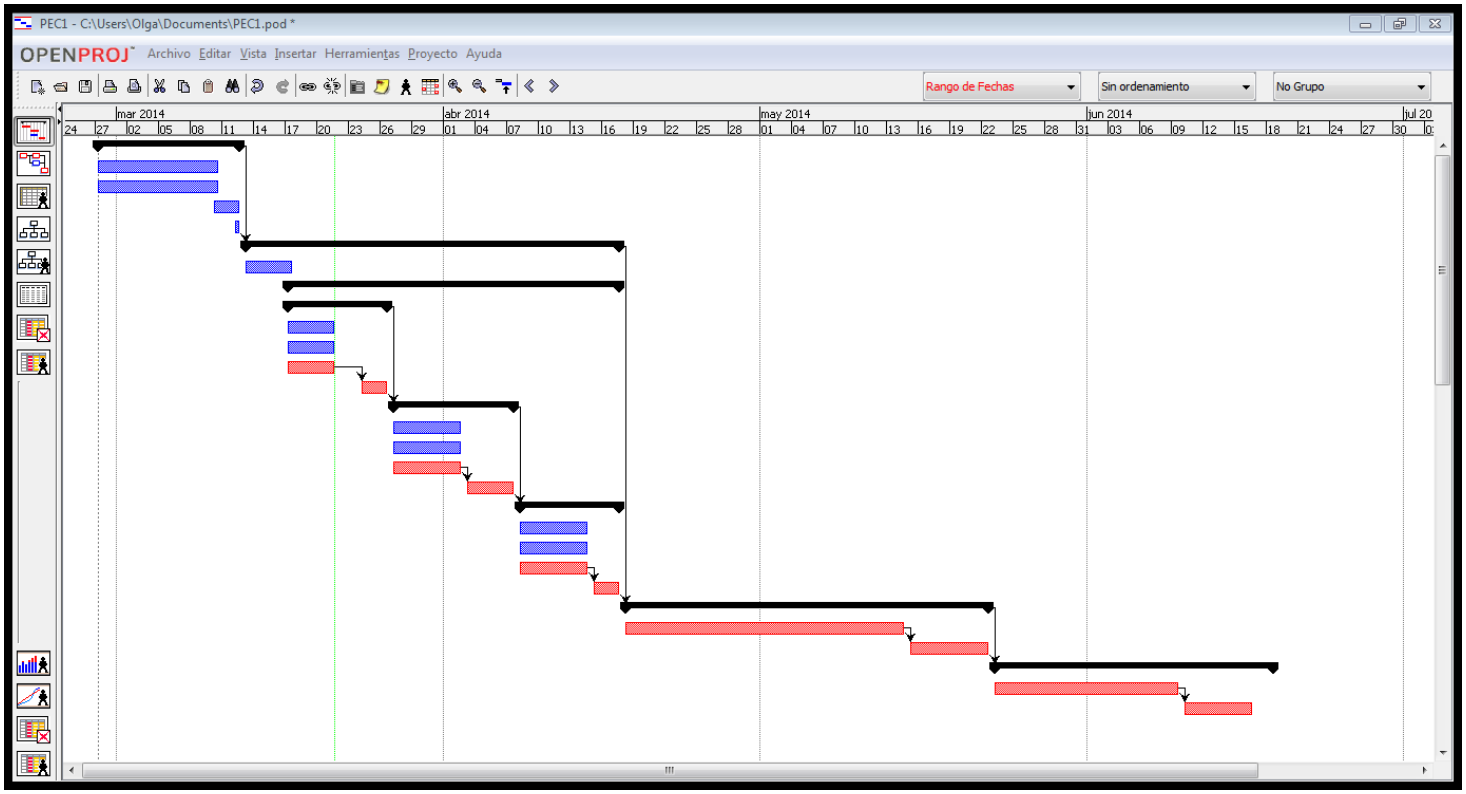


Ilustración 3.Diagrama Gantt de la planificación PFC

II. PRESENTACIÓN DE LOS MODELOS

1. BASES DE DATOS ORIENTADAS A GRAFOS

1.1. Justificación uso BD Orientada a Grafos

En este apartado se hará una introducción a la representación de bases de datos basada en grafos; la elección de este modelo parte de considerar que se trata de una simbología lo suficientemente flexible como para obtener más de una representación para un dominio dado, con datos complejos y fuertemente conectados. Esta cualidad permite a su vez al investigador realizar una experimentación y, con ella, abrir la puerta a la innovación a través de nuevas vías a explorar dirigidas a obtener la mejor representación posible para su dominio, es decir, la más eficaz y, a su vez, transversal de cara a su utilización práctica.

1.2. Antecedentes BD Orientadas a Grafos

Desde la aparición en 1970 de las bases de datos relacionales (Edgar F. Codd, 1970), pocos cambios han sufrido la concepción sobre las base de datos. Y aunque es indiscutible la gran utilidad y versatilidad de los sistemas de bases de datos relacionales (SGDBR), no es operativo usar SGDBR para todas las casuísticas, surgiendo la necesidad así de desarrollar nuevos enfoques. Por otra parte, existen varios problemas con los SGDBR actuales que pueden suponer un serio limitante para la construcción de aplicaciones. Estos problemas son en gran medida el motivo por el que surgió el movimiento NoSQL. Se exponen brevemente a continuación dichos problemas:

A. Lectura de datos costosa. En el modelo relacional los datos se representan mediante tablas relacionados entre sí. Y para realizar consultas sobre las tablas, es necesario una operación *join* (combinación), lo cual es costoso en términos de recursos de cómputo. Obviamente, hay estrategias para evitar dichos *joins* basadas en la desnormalización de los datos (tener pocas tablas y pocas relaciones entre ellas); sin embargo, uno de los pilares básicos de las BD relacionales es la normalización con el objetivo de evitar redundancias (es decir, repeticiones de los datos que son evitables).
B. Abuso del uso de las transacciones. El modelo relacional tiene escrituras de datos rápidas y eficientes. Sin embargo, un elevado uso de transacciones para preservar la integridad de los datos puede conllevar a una penalización en el rendimiento por el uso de bloqueos (*locks*) a nivel de datos, que puede disminuir el rendimiento.
C. Escalabilidad del modelo de base de datos. Los SGDBR no escalan fácilmente. Si usamos un modelo de escalamiento vertical, lo que hacemos es inflar con más recursos (CPU, RAM, espacio en disco, etc.) a nuestro servidor de bases de datos. Este tipo de escalamiento está sujeto a los límites del hardware, y su coste aumenta exponencialmente. Llegará un punto en que ya no podamos seguir escalando. Por otra parte, puede ser muy complicado escalar horizontalmente un SGDBR. El escalamiento horizontal involucra usar más servidores de forma paralela. En definitiva, en la medida de lo posible, se debería evitar la ejecución de *joins* distribuidos, es decir, *joins* que impliquen la combinación de distintas tablas ubicadas en diferentes nodos físicos.
D. Hay problemas que no tienen buena representación en un modelo relacional. Si bien es posible representar la mayoría de modelos de dominio usando el paradigma relacional, no siempre resulta la mejor opción.

Las bases de datos NoSQL difieren con las bases de datos tradicionales (BD relacionales) en los siguientes puntos:

- No usan SQL como lenguaje principal de consultas. Por ejemplo, Cassandra utiliza CQL, MongoDB utiliza JSON, y BigTable utiliza GQL, una versión propia de Google basada en SQL.
- No requieren estructuras fijas como tablas para almacenar los datos. Las BD relacionales obligan a definir un esquema a priori: todas las filas de una misma tabla tienen la misma estructura, y es necesario crear primero las tablas, e insertar los datos después. En las BD NoSQL, en cambio, el esquema y la inserción de los datos se puede hacer simultáneamente, y la estructura de objetos puede variar. Se trata de no imponer un esquema prefijado en forma de tablas y relaciones entre ellas, sino de ir más allá, permitiendo almacenar información en otros formatos como clave-valor (similar a *tablas de hash*), objetos, cubos, documentos o grafos.
- No suelen garantizar transacciones: las propiedades ACID (atomicidad, consistencia, aislamiento y durabilidad) son sacrificadas en las bases de datos NoSQL por una cuestión de rendimiento. Sin embargo, las réplicas de unos mismos datos serán antes o después consistentes, a pesar de que durante un lapso de tiempo (ventana de inconsistencia) no coincidan los datos. El paradigma BASE (*Basically Available Soft-state Eventual Consistency*) asegura que dado un lapso de tiempo (supuestamente pequeño) sin cambios en unos mismos datos, se espera que todas las réplicas converjan a unos mismos valores (es decir, se volverá a un estado consistente, donde todas las réplicas de unos mismos datos vuelven a coincidir).
- No suelen soportar operaciones *join*: al disponer de un volumen de datos tan extremadamente grande suele resultar deseable evitar los *join*. Las soluciones más directas consisten en desnormalizar los datos, o bien realizar el *join* mediante software, en la capa de aplicación.
- Arquitectura distribuida: así como las bases de datos relacionales suelen centralizar los datos en grandes *mainframes*, o como mucho en esquemas *master-slave* (maestro-esclavo), en el caso de las bases de datos NoSQL la información se suele compartir mediante mecanismos de *tablas hash distribuidas* (DHT).
- Escalabilidad horizontal: como consecuencia directa de su arquitectura distribuida, las bases de datos NoSQL pueden escalar de manera flexible ante picos de tráfico o necesidades puntuales de procesamiento (y de almacenamiento e incremento en el número de usuarios). Suelen funcionar bastante bien en hardware de bajo coste y permiten añadir/retirar nuevas máquinas en caliente.

NoSQL no es un sustituto a las bases de datos relacionales, es solo un movimiento que busca otras opciones para escenarios específicos como los que mencionamos; más que “no uses SQL” se trata de “no uses sólo SQL”. Existen varias formas de NoSQL, que atacan los problemas del escalamiento, rendimiento y modelado de los datos de formas distintas. (Erick Camacho, 2010).

A. Bases de datos clave-valor: éstas son unas bases de datos muy simples, ya que simplemente almacena valores identificados por una clave. Normalmente, el valor guardado se almacena como un vector de *bytes*, y es capaz de guardar grandes cantidades de datos. Sin embargo, renuncian a funcionalidades presentes en otras bases de datos como son la verificación de la integridad de los datos, *foreign keys (*claves foráneas) o *triggers* (disparadores)*.* Todo esto deberá implementarse a nivel de aplicación. Las más usadas *son* VMWare Redis, Amazon SimpleDB, Oracle BerkeleyDB, Tokyo Cabinet y Riak.
B. Bases de datos documentales: están diseñadas en torno al concepto abstracto de *documento*. Los documentos se parecen, de algún modo, a registros, tuplas o filas en una base de datos relacional, pero son menos

rígidos: no se les requiere ajustarse a un esquema estándar ni tener todos los mismos atributos o claves. Los documentos se acceden mediante una clave única, o mediante atributos de los mismos, lo que permite acciones como la utilización de índices. Dichos atributos deberán estar presentes en todos los documentos para permitir el acceso. Entre las bases de datos de este tipo, están: MongoDB, Apache CouchDB, IBM Lotus Domino, Terrastore, Marklogic y RethinkDB

C. Bases de datos orientadas a grafos: estas bases de datos almacenan los datos en forma de grafo. Esto permite darle importancia no solo a los datos, sino a las relaciones entre ellos. De hecho, las relaciones también pueden tener atributos y es posible realizar consultas directas a relaciones, en vez de a los nodos. Además, al estar almacenadas de esta forma, es mucho más eficiente navegar entre relaciones que en un modelo relacional (la teoría de grafos es usada para recorrer estas bases de datos). Este tipo de bases de datos son muy útiles cuando la información se puede representar fácilmente como una red. Este es el caso de las redes sociales o sistemas de recomendación, donde por el tipo de información que manejan, resultan muy útiles las BD en grafo. Entre las implementaciones más usadas está InfoGrid, InfiniteGraph, Neo4j, DEX, Virtuoso.
D. Bases de datos orientadas a columnas: su principal característica es la de almacenar datos en forma de columna (atributos) en lugar de hacerlo por filas (registros), lo que posibilita un acceso muy rápido a grandes cantidades de datos. Las escrituras salen más perjudicadas en este tipo de almacenamiento, por lo que este tipo de soluciones es usado en aplicaciones con un índice bajo de escrituras pero muchas lecturas. Entre la más popular están Apache Cassandra, Project Gemini, Infobright, Vertica, QD Technology, Sybase y BigTable.

1.3. BD Orientada a Grafo

Una Base de Datos en Grafo (BDG, a partir de ahora) es una base de datos que tiene como propósito almacenar estructuras de datos que tienen topología de grafo, es decir, representan la información como nodos de un grafo y sus relaciones como las aristas del mismo. De esta forma, se puede usar la teoría de grafos para recorrer la base de datos cuando ésta se encuentra almacenada en forma de nodos (entidades) y aristas (relaciones). En este tipo de modelos, cada elemento contiene un puntero directo a sus elementos adyacentes, por lo cual no es necesario realizar consultas por índices. Además, las relaciones entre entidades son parte de la teoría de grafos, lo que facilita bastante el manejo de las relaciones. Estas características nos permiten construir modelos más sofisticados para nuestro dominio del problema.

Las BDG deben estar absolutamente normalizadas, lo que significa que cada tabla tendría una sola columna y cada relación tan solo dos columnas. Con esto, se consigue que cualquier cambio en la estructura de la información tenga un efecto tan solo local. Imponiendo esta restricción se consiguen tres grandes logros: optimizar el rendimiento, mejorar la integridad de los datos y una mayor disponibilidad operacional.

Una BDG tiene las siguientes características:

- El almacenamiento organizado en forma de grafo, con acceso a nodos y aristas. Los nodos almacenan entidades, y las aristas relaciones entre los datos.
- El almacenamiento está optimizado para realizar los recorridos a través del grafo, sin necesidad de usar un índice. Por ello, las consultas en una BDG se realizan

aprovechando la proximidad de los datos: a partir de uno o varios nodos raíz, se va recorriendo el grafo. Para la realización de consultas globales es necesario usar un mecanismo de indización.

- El modelo de datos es flexible, lo que en algunas ocasiones nos permite no declarar tipos de nodos o de aristas. Permite modificar el modelo de datos en cualquier momento. En consecuencia, el modelado de los datos es muy sencillo.
- Integra funciones para aplicar los algoritmos clásicos de la Teoría de Grafos (caminos más cortos, A*, medidas de centralidad, Dijsktra, etc.)
- Facilidad para adaptarse a modelos de datos cambiantes.
- Rendimiento: ofrecen un rendimiento que tiende a permanecer constante, al igual que el crecimiento de los datos. Esto se debe a que las consultas se realizan de manera gráfica, recorriendo las relaciones (aristas) que posea la BD. Como resultado, el tiempo de ejecución de esta consulta es proporcional solo al tamaño del subgrafo que tenga que recorrer para satisfacer esta consulta, en lugar del tamaño global del grafo.
- Flexibilidad: los grafos son aditivos, lo que significa que se pueden añadir nuevos tipos de relaciones, y nuevos nodos y subgrafos a una estructura ya existente sin alterar las consultas y la funcionalidad de la aplicación. Esto tiene consecuencias positivas para el desarrollador debido a la flexibilidad del modelo con grafos, lo que también tiene tendencia a realizar menos migraciones para poder reducir los gastos de mantenimiento y el riesgo que conlleva.

Como consecuencia de todas estas propiedades, podemos concluir que las bases de datos orientadas a grafos son capaces de procesar de forma eficiente datos altamente conectados, gestionan fácilmente modelos de datos complejos, y ofrece resultados excepcionales cuando se trata de lecturas locales recorriendo el grafo.

Sin embargo, un problema frecuente en las bases de datos orientadas a grafos es la limitación estructural de su tamaño: en la mayoría de sistemas la partición de grafos es muy compleja, especialmente cuando se tienen en cuenta parámetros como la latencia de red, los patrones de acceso al grafo y la evolución en tiempo real del grafo. Esta problemática es compartida con las BD relacionales. El escalado horizontal puede generar problemas si a la hora de partir un grafo, los subgrafos resultantes quedan en nodos distintos, lo que supondría un degradado de rendimiento a la hora de realizar las consultas. Esto contribuye a que el entorno centralizado sea más idóneo para esta clase de estructuras. Por tanto, la escalabilidad vertical es más conveniente en estos casos.

Cabe mencionar una de las extensiones más relevantes de los modelos en grafo, son los *triplestores* (almacenes de tripletas), que hacen posible el almacenamiento de ontologías en formato RDF o *Resource Description Framework* (Marco para la Descripción de Recursos).

Las tripletas RDF son uno de los mecanismos más utilizados para representar la semántica de los datos que existen en la Web. Las tripletas son tuplas de tres elementos: un recurso de origen, un recurso de destino y un predicado que indica cómo están relacionados estos recursos. La manera de identificar estos tres elementos es mediante URIs (*Uniform Resource Identifier* en inglés). Las URIs son identificadores de recursos web (Rodríguez Gonzalez, M. E., Conesa i Caralt, J.).

1.4. Algunas BD Orientadas a Grafos

1.4.1. Neo4j (Neo Technology, 2014)

Neo4j es un software libre de BD orientada a grafos, implementado en Java. Neo4j fue desarrollado por Neo Technology, y la versión 1.0 de Neo4j fue lanzada en febrero de 2010.

Su principal característica es que almacena los datos en un grafo, en el que los nodos son los vértices, y las relaciones entre los nodos los arcos entre vértices.

Neo4j es muy válido para soluciones tipo red social, sistemas de recomendación, mapas topográficos, averiguar el camino más corto entre dos puntos, donde la necesidad es la navegación óptima por los datos independientemente del volumen total.

Neo4j tiene las siguientes características:

- No se define ningún esquema, es *schemaless* (esquema flexible), el esquema está implícito en el grafo, y no es necesario definir un esquema previamente.
- Transacciones ACID: Neo4j impone que todas las operaciones que modifiquen los datos se hagan dentro de una transacción, garantizando la coherencia de los datos. Por ello se dice que Neo4j es robusto.
- La introducción de índices mejoran la búsqueda de nodos en la base de datos. Una vez definidas las propiedades por las que se indexará, Neo4j se encarga del mantenimiento y gestión de los índices.
- Gran capacidad para recorrer relaciones en un volumen de datos muy grande.
- Rápido recorriendo relaciones, este tipo de consultas se conoce como *queries* transversales.
- Lenguaje de consultas (*query*) propio, Cypher.
- Usa el lenguaje *script* de grafos Gremlin.
- Alta disponibilidad, instalación en diferentes maquinas con balanceador de carga.
- Multilenguaje, proporciona un servidor con una API REST o empotrable como una biblioteca Java, pudiendo utilizarse desde cualquier lenguaje.
- Procesamiento en grafo de forma nativa (*native graph processing*): cada nodo tiene una referencia directa a su nodo adyacente, esto hace que el tiempo de una *query* no dependa del tamaño total de la base de datos sino al área de búsqueda del grafo. Esta característica es clave para un alto rendimiento en las consultas.
- Almacenamiento en grafo de forma nativa (*native graph storage*): existen ficheros para nodos, relaciones y propiedades. Al estar las propiedades de cada nodo y relación almacenadas en un fichero diferente, el almacenamiento de nodos y relaciones se preocupa sólo de la estructura del grafo. Los tamaños son fijos y se puede obtener rápidamente en memoria los nodos en base a su identificador, porque se sabe exactamente en qué posición se encuentra éste.

Neo4j dispone de los siguientes elementos:

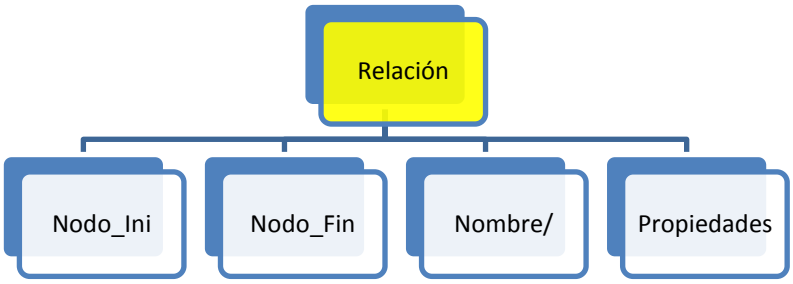


Ilustración 4. Elementos de Neo4j

A. Nodos**:** los nodos, junto con las relaciones, son las unidades fundamentales que forman los grafos. En Neo4j tanto los nodos como las relaciones pueden contener propiedades. Los nodos se utilizan para almacenar entidades. Pueden marcarse con cero o más etiquetas, al igual que las relaciones y las propiedades.

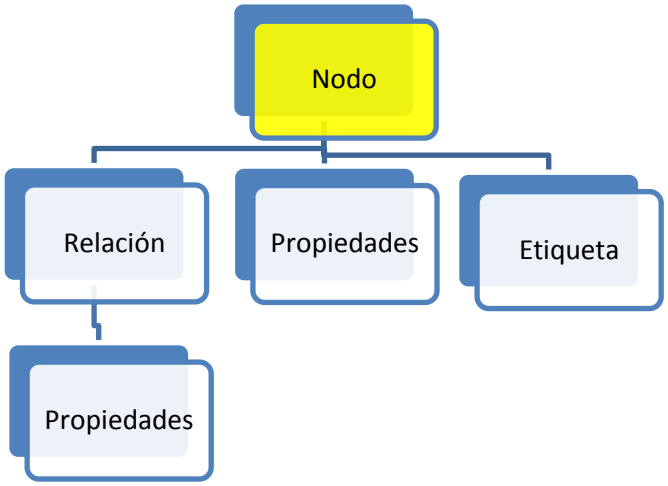


Ilustración 5. Nodos en Neo4j

B. Relaciones**:** una relación conecta dos nodos. Las relaciones entre nodos permiten encontrar datos relacionados. Son siempre dirigidas, y por tanto siempre tendrán un nodo saliente y otro entrante. Aun así, las relaciones se pueden recorrer en ambas direcciones. Un nodo también puede estar relacionado consigo mismo.

Ilustración 6. Relaciones en Neo4j

C. Propiedades**:** tanto los nodos como las relaciones pueden tener propiedades. Son pares clave-valor, donde la clave es un *string*. Las propiedades pueden ser primitivas (por ejemplo un *string*), o un conjunto de primitivas (un *array* de *string*). Los tipos más usuales son: *boolean, byte, short, int, long, float, double, char, string…*
D. Etiquetas: son estructuras que dan nombre a elementos y se utilizan para agrupar los nodos en conjuntos. Las consultas pueden operar sobre etiquetas, resultando más eficientes las consultas sobre muchos nodos. Se utilizan para reglas de restricción y añadir índices a las propiedades.
E. Caminos: un camino une uno o varios nodos recorriendo las relaciones entre ellos. Normalmente es el resultado de un recorrido.
F. Transversal: al recorrer un grafo, se puede obtener un subgrafo con los nodos visitados. Neo4j dispone de una API para recorrer los nodos que permite especificar las reglas del recorrido.
G. Esquema: Neo4j se puede usar con o sin esquemas. Si se usan, Neo4j garantiza la actualización de los índices de manera automática a medida que crece el grafo. Mediante reglas de restricciones Neo4j mantiene la integridad de los datos.

1.4.2. DEX (Sparsity Technologies)

DEX es una base de datos orientada a grafos escrita en C++ que permite analizar grandes volúmenes de datos. Su desarrollo empezó en el 2006 como un producto originado de la investigación de DAMA-UPC (grupo Data Management de la Universidad Politécnica de Catalunya (UPC), Barcelona). La primera versión estaba disponible en el tercer cuadrimestre del 2008. En Marzo del 2010 nació la empresa Sparsity Technologies creada desde la UPC para comercializar y dar servicios a las tecnologías desarrolladas en DAMA-UPC. En Febrero del 2014 para la quinta versión de la base de datos, DEX cambia su nombre a Sparksee.

DEX está basado en el modelo de base de datos en grafo, que está caracterizado por cumplir 3 propiedades:

- Las estructuras de los datos son grafos o estructuras similares a un grafo (tripletas).
- La manipulación de los datos y las consultas se realizan con operaciones orientadas a grafo.
- Existen restricciones para garantizar la integridad de los datos y de sus relaciones.

Las principales características de DEX son:

- Los grafos son multigrafo, ya que permite que existan múltiples aristas entre dos nodos aunque éstas sean del mismo tipo.
- Son grafos dirigidos. Permite que existan tanto aristas dirigidas como no dirigidas.
- Tienen etiquetas: tanto nodos como aristas pertenecen a tipos.
- Tienen atributos: nodos y aristas pueden tener tantos atributos como se desee.
- Capacidad de almacenamiento de datos y rendimiento, con órdenes de magnitud de miles de millones de nodos, aristas y atributos, gracias a una implementación con estructuras ligeras especializadas.
- Muy flexible para manejar esquemas desconocidos o dinámicos.
- Tiene unas consultas estructurales y permite la navegación entre consultas.
- Fácil mapeo entre ficheros CSV a DEX

- Una aplicación basada en Sparksee es capaz de gestionar más de una base de datos, trabajando cada una de ellas de forma independiente. Para ello, los accesos a las bases de datos se encierran en sesiones, y varias sesiones pueden acceder simultáneamente a la misma base de datos.

Elementos principales de DEX:

- Tipos. Los tipos Sparksee son identificados por un nombre único y un identificador exclusivo.
- Nodos y aristas. Los nodos y aristas son instancias de un cierto tipo. Cuando se crean se les da un identificador generado por Sparksee: el identificador de objeto (OID).
- Atributos. Se identifican por un nombre proporcionado por el usuario, y un identificador exclusivo generado por Sparksee. Los atributos se definen para un dominio concreto de datos. Los tipos de datos válidos en Sparksee son:
    - *Boolean.* Valores VERDADERO o FALSO
    - *Entero***.** Valores enteros de 32 bits con signo
    - *Largo***.** 64 bits valores enteros con signo
    - *Doble***.** 64 bits con signo
    - *Timestamp.* Distancia de Epoch (UTC) con una precisión de milisegundos
    - *Cadena de valores Unicode*. Longitud máxima se limita a 2.048 caracteres
    - *Texto***.** Objetos grandes de caracteres unicode
    - *OID Sparksee*. Valores de identificador de objeto

- Indexación. Para cada atributo de Sparksee se pueden crear diferentes índices, en función de lo cual existen tres tipos de atributos:
    - Atributos básicos. No tiene ningún índice asociado.
    - Atributos indexados. Hay un índice que mantiene el sistema de forma automática.
    - Atributos únicos. Igual que los atributos indexados pero con una restricción de integridad añadida para los atributos: dos objetos diferentes no pueden tener el mismo valor, a excepción el valor nulo.

- Sesión. Es un periodo de estado de actividad de un usuario con una base de datos. Todo tipo de manipulación de una base de datos debe estar encerrado en una sesión. Las sesiones incluyen gestión de transacciones y datos temporales.
- Transacciones. Una transacción Sparksee encierra un conjunto de operaciones que se ejecutan de manera atómica, y define el nivel de granularidad para la ejecución simultánea de las sesiones (Transacciones ACID). Hay dos tipos de operaciones: Leer (compartidas), y escribir (exclusivas**).** Sparksee se basa en el modelo de concurrencia *N-lectores 1-escritor***,** lo que significa que múltiples transacciones de lectura se pueden ejecutar al mismo tiempo, mientras que las transacciones de escritura se ejecutan en exclusiva. El tipo de transacción se define según las operaciones que se ejecuten; es decir, si una transacción empieza con una operación de lectura y un proceso actualiza en ese mismo instante la BD, dicha transacción se convierte en lectura y por tanto se vuelve exclusiva. Para poder ejecutarse una transacción de escritura, deben haber finalizado primero las transacciones de escritura.
- Datos temporales. Se gestiona automáticamente por el período de sesiones, y su vida finaliza cuando se cierra la sesión. Por esta razón, los datos temporales también pueden ser referidos como datos de sesión.

1.4.3. OrientDB

OrientDB es una base de datos escrita en código abierto Java y basada en NoSQL. Es una base de datos basada en documentos, pero las relaciones se gestionan como en las bases de datos orientadas a grafos con conexiones directas entre los registros. Además, ha sido diseñada para ser muy rápida. Hereda las mejores características y conceptos de documento, grafos y bases de datos de objetos.

Entre las características de OrientDB, se destacan las siguientes:

- Utiliza un nuevo algoritmo de indexación llamado MVRB-Tree, derivado de una mezcla de los algoritmos Arbol Rojo-Negro y B + Tree, con lo que consigue inserciones y búsquedas muy rápidas.
- Las transacciones que soporta son tanto operaciones atómicas, como las transacciones ACID. Utiliza un diario de escritura anticipada de registro (WAL), con el fin de grabar los cambios y poder recuperarlos en caso de fallo.
- Puede asignar documentos como cualquier otra base de datos de documentos, pero también puede conectar documentos como una base de datos relacional. La principal diferencia es que OrientDB no utiliza el costoso JOIN, sino que utiliza los enlaces directos muy rápidos, más propios de las base de datos de grafos. El conectar los documentos proporciona las siguientes ventajas:
    - Evita duplicados, siendo la base de datos resultante más pequeña y ligera.
    - Mejor uso de memoria RAM al ser una BD más pequeña, resultando un acceso a la misma más rápido al poder almacenarla en caché.
    - Puede ensamblar el documento completo usando conexiones transparentes al usuario.

- Tiene un fuerte sistema de seguridad basado en perfiles de usuario y roles.
- Soporta los modos *schemaless, schema-full* y *schema-mixed.*
- Soporta SQL como lenguaje de consulta, y se complementa con algunas extensiones para manipular árboles y grafos. No usa el JOIN.
- Idónea para aplicaciones Web: soporta de forma nativa HTTP, REST y JSON.
- Fácilmente integrable. Se puede ejecutar en modo local sin necesidad de conexión con el servidor.
- Es compatible con 3 algoritmos de indexación diferentes:
    - Índice SB-Tree
    - Hash índice
    - Índice MVRB-Tree

1.5. Conclusión

Las BDG son muy útiles para guardar información en modelos con muchas relaciones (muchos elementos interconectados con un número no determinado de relaciones entre ellos), como redes y conexiones sociales. Además, tienen una gran facilidad para modelar y adaptarse a modelos cambiantes, y dicho modelado puede hacerse fácilmente sin conocimientos técnicos explícitos.

Con respecto al problema objeto de este trabajo, es de gran utilidad esa característica innata para adaptarse a modelos cambiantes, lo cual permitiría la no decantación por un modelo de datos concreto a priori, pudiendo escoger el más adecuado cuando se tuviera una visión más detallada del problema. Además, las relaciones entre los objetos serán de crucial importancia a la hora de determinar las planificaciones; y este aspecto, acompañado de un volumen de datos no muy extenso, hacen de las BD orientadas a grafo una buena elección.

2. ONTOLOGÍAS

2.1. Justificación Uso Ontologías

Otro de los modelos seleccionados como instrumentos de representación se basa en las ontologías como sistemas de representación de conceptualizaciones de dominios del conocimiento. Su elección en este trabajo responde tanto a la eficacia que puede aportar esta herramienta para modelar, compartir y reutilizar las sinergias de conocimiento en las organizaciones, así como en su contribución de cara a la toma de decisiones a partir de las representaciones obtenidas en estos sistemas. Se trata, además, de una cuestión que resulta de plena vigencia en la permanente búsqueda de instrumentos de utilidad que persiguen las Ciencias de la Computación haciendo uso, a su vez, de los conocimientos que pueden aportar otras disciplinas.

2.2. Introducción al Concepto de Ontología

La palabra ontología se deriva del griego *ontos* (estudio del ser) y *logos* (palabra). Filosóficamente, la ontología es la ciencia del *qué es*, es una explicación sistemática de la existencia, de los tipos de estructuras, categorías de objetos, propiedades, eventos, procesos y relaciones de cada área con la realidad. Una definición clásica de la ontología la considera como la rama de la metafísica que estudia la naturaleza de la existencia. En el campo de la informática, sin embargo, una ontología es una entidad computacional, y ha de ser considerada como recurso artificial que se crea, en lugar de considerarse como una entidad natural que se descubre (Mahesh, 1996). Una ontología ha de verse como un entendimiento común y compartido de un dominio, que define el vocabulario de un área mediante un conjunto de términos básicos y las relaciones entre dichos términos, así como las reglas que combinan términos y relaciones. La ontología nos permite distinguir entre diferentes tipos de objetos y sus relaciones, dependencias y propiedades. Aplicando la ontología a los sistemas basados en conocimiento, podemos concluir que, lo que existe es exactamente lo que se puede representar. Desde esta perspectiva computacional, existen varias definiciones de distintos autores relevantes que contextualizan las ontologías eficazmente:

- Weigand (1997) ofrece una definición de ontología más concreta: a*n ontology is a database describing the concepts in the world or some domain, some of their properties and how the concepts relate to each other.* (Una ontología es una base de datos que describe los conceptos del mundo o de otro dominio, algunas de sus propiedades y cómo los conceptos están relacionados unos con otros).
- Por su parte, Gruber (1993) considera que *an ontology is an explicit specification of a conceptualization. The term is borrowed from philosophy, where an Ontology is a systematic account of Existence. For AI system, what exists is that which can be represented.* (Una ontología es una especificación explícita de conceptualización. El término proviene de la filosofía, donde una ontología es una explicación sistemática de la existencia. Para los sistemas IA, lo que existe se puede representar).
- La definición de Gruber fue clarificada por Borst (1997): *an ontology is a formal specification of a shared conceptualization.* (Una ontología es una especificación formal de una conceptualización compartida). Esta definición realiza dos grandes aportaciones claves: lenguaje formal, y conocimiento compartido.
- Por su parte, Studer (1998) explica y fusiona las definiciones de Gruber y Borst postulando lo siguiente: *an ontology is a formal, explicit specification of a*

shared conceptualisation (Una ontología es una especificación formal y explícita de una conceptualización compartida).

Las definiciones anteriormente expuestas, demuestran que existe una multiplicidad de interpretaciones posibles del concepto de ontología, que aportan distintos y complementarios puntos de vista, aún dentro de la misma área del conocimiento.

2.3. Uso de las Ontologías

El uso de las ontologías está extendido en muchos campos diferentes de la actividad humana. En concreto, la fase de análisis ontológico las hace propicias para distintos ámbitos por los beneficios que conlleva:

- Clarifican la estructura de conocimiento.
- Reducen la ambigüedad conceptual y terminológica.
- Permiten compartir conocimiento.

Según Guarino (1998), al analizar el impacto del uso de ontologías en el campo de los sistemas de información deben tenerse en cuenta dos dimensiones: el tiempo en el que son utilizadas y el aspecto estructural:

- Respecto al momento en que son utilizadas: el uso de ontologías puede llevarse a cabo durante el desarrollo o en tiempo de ejecución. Cuando la ontología se utiliza en tiempo de ejecución el sistema se dice *dirigido por ontología*, mientras que cuando las ontologías son utilizadas durante el desarrollo del sistema, este se dice *desarrollo dirigido por ontología*.
- Respecto a la dimensión estructural: el uso de las ontologías en bases de datos es muy frecuente por su gran capacidad de complementación respectiva. Además, las ontologías han sido exitosamente utilizadas para generar interfaces basadas en formularios que chequean por restricciones de violación de tipos. Durante el desarrollo se puede generar la parte estática de un programa con ayuda de una ontología. Además, las ontologías integradas con recursos lingüísticos pueden ser utilizadas para soportar el desarrollo de software orientado a objetos.

Las ontologías se están empleando en todo tipo de aplicaciones informáticas para definir un conjunto de entidades relevantes en un campo de aplicación determinado, así como las interacciones entre las mismas. Algunas ontologías se crean con el mero objetivo de alcanzar una comprensión del Universo de Discurso (UoD) pertinente, ya que su creación impone una especificación muy detallada. Otras ontologías han sido creadas con un propósito general, como por ejemplo el proyecto Cyc (Guha & Lenat 1990), que está orientado a la construcción de una base de conocimiento que contenga el conocimiento humano necesario para hacer inferencias.

Desde el punto de vista de la Inteligencia Artificial (IA), el concepto ontológico distingue dos corrientes principales. Por un lado, se considera que las ontologías definen los términos básicos y relaciones de un dominio particular. Por otro lado, la comunidad filosófica de la IA defiende la noción filosófica original de ontología y usan las ontologías de manera más formal. Actualmente, las ontologías son la tecnología más extendida para representar conocimiento en IA, porque proporcionan una representación estructurada y formal del conocimiento, y las representaciones ontológicas son compartibles y reutilizables. En IA, las ontologías son pesadas cuando incluyen axiomas, y son ligeras en otro caso.

Por otro lado, las ontologías se han convertido en herramientas que pueden asistir eficientemente en las actividades de desarrollo y mantenimiento de software ya que, al

reducir la ambigüedad y proveer un marco de unificación, ayudan a compartir conocimiento, facilitan la comunicación y permiten una alta reutilización de conocimiento. Y más concretamente, si las referenciamos dentro del campo de Ingeniería del Software, una ontología puede verse como un vocabulario de representación para un dominio específico, que representa elementos conceptuales y relaciones entre ellos; sin embargo la ontología no es el vocabulario en sí mismo, sino lo que él representa (Chandrasekaran B., Josephson J.R., Benjamins V., 1999).

La Web semántica fue impulsada por Tim Berners-Lee, creador de la WWW, y otras personas relacionados con el W3C (World Wide Web Consortium). La primera vez que salió a la luz dicho término fue en septiembre de 1998 en la publicación de dos documentos denominados *Semantic Web Road Map* y *What the Semantic Web can represent*, por parte de Berners-Lee. Introdujo la Web semántica como una red de documentos más inteligentes que permitirían, a su vez, búsquedas más inteligentes. La idea sería aumentar la inteligencia de los contenidos de las páginas web dotándolas de contenido semántico. Al dotar a la Web de más significado se pueden obtener soluciones globales a problemas habituales de búsqueda de información, gracias a la utilización de una infraestructura común, mediante la cual, es posible compartir, procesar y transferir información de forma sencilla (W3C, 1998).

Además, la Web semántica ayuda a resolver la sobrecarga de información, heterogeneidad de fuentes de información, y reusabilidad, ya que posee la capacidad de construir una base de conocimiento sobre las preferencias de los usuarios y que, a través de una combinación entre su capacidad de conocimiento y la información disponible en Internet, ésta es capaz de atender de forma exacta las demandas de información por parte de los usuarios.

En pocas palabras, la Web semántica trata de extender la Web actual añadiendo información semántica que los ordenadores puedan procesar de manera automática. Disponer de información bien definida a nivel semántico permite mejorar la interoperabilidad entre sistemas y usuarios, permitiendo ofrecer mejores servicios. Sin embargo, describir el conocimiento a nivel semántico es una labor costosa y complicada, así que resulta imprescindible disponer de tecnologías y herramientas que potencien la posibilidad de compartir y reutilizar dicho conocimiento. Uno de los principales mecanismos propuestos para aliviar este problema es el uso de ontologías. En este sentido, las ontologías permiten representar el conocimiento de manera intuitiva y extensible mediante el uso de jerarquías conceptuales.

A modo de síntesis, se enumeran a continuación los principales usos de las ontologías y sus respectivos objetivos:

A. Documentación
    - Representar el conocimiento de un dominio
    - Clasificar objetos de información
    - Recuperar información
B. Ingeniería del Software
    - Modelado de bases de datos
    - Modelado de aplicaciones
    - Ayudas gráficas para comunicación entre clientes, analistas y desarrolladores
C. Inteligencia Artificial
    - Imitar la mente humana
    - Almacenar conocimiento común
    - Mecanismos para realizar inferencias
    - Mecanismos de aprendizaje

- Capacidad de operar con sistemas informáticos

D. Web Semántica
   - Representar el conocimiento
   - Clasificar objetos de información
   - Recuperar información
   - Operar en sistemas heterogéneos
   - Interoperabilidad

2.4. Clasificación de las Ontologías

Las ontologías se pueden clasificar según distintos criterios, en función de la aplicación a la que vaya dirigida, así como del autor que realice dicha clasificación. A continuación citaré alguna de las clasificaciones disponibles en la literatura ontológica:

Con respecto al nivel de generalidad (Guarino, N. 1998):

- Ontologías de Alto Nivel: describen conceptos generales como espacio, tiempo, materia, objeto. Son independientes de un dominio o problema particular. Su intención es unificar criterios entre grandes comunidades de usuarios.
- Ontologías de Dominio: describen el vocabulario relacionado a un dominio genérico, por medio de la especialización de los conceptos introducidos en las ontologías de alto nivel.
- Ontologías de Tareas: describen el vocabulario relacionado a una tarea o actividad genérica, por medio de la especialización de los conceptos introducidos en las ontologías de alto nivel.
- Ontologías de Aplicación: describen conceptos que pertenecen a la vez a un dominio y a una tarea particular, por medio de la especialización de los conceptos de las ontologías de dominio y de tareas. Generalmente corresponden a roles que juegan las entidades del dominio cuando ejecutan una actividad.

Con respecto al tipo de estructura de conceptualización (Van Heijst, G., Schereiber, A.T. Y Wielinga, B.J, 1996)

- Ontologías Terminológicas: especifican términos a utilizarse para representar el conocimiento en el dominio de estudio. Intentan obtener un lenguaje unificado sobre un tema específico.
- Ontologías de Información: especifican la estructura de los registros de una base de datos, determinando un marco para el almacenamiento estandarizado de información.
- Ontologías de Representación de Conocimiento: especifican conceptualizaciones del conocimiento. Suelen estar enfocadas a un uso particular del conocimiento que describen.

Con respecto a los aspectos del mundo real que intentan modelar (Jurisica I., Mylopoulos J., Yu E., 1999)

- Ontologías Estáticas: describen las cosas que existen, sus atributos y las relaciones entre ellos. Términos que utilizan: atributos, relaciones y entidades dotadas de una identidad única e inmutable.
- Ontologías Dinámicas: describen los aspectos que pueden cambiar en el mundo que modelan. Para modelarlas se pueden utilizar máquinas de estados

finitos, redes de Petri, etc. Términos que utilizan: procesos, estados, transición de estados.
- Ontologías Intencionales: describen aspectos que tienen que se refieren al mundo de las motivaciones, intenciones, metas, creencias, alternativas y elecciones de los agentes involucrados. Términos que utilizan: aspecto, objetivo, soporte, agente.
- Ontologías Sociales: describen aspectos que se relacionan con lo social, estructuras organizacionales, redes, interdependencias. Términos que utilizan: actor, posición, rol, autoridad, compromiso.

Con respecto a la riqueza de la estructura interna (Corcho, O., Fernandez-Lopez, M., Gomez-Perez, A. 2003), clasificación propuesta por Lassila y McGuinness:
- Vocabularios controlados: lista finita de términos, por ejemplo un catálogo.
- Glosarios: son listas de términos con sus significados expresados en lenguaje natural.
- Tesauros: proveen semánticas adicionales entre términos, como por ejemplo información referida a sinónimos.
- Jerarquías Informales *es-un*: son jerarquías de términos que no corresponden a una subclase estricta.
- Jerarquías Formales *es-un*: en este caso existe una relación estricta entre instancias de una clase y de las superclases correspondientes. Su objetivo es explotar el concepto de herencia.
- Marcos: son ontologías que incluyen tanto clases como sus propiedades, las cuales pueden ser heredadas por otras clases en los niveles más bajos de una taxonomía formal *es-un*.
- Ontologías que expresan restricciones de valor: por ejemplo expresan restricciones de acuerdo al tipo de dato de una propiedad.
- Ontologías que expresan restricciones lógicas generales: son las ontologías más expresivas. Especifican, por medio de lógica de primer orden, restricciones entre los términos de la ontología.

Con respecto a la riqueza del sujeto de conceptualización (Van Heijst, G., Schereiber, A.T. Y Wielinga, B.J., 1996):
- Ontologías de representación de conocimiento: capturan primitivas de representación utilizadas para formalizar conocimiento bajo un paradigma de representación de conocimiento dado.
- Ontologías comunes o generales: representan conocimiento de sentido común y reutilizable en distintos dominios.
- Ontologías de alto nivel: son ontologías que describen conceptos y nociones generales bajo las cuales pueden enlazarse los términos raíces de todas las ontologías.
- Ontologías de dominio: son aquellas ontologías reutilizables en un dominio particular. Proveen vocabularios sobre conceptos dentro del dominio y sus relaciones.
- Ontologías de tareas: describen vocabulario relacionado a actividades genéricas. Proveen un vocabulario sistemático de términos utilizados para resolver problemas que pueden o no pertenecer a un mismo dominio.
- Ontologías de tareas de dominios: a diferencia de las ontologías de tareas, estas ontologías son reutilizables en un dominio dado, y no entre dominios diferentes.
- Ontologías de métodos: proveen definiciones de conceptos relevantes y relaciones aplicables a un proceso de razonamiento específico a fin de cumplir una tarea particular.

- Ontologías de aplicaciones: son dependientes de las aplicaciones. A menudo extienden y especializan vocabulario de una ontología de dominio o de tareas para una aplicación particular.

2.5. Componentes de una Ontología

Los componentes de una ontología varían de acuerdo al dominio de interés y a las necesidades de los desarrolladores. Por lo general entre los componentes se encuentran los siguientes (Corcho, O., Fernandez-Lopez, M., Gomez-Perez, A., 2003):

- Clases: son las ideas básicas que se intentan formalizar, describen los conceptos del dominio. Estarán enfocados al área concreta que estudie la ontología en cuestión. Son las unidades básicas para la especificación, ya que proveen una base para la descripción de la información. Las clases tienen propiedades o atributos asociados.
- Relaciones: representan las interacciones entre los conceptos del dominio. Se utilizan para representar correspondencias entre diferentes conceptos y para proveer una estructura general a la ontología. Representan la interacción y el enlace entre conceptos del dominio. Pueden ser de los siguientes tipos (Gómez-Pérez, A. Moreno, A., Pazos, J., Sierra-Alonso, A., 2000):
    - Función: son un tipo concreto de relación, donde se identifica un elemento mediante el cálculo de una función que considera varios elementos de la ontología.
    - Taxonomía: taxonomía es la ciencia que estudia la división en grupos ordenados o categorías. Desde un punto de vista ontológico, en una relación de orden parcial llamada *es-un* (inclusión), a través de la cual se agrupan las entidades.
    - Mereología: es un término griego que significa *estudio de las partes*. La mereología se ocupa de las relaciones entre partes, tanto de las partes con el todo, como de las partes con otras partes.
    - Relación de equivalencia: establece equivalencia entre dos expresiones aparentemente diferentes. Dos clases serán equivalentes cuando tengan las mismas instancias, aunque no necesariamente tienen que ser equivalentes, ya que las mismas instancias no tienen porqué representar el mismo concepto.
    - Relación de dependencia: la relación de dependencia es un tipo especial de relación de asociación a través de distintos atributos y una participación.
    - Relación topológica: la relación topológica describe la distribución espacial de conceptos físicos e interconexiones entre esos conceptos.
    - Relación causal: este tipo de relación describe como dados unos estados o acciones se induce a otros estados o acciones.
    - Relación funcional: la relación funcional describe las condiciones para las acciones y reacciones que tienen lugar y las posibles consecuencias de las acciones.
    - Relación cronológica: esta relación se conoce también como relación temporal y describe la secuencia de tiempo en la que ocurren eventos.
    - Relación de similitud: establece que conceptos son iguales o análogos y en qué medida.
    - Relación condicional: la relación condicional define las condiciones en las cuales ciertas cosas tienen lugar.
    - Relación de propósito: esta relación establece el porqué y para qué de los conceptos.

- Instancias: utilizadas para representar objetos determinados de un concepto.
- Axiomas: teoremas que se declaran mediante relaciones que deben cumplir los elementos de la ontología. Se usan para modelar sentencias que son siempre ciertas y verificar así la consistencia de la ontología. Los axiomas definidos en una ontología pueden ser estructurales o no estructurales: un axioma estructural establece condiciones relacionadas con la jerarquía de la ontología, conceptos y atributos definidos; un axioma no estructural establece relaciones entre atributos de un concepto y son específicos de un dominio.

2.6. Herramientas y Lenguajes Ontológicos

Son variadas las herramientas relacionadas con las ontologías. Desde las iniciales Ontolingua Server, Ontosaurus, a las más recientes Protégé 2000, WebODE, y Ontoedit. Gómez-Pérez, A. (2002) ofrece la siguiente tipología:

A. Herramientas de desarrollo de ontologías: este grupo incluye las herramientas que sirven para la construcción de nuevas ontologías o bien para la reutilización de las existentes. Destacan entre sus funcionalidades la edición y la consulta, así como la exportación e importación de ontologías, la visualización en diversos formatos gráficos, etc. Destacan: Protégé, KAON, OilEd, OntoSaurus, SWOOP, Ontolingua Server,Topbraid Composer y WebOnto.
B. Herramientas para la fusión y de la integración de las ontologías: pretenden solucionar el problema de la combinación y la integración de diversas ontologías del mismo dominio, lo que ocurre cuando se unen dos organizaciones diferenciadas, o cuando se pretende obtener una ontología de calidad, a partir de las ya existentes. Destacan: RDFSuite, Sesame, Jena, KAON API.
C. Herramientas de evaluación de ontologías: aparecen como instrumentos de apoyo que deben asegurar que tanto las ontologías como las tecnologías relacionadas tengan un nivel mínimo de calidad. Este esfuerzo pudo también conducir a las certificaciones estandarizadas.
D. Herramientas basadas de la anotación: estas herramientas se han diseñado para permitir a usuarios que inserten informaciones y datos. La mayoría de estas herramientas han aparecido recientemente, junto con la aparición de la idea de Web semántica. AeroDAML, OntoAnnotate, SHOE Knowledge Annotator.
E. Herramientas de almacenaje y de preguntas: son instrumentos que se han creado para permitir usar fácilmente las ontologías. La clave está en el intento de que la Web se convierta en una auténtica plataforma para transmitir conocimiento.
F. Herramientas de aprendizaje: se utilizan semi-automáticamente para construir ontologías a partir de la lengua natural. ASIUM, DODDLE, OntoBuilder, Racer, OntoLearn, y Text-To-Onto.

Los últimos desarrollos en cuanto a herramientas ontológicas se refiere, se orientan a enfocar ontologías a sistemas de información existentes. Por lo general son independientes del lenguaje de implementación de la ontología, lo que los diferencia de los entornos primitivos. La más usada actualmente es Protégé-2000. Fue desarrollado por el Stanford Medical Informatics (SMI) de la universidad de Stanford. Es un proyecto Java de fuente libre que provee una arquitectura extensible para la creación de herramientas de bases de conocimiento. Es una herramienta muy versátil porque permite al usuario construir el dominio de una ontología, configurar formularios para almacenamiento de datos e ingresar el dominio de conocimiento. Es una

plataforma que puede ser extendida con gráficos, diagramas, componentes animados para acceder a aplicaciones embebidas en sistemas de bases de conocimientos. Es también una librería a la que otras aplicaciones pueden acceder.

Una vez que los componentes de la ontología están definidos, la ontología puede ser representada mediante varios lenguajes. Estos lenguajes comenzaron a surgir a comienzos de 1990 y se basan principalmente en lógicas de primer orden, en marcos (*frames*) combinados con lógicas de primer orden y lógicas descriptivas (DL). A continuación se listan algunos de ellos:

A. RDF (Resource Description Framework). Es un lenguaje para especificar metadatos, basado en XML y recogido en las recomendaciones W3C. En este lenguaje las relaciones entre dos objetos se establecen mediante el nombre de la relación y dichos elementos, similar a las redes semánticas.
B. OWL. Lenguaje basado en XML y RDF, pertenece a la W3C y es el estándar actual. Puede representar los elementos de lógica descriptiva. Además tiene mayor capacidad expresiva. Es el más usado en Internet, estando sus elementos definidos con las fuentes de RDF. Tiene tres variantes según la complejidad que se necesite especificar, se muestran en orden ascendente de complejidad: OWL Lite, OWL DL y OWL Full.
C. CycL (D. B. Lenat, R. V. Guha, 1990). Desarrollado por Doug Lenat Cyc en un proyecto basado en la inteligencia artificial, por ello es más bien un lenguaje declarativo basado en lógica de predicados de primer orden, añadiendo las extensiones para operadores, es un sistema de código abierto, y sus principales características son la utilización de constantes para representar conceptos y representación de jerarquía, así como las normas en las que se apoyan la relaciones entre conceptos.
D. DOGMA. Proyecto desarrollado en Vrije Universiteit Brussel Starlab, cuyo principal objetivo es solventar el problema lingüístico de las ontologías, adaptándolas a una independencia del idioma. Para ello realiza una separación del dominio con respecto a la conceptualización de su aplicación. Podríamos considerar este lenguaje una mezcla entre RDF y OWL, teniendo en cuenta la diferenciación que hace con el nivel conceptual y el nivel del idioma.
E. SPARQL (SPARQL Protocol and RDF Query Language). Este lenguaje derivado de RDF permite consultas basadas en tres modelos: conjunciones, disyunciones, y, patrones. Permite la definición de prefijos y se pueden realizan consultas muy específicas. Su principal función es su utilización como lenguaje de consulta en la Web semántica.
F. KIF. Es un lenguaje basado en lógica de primer orden y se creó como formato de intercambio para diversos sistemas de representación de conocimiento. Es el más expresivo de los lenguajes usados para representar ontologías, permitiendo representar conceptos, taxonomías de conceptos, relaciones n-arias, funciones, axiomas, instancias y procedimientos. Sin embargo el lenguaje en sí, no provee soporte para razonamiento automático (M. Genesereth, R. Fikes, 1992).
G. Ontolingua. Es un sistema para describir ontologías de manera compatible con múltiples sistemas de representación. Provee formas para definir clases, relaciones, funciones, objetos y teorías. Las ontologías escritas en Ontolingua pueden ser compartidas por varios grupos de usuarios. La sintaxis y semántica de las definiciones de Ontolingua están basados en KIF y traduce las definiciones a distintos sistemas de representación implementados (T. R. Gruber, 1993).
H. FLogic (*Frame Logic*). Combina marcos y lógica de primer orden. Permite representar conceptos, taxonomías de conceptos, relaciones binarias, funciones, axiomas y reglas deductivas. Se diferencia con los anteriores en que es el único que no tiene una sintaxis similar al lenguaje Lisp. Provee un mecanismo de

inferencia (Ontobroker) que puede ser usado para verificación de restricciones y deducción de información nueva (M. Kifer, G. Lausen, J. Wu, 1995)

I. OCML: fue construido para desarrollar ontologías ejecutables y modelos en métodos de resolución de problemas. A las posibilidades de KIF se le agregan reglas de producción y deducción, y definiciones operacionales de funciones (E. Motta, 1999).

Al implementar una ontología es importante decidir primero las necesidades en términos de expresividad y servicios de inferencia, porque no todos los lenguajes permiten representar los mismos componentes de la misma forma. La representación y razonamiento con información básica, como conceptos, taxonomías, y relaciones binarias, no siempre es suficiente si se quiere crear una ontología pesada y hacer razonamientos complejos. Es frecuente que las traducciones entre lenguajes no sea lo suficientemente precisa por lo que puede perderse información en el proceso de traducción. Por eso, la decisión del uso de un lenguaje específico para representar una ontología, juega un papel crucial (Corcho, O., Fernandez-Lopez, M., Gomez-Perez, A., 2003).

2.7. Construcción de una Ontología

T. R. Gruber (1993) detalla un conjunto de principios y criterios de diseño que suelen ser útiles en el desarrollo de ontologías. Se citan a continuación los más relevantes:

- Claridad y objetividad: la ontología debe proveer el significado de los términos definidos proveyendo definiciones objetivas y documentación en lenguaje natural.
- Completitud: una definición expresada en términos de condición necesaria y suficiente es preferible sobre una definición parcial (definida sólo sobre condición necesaria o suficiente).
- Coherencia: permite hacer inferencias válidas consistentes con las definiciones.
- Máxima extensibilidad: los nuevos términos (generales o especializados) debieran ser incluidos en la ontología de manera que no requiera la revisión de definiciones existentes.
- Mínimo compromiso ontológico: es un criterio inherente a la dimensión estructural, por el cual se pretende suprimir las suposiciones acerca del mundo modelado y reducir al mínimo los casos especiales. De esta forma, el compromiso ontológico representa los acuerdos para usar el vocabulario compartido de una manera coherente y consistente.
- Otros principios son: estandarización de nombres (tanto como sea posible), modularidad (para minimizar el acoplamiento entre módulos) y el hecho de que las clases en una ontología debieran ser disjuntas.

El proceso de construcción de una ontología permite convertir el conocimiento tácito que poseen los integrantes de la organización o dominio, en conocimiento explícito y representable. El modelo de conocimiento se puede editar y gestionar, pero también se puede transmitir, de manera que un sistema entienda la conceptualización que se ha utilizado en el otro. De este modo, la utilidad de una ontología se puede medir en la capacidad de permitir a los sistemas, hacer referencias a otros componentes de conocimientos definidos, siempre que ambos compartan la misma conceptualización. Una ontología compartida solo necesita describir un vocabulario común para hablar sobre un dominio. Hay varios métodos para desarrollar ontologías, en función del desarrollador que se considere y el uso que se le vaya a dar, se pueden identificar unas u otras fases (en el apéndice A se muestra un cuadro con los principales métodos por desarrollador).

También es crucial el uso que se le vaya a dar a la ontología a la hora de decantarse por uno u otro método, ya que la elaboración y construcción de una ontología debe tener en cuenta su relación con la arquitectura del sistema de información en el que está inmersa. Sin embargo, en términos generales los pasos a seguir en la creación de una ontología son los siguientes (Martínez Comeche, J. A.):

A. Definir el dominio y alcance de la ontología: recopilar la información necesaria, esto se puede hacer mediante preguntas relevantes y de verificación. Se debe definir el uso de la aplicación final, qué dominio cubrirá, qué preguntas debería contestar la ontología, y quién usará y mantendrá la ontología.
B. Considerar la reutilización de ontologías existentes. Conviene comprobar si se puede partir de alguna ontología existente, o incluso reutilizar algunos términos de otras ontologías.
C. Definir las clases y las jerarquías de clases. Definir la estructura conceptual del dominio, siguiendo el método *top-down*, primero se crean las clases para los conceptos generales en el dominio y su especificación subsiguiente. Se puede usar un diagrama para representar la jerarquía.
D. Definir las propiedades o slots de las clases. Las clases por si solas no proveen suficiente información para responder a las peticiones. Una vez definidas las clases, se deberá describir su estructura y características. En general, se trata de identificar las características de los objetos (intrínsecas, extrínsecas, partes del objeto, relaciones con otras clases, etc.).
E. Definir las facetas o restricciones de las propiedades o slots. Especificar los valores permitidos, el tipo de valores que tomarán los slots, el número de valores, etc.
F. Creación de las instancias individuales de las clases en la jerarquía. Para ello habrá que elegir la clase, crear la instancia individual para esa clase y rellenar los valores de las propiedades.
G. Formalizar el modelo obtenido a través de un lenguaje de representación.

Sin embargo, aunque muy útil y extendido, el reúso ontológico puede llevar asociados los siguientes problemas:

- Paradoja lingüística: al adaptar ontologías de un lenguaje a otro puede resultar que no siempre el concepto está representado como entidad en un término, o el término es confuso para representar el conocimiento social referente a esa materia de la que es objeto la realización de la ontología.
- El dinamismo de los conceptos: este problema surge al ser los conceptos entidades dinámicas que evolucionan con el tiempo, esto requiere una continua actualización de la ontología

La forma de paliar estos problemas podría ser revisiones de la ontología, así como tener en cuenta la polisemia, ambigüedad, tratamiento de conceptos difusos y contemplar todos los casos conflictivos que puedan surgir.

2.8. Conclusión

La principal utilidad de una ontología es ayudar a la compartición del conocimiento entre diferentes actores de un dominio determinado, como pueden ser personas, organizaciones o sistemas software. La disponibilidad del conocimiento almacenado en ontologías puede proveer los mecanismos necesarios para organizar, almacenar y

acceder a la información que incluyen esquemas de bases de datos, objetos de interfaz de usuario, o programas de aplicación.

También es importante destacar que no hay una única ontología correcta para un dominio dado. La corrección de una ontología va a venir dada en la medida que cumpla con el objetivo de obtener el conocimiento deseado.

En el caso que nos ocupa, a la hora de emplear las técnicas de planificación podemos encontrarnos con distintos dominios muy extensos en conocimiento, difíciles de modelar y con gran cantidad asociada. En este tipo de dominios surgen nuevos problemas relacionados con la adquisición y gestión de grandes bases de conocimiento. Por otro lado, las ontologías proporcionan un vocabulario rico con el que poder expresar conocimiento usando distintos niveles de detalle. Esta capacidad de razonamiento abstracto permite realizar inferencias interesantes a partir del conocimiento disponible.

Hay que tener en cuenta también que estas ontologías son uno de los pilares fundamentales de la web semántica, por lo que existe toda una infraestructura montada a su alrededor de la que se pueden aprovechar futuros desarrollos.

3. LÓGICA TEMPORAL

3.1. Justificación Uso Lógica Temporal

La lógica temporal ha sido utilizada como sistema de representación para definir semántica de expresiones temporales en los más diversos dominios; en el presente trabajo se presenta como herramienta para expresar aspectos temporales en la planificación de tareas. La lógica temporal extiende a la lógica clásica permitiendo especificar en qué momento del tiempo está teniendo lugar un hecho o proposición, dato fundamental en el caso que nos ocupa a la hora de programar la carga lectiva de una asignatura, las pruebas de evaluación continua, las prácticas y los exámenes. También resulta de gran utilidad para la verificación del modelo resultante; existen técnicas y herramientas para asegurar el correcto funcionamiento de sistemas críticos, y la mayoría de estas herramientas se basan en la lógica temporal. De este modo, establece si el sistema está construido de forma correcta (Boehm, 1984).

3.2. Flujo de Tiempo

Antes de hablar de lógica temporal, es conveniente hacer una introducción de las distintas concepciones en la estructura del tiempo, puesto que según el flujo de tiempo que se considere, se podrá aplicar uno u otro tipo de lógica. (Casas Joan, 2013)

A. Estructura lineal: representada por una línea formada por varios puntos (instantes de tiempo) unidos por una relación de orden. El pasado y el futuro se representan en este caso como una línea determinada, sin caminos o ramas alternativas posibles. La relación de orden tiene que cumplir las siguientes propiedades: es irreflexiva, transitiva, con pasado y futuro infinitos y fuertemente conectada. Puede ser denso o discreto.

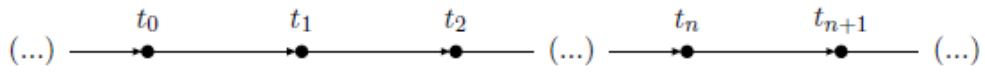


Ilustración 7. Tiempo Lineal

B. Estructura cíclica: el tiempo se representa como un ciclo cerrado en el que, dado un punto cualquiera considerado como el punto inicial, se llega de nuevo a él siguiendo avanzando hacia el futuro. En este caso, las propiedades de irreflexibilidad y transitividad son incompatibles simultáneamente. Es decir, la relación de orden será irreflexiva o transitiva, pero no las dos a la vez. En este tipo de estructura, no tiene sentido instante inicial y final, aunque por convención si existen. Tanto el pasado como el futuro son infinitos

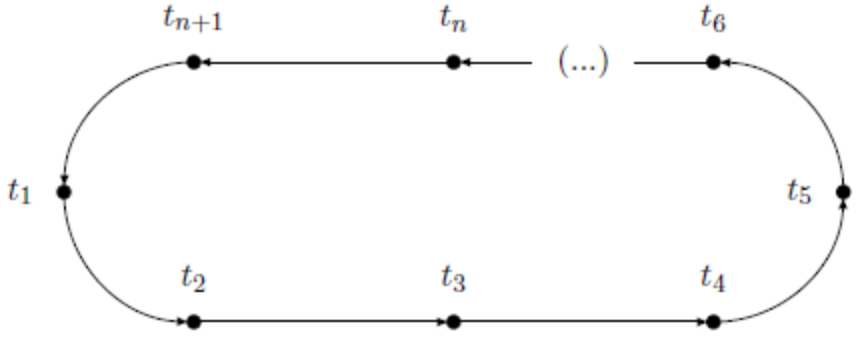


Ilustración 8. Tiempo Cíclico

C. Estructura ramificada: el tiempo se concibe en este caso como una estructura en la que en determinados puntos se abren diferentes caminos que siguen diferentes líneas temporales. Esta concepción temporal ofrece la posibilidad de contemplar conceptos como "futuro contingente" o "futuro posible". Los caminos se bifurcan ante esos futuros posibles. El pasado se considera determinado e imposible de cambiar.

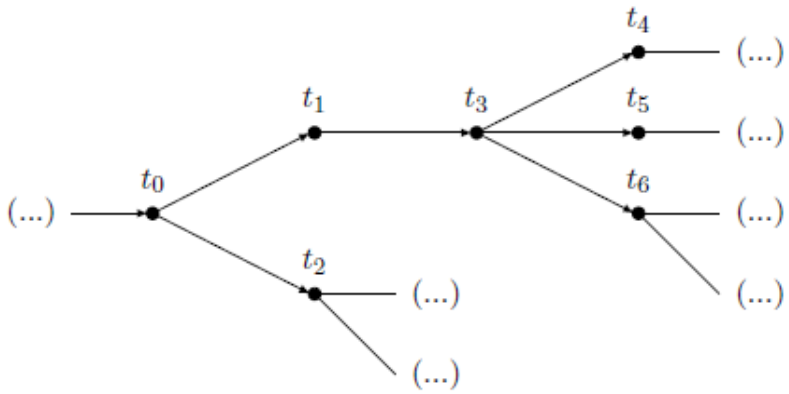


Ilustración 9. Tiempo Ramificado

3.3. Antecedentes de la Lógica Temporal

Un sistema lógico está compuesto por:

- Un Alfabeto: conjunto de símbolos primitivos
- Una Gramática: conjunto de reglas de formación que nos dice cómo construir fórmulas bien formadas a partir de los símbolos primitivos.
- Un Conjunto de Axiomas. Cada axioma debe ser una fórmula bien formada.
- Un Conjunto de Reglas de Inferencia. Estas reglas determinan qué fórmulas pueden inferirse de qué formulas.
- Una Interpretación Formal. Las interpretaciones formales asignan significados inequívocos a los símbolos, y valores de verdad a las fórmulas

Durante décadas, los filósofos clásicos se han preguntado tanto por la naturaleza del tiempo (¿es una entidad real -de naturaleza física- o ideal -un concepto matemático o de algún otro tipo-?), como por su estructura o forma (¿es finito o infinito, tiene o no dimensiones, etc.?) (Van Fraassen, B. C., 1978).

La noción temporal se introdujo en el campo de la lógica a partir de la segunda mitad del siglo XX; hasta entonces el contexto tiempo era irrelevante. Fue entonces cuando se consideró como un elemento básico para la argumentación y el razonamiento. De este modo, se pasó a realizar razonamiento sin obviar consideraciones temporales, según las cuales, un razonamiento tenía uno u otro significado.
Posteriormente se extendió el uso de la unidad temporal a otros ámbitos matemáticos como la teoría de la demostración, propiedades matemáticas, y posteriormente a la ciencia de la computación.

La lógica temporal es una extensión de la lógica clásica para permitir la formalización de enunciados que incluyan precisiones acerca del momento del tiempo en que han tenido lugar. En particular, las lógicas temporales derivan de las lógicas modales.

El propósito de la lógica temporal es diferente según el punto de vista de la ciencia que lo trate: desde el punto de vista filosófico, nos permite razonar sobre el tiempo. Sin embargo desde el punto de vista de la computación, la lógica temporal aborda el comportamiento de los sistemas que evolucionan en un lapso de tiempo.

Nuestro interés con respecto a la lógica temporal radica en su potencialidad para convertir conocimiento implícito en conocimiento explícito. Para ello ofrece las siguientes herramientas: (Huertas, M.A., 2006)

A. Lenguaje formal con el que expresar "conocimiento" con rigor y precisión.
B. Semánticas formales con las que precisar el significado de "verdad" o "validez" de un razonamiento.
C. Razonadores automáticos que pueden deducir y convertir el conocimiento "implícito" en "explícito".
D. Pruebas de los razonamientos en un sistema de inferencia (deductivo) que proporcionan "explicaciones" cuando son suficientemente cercanos a la inferencia humana (como por ejemplo los sistemas de deducción natural).

Para cada uno de estos factores tenemos una lógica que los implementa: lógicas de proposiciones y de primer orden, de orden superior, bivaloradas, trivaloradas y multivaloradas, fuzzy, modales, dinámicas, temporales, intuicionistas, cuánticas, etc.

Una característica importante de la lógica es su capacidad para recoger o expresar todos aquellos aspectos del sistema cuya modelación es conveniente. Es lo que se denomina expresividad de la lógica. Sin embargo, esta característica tan deseable en la lógica dificulta los algoritmos, debido a que cuanto más expresiva es la lógica más ineficientes se vuelven los algoritmos.

Otro aspecto a considerar es la finitud de los estados en la verificación automática, ciencia en la que la lógica temporal es ampliamente utilizada. La lógica temporal posibilita la existencia de múltiples estados en un modelo, y la transición de uno a otro. Sin embargo, la verificación automática es posible sólo si el modelo a verificar es finito, y el número de estados no lo convierte en computacionalmente intratable. Cuando estas propiedades no se cumplen, es necesario atacar el problema con herramientas más expresivas, tales como un sistema deductivo.

3.4. Clasificación de la Lógica Temporal

3.4.1. Lógica temporal proposicional minimal K_t^+

El subíndice t denota que se trata de una lógica temporal, mientras que el superíndice + indica que solo contempla el tiempo futuro. El alfabeto, aK_t^+, de K_t^+ se obtiene añadiendo los símbolos de conectivas temporales G y F al alfabeto aprop de un lenguaje de la lógica clásica proposicional, es decir, aLM = aprop U {G, F}.

Lo lógica temporal proposicional minimal está compuesta por:

- Alfabeto:
  - Variables proposicionales: Vprop = {p, q, r,…, p_1, q_1, r_1,… p_n, q_n, r_n,….}
  - Constantes: T (verdad), ⊥ (falsedad)
  - Conectivas booleanas : $\wedge, \vee, \rightarrow, \leftrightarrow, \neg$
  - Conectivas temporales: G (siempre en el futuro) y F (alguna vez en el futuro)
  - Símbolos de puntuación: "(" y ")"

- Fórmulas bien formadas de K_t^+ (fbfs);
  - T y ⊥ son fbfs llamadas fbfs atómicas
  - Si A es una fbfs, entonces ¬A, GA y FA son fbfs
  - Si A y B son fbfs, entonces (A ∧ B), (A ∨ B), (A→B) Y (A↔B) son fbfs
  - Solo las cadenas obtenidas aplicando las reglas anteriores son fbfs

- Árbol sintáctico: el árbol sintáctico para una fbfs A de K_t^+, denotado por T_A es el aK_t^+ árbol definido recursivamente como sigue:
  - T_A es , si A ∈ Vprop U {T y ⊥**}**
  - T_{+A} donde * ∈ {¬, G, F} es * —— T_A
  - T_{A+B} donde * ∈ {∧, ∨, →, ↔, ¬ }, es * < T_A / T_B

- Semántica de K_t^+:un modelo para K_t^+ es una terna M = (T, R_T, h) donde T es el flujo temporal, R_T una relación de precedencia y h una función de evaluación que satisface las siguientes propiedades:
  - h(T) = T, y h(⊥) = ∅
  - $h(\neg A) = h(A)^c$
  - $h(A \wedge B) = h(A) \cap h(B)$
  - $h(A \vee B) = h(A) \cup h(B)$
  - $h(A \rightarrow B) = h(A)^c \cup h(B)$
  - $h(A \leftrightarrow B) = h(A \rightarrow B) \cup h(B \rightarrow A)$
  - $h(GA) = \{t \in T \mid R_T(t) \subseteq h(A)\}$
  - $H(FA) = \{t \in T \mid R_T(t) \cap h(A) \neq \emptyset\}$

- Leyes en K_t^+
  - Negación: GA ≡ ¬G¬A; FA ≡ ¬F¬A; ¬GA ≡ F¬A; ¬FA ≡ G¬A
  - Todas las leyes de la lógica clásica proposicional
  - GT ≡ T; F⊥ ≡ ⊥
  - ⊨ FA v G¬A
  - G(A∧B) ≡ GA ∧ GB
  - ⊨ (GA v GB) →G(AvB)
  - F(AvB) ≡ FA v FB
  - ⊨ F(A∧B) → (FA ∧ FB)
  - ⊨ G(A→B) → (GA → GB)
  - ⊨ G(A↔B) → (GA ↔GB)

3.4.2. Lógica temporal proposicional minimal K_t

Lógica bimodal que considera las conectivas temporales de pasado y futuro pero no tiene en cuenta ninguna condición sobre la estructura del tiempo (sobre las propiedades de la relación temporal R_T).

- El alfabeto de K_t se obtiene añadiendo al alfabeto de K_t^+ los símbolos de conectivas temporales siguientes :
  - HA: A en el pasado siempre fue verdadera
  - PA: A en el pasado alguna vez fue verdadera

- Las fbfs de de K_t se obtienen con las mismas reglas de formación que en K_t^+, junto con la siguiente:
  - Si A es una fbf, HA y PA son fbfs

- Semántica de K_t: un modelo temporal para K_T es una terna M = (T, R, h) donde R = {R_T, R_T^{-1}} y la función de evaluación h: $K_t \rightarrow 2^T$ satisface las mismas condiciones que en K_t, junto con las siguientes:
  - H(PA) = {t ϵ T | R_T^{-1}(t) ∩ h(A) ≠ ∅ }
  - H(HA) = {t ϵ T | R_T^{-1}(t) ⊆ h(A)}

- Axiomas necesarios para que la lógica K_t sea correcta y completa;
  - Los de K_t^+
  - H(A→B) → (HA→HB)
  - A → HFA y PGA → A
  - A → GPA y FHA → A

3.4.3. Lógica temporal proposicional K_t^{Par}

Esta lógica, además de incorporar las conectivas de pasado y futuro, considera a R_T como una relación de orden transitiva y asimétrica. De este modo, se incorporan las conectivas < (relación de orden parcial estricto) y ≤ (relación de orden parcial)

- Si (X, <) es un orden parcial estricto y a, b ϵ X, entonces
  - a ≤ b denota que a<b o bien a=b
  - a ‖ b denota que a ≠ b y a y b no están relacionados (a≮b y b≮a)

- Si (X, <) es un orden parcial estricto, e Y ⊆ X / Y ≠ ∅
  - b ϵ Y es el mínimo si b ≤ y, ∀y ϵ Y
  - b ϵ Y es el máximo si y ≤ b, ∀y ϵ Y
  - (X, <) es un buen orden si todo Y subconjunto de X tiene primer elemento
  - a ϵ X es una cota superior de Y si y ≤ a, ∀y ϵ Y. Cot^+(Y) es el conjunto de todas las cotas superiores de Y. Si Cot^+(Y) ≠ ∅ diremos que Y está acotado superiormente
  - a ϵ Si Cot^+(Y) es el extremo superior de Y (a = sup Y) si ∀y' cota superior de Y, si se cumple que a≤'a
  - a ϵ X es una cota superior de Y si a ≤ y, ∀y ϵ Y. Cot^-(Y) es el conjunto de todas las cotas inferiores de Y. Si Cot^-(Y) ≠ ∅ diremos que Y está acotado inferiormente
  - a ϵ Si Cot^-(Y) es el extremo inferior de Y (a = inf Y) si ∀y' cota inferior de Y, si se cumple que a'≤a
  - Y se dice acotado si está acotado superior e inferiormente
  - Un retículo es un conjunto parcialmente ordenado (X, ≤) | ∀ x, y ϵ X, ∃ sup{x, y} e inf{x, y}

- Intervalos: dado un orden parcial (X, <), con a ϵ X, entonces:
  - [a, →) = {x ϵ A | a ≤ x} y (→,a] = {x ϵ A | a ≥ x}
  - (a, →) = {x ϵ A | a ≤ x} y (→,a) = {x ϵ A | a ≥ x}
  - [a, b) = {x ϵ A | a ≤ x <b} y (a,b] = {x ϵ A | a < x ≤ b}
  - [a, b] = {x ϵ A | a ≤ x ≤b} y (a,b) = {x ϵ A | a < x < b}

- Dotada de 2 esquemas fbfs que recogen la transitividad de <:
  - (Par_1) GA → GGA
  - (Par_2) HA → HHA

3.4.4. Lógica Kl con tiempo lineal y sus extensiones

Se trata de una lógica de orden lineal que cumple con la propiedad de tricotomía: $\forall t, t' \epsilon\, T$ se tiene que t<t' o t=t' o t'<t Esta lógica incorpora por tanto instantes pasados y futuros.

- Propiedades: dado un flujo de tiempo (T, <) con orden lineal, y t,t',t''$\epsilon\, T$:
  - Lineal hacia el futuro: si t<t' y t<t'' → t'<t'' o t'=t'' o t''<t'
  - Lineal hace el pasado: si t'<t y t''<t → t'<t'' o t'=t'' o t''<t'
  - Ramificado hacia el futuro: $\exists t_1, t_2\, \epsilon$ (t,→) tales que $t_1 || t_2$
  - Ramificado hacia el pasado: $\exists t_1, t_2\, \epsilon$ (→,t) tales que $t_1 || t_2$
  - No ramificado: no es ramificado hacia el futuro ni hacia el pasado

- Axiomas necesarios para que el sistema sea correcto y completo:
  - $\vdash GA \rightarrow GGA$
  - $\vdash HA \rightarrow HHA$
  - Linealidad hacia el futuro
    - $\vdash \big(G(A \vee B) \wedge G(A \vee GB) \wedge G(GA \vee B)\big) \rightarrow (GA \vee GB)$
    - $\vdash (FA \wedge FB) \rightarrow \big(F(A \wedge B) \vee F(A \wedge FB) \vee F(FA \wedge B)\big)$
  - Linealidad hacia el pasado
    - $\vdash \big(H(A \vee B) \wedge H(A \vee GB) \wedge H(HA \vee B)\big) \rightarrow (HA \vee HB)$
    - $\vdash (PA \wedge PB) \rightarrow \big(P(A \wedge B) \vee P(A \wedge PB) \vee P(PA \wedge B)\big)$

- Primer y último elemento
  - Si existe último momento t_∞ , < es letal, y se tienen que cumplir los axiomas
    - Max = $GA \vee FGA$
    - Max'= $G \perp \vee FG \perp$
  - Si ∄ último momento, < es serial y NMax = $GA \rightarrow FA$
    - Tiempo infinito por la derecha: NMax U $K_l \equiv K_l^{+\infty}$
  - Si existe primer momento $t_{-\infty}$, > es letal, y se tienen que cumplir los axiomas
    - Min = $HA \vee PHA$
    - Min'= $H \perp \vee PH \perp$
  - Si ∄ primer momento, > es serial y NMin = $HA \rightarrow PA$
    - Tiempo infinito por la izquierda: NMin U $K_l \equiv K_l^{-\infty}$
  - Tiempo lineal infinito: NMax U NMin U $K_l \equiv K_l^{\pm\infty}$

- Tiempo discreto. K_l es completo respecto a las estructuras temporales lineales discretas si cumple los siguientes axiomas:
  - Si t no es el último elemento de T, entonces $\exists t' \in T |\, t < t' y\, (t, t') = \emptyset$
    - $Dis1 \equiv A \cap HA \rightarrow (G \perp \vee FHA)$
  - Si t no es el primer elemento de T, entonces $\exists t' \in T |\, t' < t\, y\, (t, t') = \emptyset$
    - $Dis2 \equiv A \cap GA \rightarrow (H \perp PGA)$

- Axioma de Löb: define una relación que es transitiva y a la vez buen orden: $Bf \equiv H(HA \rightarrow A) \rightarrow HA$
- La relación < es conexa si, $\forall t, t' \in T, \exists t_1, \dots t_k \in T$ tal que:
  1. $t_1 = t$
  2. $t_k = t'$ y
  3. $t_i<t_{i+1}$ o $t_{i+1}<t_i$ para cada i ($1 \leq i<k$)
  ➔ $Conex1 \equiv G(GA \rightarrow A) \rightarrow (FGA \rightarrow GA)$
  ➔ $Conex2 \equiv H(HA \rightarrow A) \rightarrow (PHA \rightarrow HA)$

- Densidad del tiempo: una relación es densa si entre cada dos momentos existe otro momento $\forall t_1, t_2 \in T \,|t_1 < t_2, \exists t_3 \in T \,|t_1 < t_2 < t_3$. La densidad del flujo del tiempo se puede recoger en los siguientes axiomas:
  - $Den1 \equiv GGA \rightarrow GA$ o equivalentemente $FA \rightarrow FFA$
  - $Den2 \equiv HHA \rightarrow HA$ o equivalentemente $PA \rightarrow PPA$
    ➔ Sistema de tiempo racional: Den1 U Den2 U $K_l^{\pm\infty} \equiv K_{ld}^{\pm\infty}$

- Continuidad del tiempo: una relación de orden lineal estricto es completa si cumple que todo subconjunto no vacío y superiormente acotado de T tiene extremo superior y, similarmente, todo subconjunto no vacío e inferiormente acotado de T tiene extremo inferior. Axiomas que recogen esta propiedad:
  - $Comp1 \equiv (FA \wedge FG\neg A) \rightarrow F(HFA \wedge G\neg A)$
  - $Comp2 \equiv (PA \wedge PH\neg A) \rightarrow P(GPA \wedge H\neg A)$
    → Sistema correcto y completo respecto de estructuras temporales continuas: Comp1 U Comp2 U $K_{ld}^{\pm\infty} \equiv K_R^{14}$

3.4.5. Lógica FNext

Se trata de una lógica de tiempo lineal infinito y discreto que utiliza únicamente conectivas de futuro. Un modelo temporal para FNext es una terna M = (Z,<,h) donde h es una aplicación h: FNext → 2^Z denominada interpretación temporal, y asocia a cada $p \epsilon V_{prop}$, el conjunto de instantes en los que es verdadera. Este modelo permite contemplar el paso del tiempo como una transición de un estado a otro, con interpretaciones diferentes para cada estado en un instante dado.

- Alfabeto (a_{FNext}) consta de los mismos elementos que la lógica $\boldsymbol{K_t^+}$ más la conectiva temporal $\oplus$

- Fórmulas bien formadas (fbfs) de Fnext
  - $\perp$, T y toda $p \in V_{prop}$ son fbfs
  - Si A es fbf, →$\neg A, FA, GA\ y \oplus A$ son fbfs
  - Si A y B son fbfs → (A*B) donde * ∈ {∧, ∨, →, ↔} es una fbfs
  - Solo las cadenas obtenidas aplicando las reglas anteriores son fbfs
  - $\oplus^k A = \oplus\oplus^{k-1} A\ si\ k \geq 1\ y \oplus^0 A = A$

- Propiedades
  - Seguridad: nunca ocurrirá algo malo en el sistema
    - $p \rightarrow Gq$

- $G(\neg q \vee \neg p)$
- Viveza: algo bueno ocurrirá eventualmente en el sistema
  - $p \rightarrow Fq$
  - $GFp \rightarrow Fq$
  - $Gp \rightarrow Fq$
- Una fórmula A ϵ FNext se dice satisfacible si ∃ M = (Z,<,h) y un instante $t\epsilon Z \mid t\epsilon h(A)$. Se denota por M,t ⊨A
- Una fórmula se dice válida en el modelo M =(Z,<,h) si h(A)=Z, es decir, M,t ⊨A $\forall t\epsilon Z$. Se denota por $\models_M$
- Una fórmula se dice válida, denotado por ⊨A si $\models_M$ A ∀ M
- A, B ϵ FNext se dicen equivalentes si $\models A \leftrightarrow B$, es decir,

  $\forall\, M = (Z, <, h), h(A) = h(B)$M

- Ω conjunto de fbfs, $\Omega \subset FNext$ se dice satisfacible si ∃ M = (Z,<,h) y un instante t tal que,

  $t\epsilon Z \mid \mathrm{M}, \mathrm{t} \models \mathrm{A}\,, \forall\, A\epsilon\ \Omega$

- Ω conjunto fbfs, y C una fbf C es consecuencia o se deriva semánticamente de Ω, denotado por $\Omega \models C$ si ∀ M = (Z,<,h) y $\forall t\epsilon Z$ se tiene que: si $\mathrm{M}, \mathrm{t} \models \mathrm{A}\,, \forall\, fbf\ A\epsilon\ \Omega$, entonces también se tiene que $\mathrm{M}, \mathrm{t} \models \mathrm{C}$

- Semántica (m_{tA}^{+}): para definir la semántica FNext se considera como flujo temporal (Z, <) (no tiene porqué existir primer instante).

  $$m_{tA}^{+} = \{t'\epsilon Z | t' < t\ y\ A\ es\ verdadero\ en\ t'\}$$

- Axiomas: un modelo FNext debe cumplir las siguientes propiedades:
  - $h(\mathsf{T}) = Z$ $(\perp) = \emptyset$
  - $h(\neg A) = Z\ \setminus h(A)$
  - $h(A \wedge B) = h(A) \cap h(B)$
  - $h(A \vee B) = h(A) \cup h(B)$
  - $h(A \rightarrow B) = \big(Z \setminus h(A)\big) \cup h(B)$
  - $h(A \leftrightarrow B) = h(A \rightarrow B) \cap h(B \rightarrow A)$
  - $h(FA) = \{t\epsilon Z\ |(t, \rightarrow) \cap h(A) \neq \emptyset\} = \{t\epsilon Z\ | m_{tA}^{+} < +\infty\}$
  - $h(GA) = \{t\epsilon Z\ |(t, \rightarrow) \subseteq h(A)\} = \{t\epsilon Z\ | m_{t\neg A}^{+} = +\infty\}$
  - $h(\oplus A) = \{t\epsilon Z\ | m_{tA}^{+} = t + 1\}$

  donde $m_{tA}^{+} = \begin{cases} \min\big((t, \rightarrow) \cap h(A)\big) & si\ (t, \rightarrow) \cap h(A) \neq \emptyset \\ +\infty & en\ otro\ caso \end{cases}$

- Relaciones en FNext
  - Equivalencia (≡): $[A] = \{B\ \epsilon\ FNext\ | h(A) = h(B)\}$
  - Consecuencia (⇒):
    - A⇒B si y solo si $\models A \rightarrow B$
    - A⇒B si y solo si $h(A) \subseteq h(B)$
    - $A \equiv B\ si\ y\ solo\ si\ A \Rightarrow y\ B \Rightarrow A$

- Leyes de FNext: dispone de todas las leyes definidas en la lógica clásica proposicional, más las propias de conectivas temporales:
  - $G(A \wedge B) \equiv GA \wedge GB$
  - $F(A \vee B) \equiv FA \vee FB$
  - $\oplus(A \wedge B) \equiv \oplus A \wedge \oplus B$
  - $\oplus(A \vee B) \equiv \oplus A \vee \oplus B$
  - $\neg\oplus A \equiv \oplus\neg A; \neg FA \equiv G\neg A; \neg GA \equiv F\neg A$
  - $FFA \equiv \oplus FA \equiv F\oplus A; GGA \equiv \oplus GA \equiv G\oplus A$
  - $\gamma FGA \equiv FGA \; y \; \gamma GFA \equiv GFA$ siendo γsecuencia finita de elementos en $\{\oplus, F, G\}$

- Literales: las fbfs más simples en FNext son aquellas cuyo árbol sintáctico es un árbol rama:

  FNext^{un} = $\{\gamma_1, \ldots, \gamma_1\emptyset \mid \emptyset \epsilon V_{prop} \cup \{\bot,\top\} y \;\; \gamma_i \epsilon \{\neg, F, G, \oplus\} \; \forall 0 \leq i \leq n\}$

3.4.6. Lógica LN

Lógica modal temporal de puntos sobre tiempo lineal y discreto, con conectivas de pasado y de futuro.

- Dos conectivas temporales primitivas recogen los conceptos de precedencia y posterioridad:
  - $A \preccurlyeq B$: En el futuro se dará A, y la próxima aparición de A será anterior o simultánea con la próxima aparición de B.
  - $A \succcurlyeq B$:En el pasado se dio A, y la última aparición de A fue posterior o simultánea con la última aparición de B

- Sintaxis: el alfabeto de LN consta de:
  - V_{prop} = {p, q, r, …}
  - Conectivas y constantes booleanas: $\wedge, \vee, \rightarrow, \leftrightarrow, \neg, \bot, \top$
  - Conectivas temporales binarias: $\preccurlyeq, \succcurlyeq$. Estas conectivas tienen prioridad respecto a las conectivas booleanas binarias.

- Inducción de fbfs en LN
  - Todo p ∈ V_{prop} es una fbf
  - ⊥ y ⊤ son fbfs
  - Si A y B son fbfs, → $\neg A$, $(A \wedge B)$, $(A \vee B)$, $(A \rightarrow B)$, $(A \preccurlyeq B)$, $(A \succcurlyeq B)$ son fbfs

- Semántica: si A es fbf de LN y t∈ Z, entonces:
  - $m_{tA}^{+} = \min\{t' \in Z | t' > t \; y \; A \; es \; verdadero \; en \; t'\}$
  - $m_{tA}^{-} = \max\{t' \in Z | t' < t \; y \; A \; es \; verdadero \; en \; t'\}$
  - $Min \; \emptyset = \; +\infty \; y \; Max \; \emptyset = \; -\infty$

- Un modelo LN es una función $h: V_{prop} \longrightarrow 2^Z$ donde se cumple:
  - $h(\neg A) = Z \setminus h(A)$
  - $h(A \wedge B) = h(A) \cap h(B)$
  - $h(A \vee B) = h(A) \cup h(B)$
  - $h(A \rightarrow B) = \big(Z \setminus h(A)\big) \cup h(B)$
  - $h(A \preccurlyeq B) = \{t \epsilon Z \; | m_{tA}^{+} < +\infty \;\; y \;\; m_{tA}^{+} \leq m_{tB}^{+} \}$
  - $h(A \succcurlyeq B) = \{t \epsilon Z \; | m_{tA}^{-} > -\infty \;\; y \;\; m_{tA}^{-} \geq m_{tB}^{-} \}$

    donde $m_{tA}^{+} = \big((t, +\infty) \cap h(A)\big) \; y \; m_{tA}^{-} = \big((-\infty, t) \cap h(A)\big)$

- Si A y B son fbfs de LN^+, n∈N y t∈Z, entonces:
  - $m^+_{t(A\vee B)} = \min\{m^+_{tA}, m^+_{tB}\}$
  - $m^+_{t(A\wedge B)} \geq \max\{m^+_{tA}, m^+_{tB}\}$
  - $m^+_{t\oplus^n A} = m^+_{(t+1)\oplus^{(n-1)}A} - 1$
  - $m^+_{t\ F\oplus^n A} = \begin{cases} t+1 & si\ m^+_{(t+n+1)A} < \infty \\ +\infty & en\ otro\ caso \end{cases}$
  - $m^+_{t\ G\oplus^n A} = \begin{cases} t+1 & si\ t+n+1 \in h(GA) \\ m^+_{(t+1)G\oplus^n A} & si\ t+n+1 \in h(FGA \wedge F\neg A) \\ +\infty & en\ otro\ caso \end{cases}$
  - $m^+_{t\ GFA} = \begin{cases} t+1 & si\ t \in h(GFA) \\ +\infty & en\ otro\ caso \end{cases}$
  - $m^+_{t\ FGA} = \begin{cases} t+1 & si\ t \in h(FGA) \\ +\infty & en\ otro\ caso \end{cases}$

- Propiedades: sean A y B fbfs de LN, h una interpretación temporal, y t∈ $Z \mid t \in h(A)$
  - A se dice válida si h(A)=Z. Se denota por $\vDash A$
  - A y B son equivalentes si h(A)=h(B) ∀h. Se denota por A≡B

- Conectivas definidas en LN
  - Conectiva fuerte de precedencia estricta: ≺
    $h(A \prec B) = \{t \in Z \mid\ m^+_{tA} < m^+_{tB}\}$
  - Conectiva débil de presencia estricta: ⊏
    $h(A \sqsubset B) = \{t \in Z \mid m^+_{tA} < m^+_{tB}\} \cup \{t \in Z \mid \{m^+_{tB} = \infty\}$
  - Conectiva débil de presencia amplia: ⊑
    $h(A \sqsubseteq B) = \{t \in Z \mid\ m^+_{tA} \leq m^+_{tB}\}$
  - Conectiva fuerte de simultaneidad: $\approx^+$
    $h(A \approx^+ B) = \{t \in Z \mid \{m^+_{tA} = m^+_{tB} < +\infty\}$
  - Conectiva débil de simultaneidad: $=^+$
    $h(A =^+ B) = \{t \in Z \mid\ m^+_{tA} = m^+_{tB}\}$
  - A atnextB: A será verdadera en la próxima ocurrencia de B
  - Isolf A: en el futuro A y en la próxima ocurrencia de A ocurrirá que mañana ¬A
  - A while B: en el futuro, A mientras B
  - A unless B: si alguna vez en el futuro B, entonces A hasta que B, o siempre en el futuro A.
  - Release (A, B): B es cierto en el futuro hasta que sea liberado por A.
  - A atlast B: A fue verdadero en la última ocurrencia de B
  - Isolp A: en el pasado A y en última ocurrencia de A, ocurrió que ayer ¬A
  - A while−B: en el pasado A mientas que B
  - A unless−B: si alguna vez en el pasado B, entonces A desde que B, o siempre en el pasado A.

- LN^+ usa tablas semánticas como método de refutación, es decir para analizar la validez de una fbf.

3.4.7. Lógica de Tiempo Ramificado: CTL

Es una lógica de tiempo ramificado en la que solo se contemplan conectivas de futuro. Su semántica evalúa las fórmulas sobre órdenes arbóreos.

- Un orden arbóreo < sobre un conjunto infinito numerable S es un orden parcial tal que $\forall s \in S, el\ conjunto\ (\leftarrow, s) = \{s' | s' < s\}$ es totalmente ordenado y tienen un mínimo elemento. De modo que si (s_i<s) ∧ (s_k<s) → (s_i<s_k) ∨ (s_i=s_k) ∨ (s_k<s_i). A los elementos de S se los denomina estados.

- Sea un orden arbóreo (S, <),
  - Un camino π es una secuencia s_i, s_{i+1},… de estados de S | s_{i+j}<s_{i+j+1}. Es decir, (π,<) es totalmente ordenado. Un camino puede ser finito o infinito. El conjunto de caminos se denota por Π
  - Una rama ρ es un camino maximal

- Un orden arbóreo (S,<) es discreto si cumple las siguientes condiciones:
  - $\forall s \in S, \exists s' > s \mid \forall s''que\ cumpla\ s'' > s\ entonces\ s'' \geq s'. (s\ es\ el\ suceor\ inmediato\ de\ s')$
  - $\forall s \in S, \exists$, un conjunto de sucesores inmediatos de s, Suc(s), cuyos elementos no son comparables 2 a 2, es decir, si s',s''∈ Suc(s), entonces ni s'<s ni s''<s'

- Un orden arbóreo tiene un grado de ramificación máximo n si y solo si, casa estado tiene a lo sumo n sucesores.

- Lenguaje de CTL
  - V_{prop} = {p_1, q_1, r_1, …., p_n, q_n, r_n}
  - Constantes booleanas: T y ⊥
  - Conectivas booleanas clásicas: ∧, ∨, →, ↔, ¬
  - Conectivas temporales monarias: $F, G, \oplus$
  - Conectivas temporales binarias: U y ≼
  - Símbolos de cuantificadores sobre ramas: A (para toda rama) y E (existen una rama)

- Inducción de fbfs:
  - ⊥, ⊤ y todo p ∈ V_{prop} son fbfs
  - Si ϕ es fbf, entonces $\neg\phi$, $A\oplus\phi$, $E\oplus\phi$, $AF\phi, EF\phi, EG\phi, y\ AG\phi$ son fbfs
  - Si ϕ y ψ son fbfs, entonces (ϕ * ψ) donde * ∈ {∧, ∨, →, ↔, ≼} son fbfs
  - Si ϕ y ψ son fbfs, entonces $A(\phi \preccurlyeq \psi), E(\phi \preccurlyeq \psi), AU(\phi, \psi) Y\ EU(\phi, \psi)$ son fbfs.
  - Sólo las cadenas obtenidas aplicando las reglas anteriores son fbfs.

- Un modelo ramificado es una tupla M = (S,<,h), donde (S,<) es un orden arbóreo discreto en el que todas las ramas son infinitas y $h: V_{prop} \longrightarrow 2^S$ es una función que asigna a cada p ∈ V_{prop} el conjunto de estados en el que satisface p.

- Una fbfs $\phi \in CTL$ se dice satisfacible, si $\exists\ M = (S, <, h)$ y un estado s∈S | S ∈ $h(\phi)$. Si una fbf es satisfacible, entonces posee un modelo finito cuyo tamaño está acotado por alguna función de la longitud de ϕ. En consecuencia, CTL es decidible.

- Leyes en CTL
  - $\neg A \oplus \phi = E \oplus \neg \phi$
  - $\neg AF\ \phi = EG \neg \phi$
  - $\neg EF\ \phi = AG \neg \phi$
  - $AG\ \phi = A(\neg A \preccurlyeq \bot)$
  - $AG\phi \wedge AG\psi \equiv AG(\phi \wedge \psi)$
  - $EF\phi \vee EF\psi \equiv EF(\phi \vee \psi)$

- CTL usa estructuras arbóreas y el algoritmo de etiquetado para analizar la validez de una fbf.

3.5. Conclusión

La lógica temporal permite formalizar una acción en momentos diferentes, gracias a los operadores modales. Y aunque sea compleja su representación, es un acercamiento muy intuitivo a la forma de pensar de los humanos, ya que se aproxima bastante a lo que se denomina “pensar en el futuro” y “recordar el pasado”.

En el caso que nos ocupa, la lógica temporal juega un papel crucial a la hora de ordenar temporalmente los predicados, requisito indispensable para una buena planificación de la asignatura. Haciendo uso de los razonadores automáticos se podrán inferir predicados válidos y coherentes que nos ayuden en la consecución de nuestro objetivo, una planificación eficaz que permitirá la superación de la asignatura.

En concreto, el modelo CTL sería el más adecuado, ampliamente utilizada en los verificadores automáticos. Desde un punto de vista funcional, nuestro modelo de representación puede tener ciertas similitudes con un verificador automático. El funcionamiento de un verificador es el siguiente: a partir de la descripción formal de un sistema y su comportamiento esperado, se chequea si el sistema cumple con el comportamiento esperado.

Dado un conjunto de elementos de entrada (unidades didácticas, pruebas, fechas), mediante la aplicación de un algoritmo se obtienen distintos esquemas de planificación, los cuales serán validados mediante un verificador automático. De este modo, el sistema deberá ser capaz de comprobar que la planificación propuesta cumple con las fechas requeridas, y seleccionará de este modo el esquema de planificación válido.

III. ELECCIÓN DEL MODELO PARA LA REPRESENTACIÓN

1. FORMULACIÓN DEL PROBLEMA

El conocimiento a representar es la programación de una asignatura desde un punto de vista doble: por un lado, se desea modelar el diseño de la enseñanza ordenando todos los elementos que intervienen en un plazo determinado de tiempo. Este modelo da como resultado el calendario del aula de la asignatura. Desde un segundo plano, se desea detallar la programación personal del estudiante, en el que aparecen como premisas iniciales el calendario de la asignatura, otras asignaturas que esté cursando el alumno, el tiempo del que dispone para dicha asignatura, conocimientos previos, etc. Este segundo modelo parte del modelo de planificación estándar, y teniendo en cuenta los parámetros y restricciones temporales del estudiante, obtiene un calendario personal del estudiante. Por tanto este modelo surge de la adaptación del calendario estándar, y será un submodelo del primero. El fin a alcanzar es la obtención de conocimiento necesario para la superación de la asignatura.

Existen multitud de algoritmos muy potentes que intentan realizar el arte de la planificación de forma automática. Estos algoritmos están enfocados a problemas de dominio muy sencillo en los que se usa muy poco conocimiento. Sin embargo, para problemas del mundo real, con una carga de conocimiento implícito muy fuerte, los algoritmos se dejan informaciones importantes sin considerar.

Las tareas de planificación típicamente consisten en obtener un conjunto ordenado de acciones, cuya ejecución permite alcanzar unos objetivos determinados. La dificultad más importante que presenta este tipo de tareas es que su resolución tiene un coste computacional muy elevado.

El problema en cuestión es una planificación temporal, con una percepción muy concisa, unas acciones deterministas, durativas, concurrentes, con recursos discretos y coste unitario, y un entorno observable dinámico, aunque con pautas establecidas previamente. En planificación temporal cobra especial relevancia la duración de las acciones, así como la concurrencia de las mismas.

Empezaré por definir los conceptos generales asociados a mi universo de representación, para luego mediante especializaciones sucesivas llegar a los objetos, sus propiedades y sus relaciones, siguiendo de este modo un método *top-down* (descendente). Los objetos que intervienen son (www.wordreference.com):

- Programación: es la ordenación en el tiempo de las acciones necesarias para la superación de la asignatura.
- Tiempo: duración de las cosas sujetas a cambios o de los seres que tienen una existencia finita. En este caso particular, el tiempo es considerado como intervalo entre dos fechas.
- Fecha: tiempo concreto, momento puntual en el que el sucede un acontecimiento.
- Asignatura: cada una de las materias que se enseñan en un centro docente o de que consta una carrera o plan de estudios.
- Prueba: examen o experimentación para comprobar el buen funcionamiento de alguna cosa, o su adecuación a un determinado fin.

- Planificar: hacer un plan de acción.
- Recurso docente: herramientas para la difusión y asimilación del conocimiento.
- Actividades a realizar por el estudiante:
    - Leer o estudiar contenido docente.
    - Buscar información relacionada con la asignatura para ampliar el conocimiento.
    - Realizar ejercicios o actividad práctica.
    - Realizar una PEC.
    - Resolver o plantear dudas en el aula de la asignatura.
    - Entrar periódicamente al aula para estar al día de las notificaciones por parte del consultor.

Al materializar esos conceptos generales, adaptándolos al tema en estudio, se desprenden los siguientes objetos más específicos:

- Programación: secuenciación de todas las tareas y activadas inherentes a la asignatura. Ésta es una programación general realizada por el consultor, en la que se proponen acciones a realizar y fechas límites para su realización.
    - Programación personalizada del estudiante. Introduce además los factores personales del estudiante que interfieren en dicha planificación en cuanto a que limitan el tiempo. Esta segunda planificación parte de la primera estándar, y mediante reiterados refinamientos obtiene una planificación más precisa y detallada, en la que se contemplan actividades del estudiante ajenas a la asignatura en cuestión.

- Fecha: momentos puntuales en los que sucede un evento.
    - Fecha publicación: fecha en la que se publica una prueba.
    - Fecha entrega: fecha máxima en la que se tiene que entregar una prueba.
    - Fecha de solución: fecha en la que se publica la solución de una prueba.
    - Fecha calificación: fecha en la que se publica la calificación de una prueba.
    - Fecha de prueba de validación y examen: fecha en la que tiene lugar la prueba de validación y el examen.

- Asignatura:
    - Créditos: el crédito es la unidad de medida de la carga académica de la asignatura. Cada crédito se corresponde con 25 horas de trabajo por parte del estudiante.
    - Unidad didáctica: segmento estructurado en los que se divide una asignatura. A cada unidad didáctica le corresponde un número de créditos del total de la asignatura.
    - Ejercicios: casos prácticos de cada una de las unidades didácticas que permiten practicar la teoría aprendida en la unidad.
    - Otras actividades de aprendizaje.
    - Guía didáctica: índice de contenidos de la asignatura.
    - Foro: aplicación web que da soporte a discusiones u opiniones en línea.

  - Tablón: aplicación web donde se publicarán las notificaciones relevantes para la asignatura.
  - Bibliografía: relación de libros o escritos referentes a los contenidos de la asignatura.

- Prueba:
  - Prueba de evaluación continua (PEC): ejercicio práctico que se realiza durante el trimestre y valida los conocimientos adquiridos por el estudiante para aprobar mediante la modalidad de evaluación continúa. Cada asignatura puede tener varias pruebas de evaluación continua. Cada PEC tiene una fecha de publicación, una fecha de entrega, y una fecha de solución.
  - Práctica (PR): ejercicio que permite poner en práctica los conocimientos adquiridos en una asignatura. Cada asignatura puede tener cero, una o varias PR. Cada PR tiene una fecha de publicación, una fecha de entrega, y una fecha de solución. La superación de la PR es imprescindible para aprobar la asignatura.
  - Prueba de validación (PV): prueba presencial que valida los conocimientos adquiridos en la realización de las PECs y las PRs, y en general los conocimientos de la asignatura.
  - Examen (Ex): prueba presencial que valida los conocimientos adquiridos en la asignatura, cuando no se escoge la modalidad de evaluación continua o se suspende la evaluación continúa.
  - Calificación: nota con que se valora una prueba.

- Restricciones temporales: conjunto de las restricciones temporales y personales de cada alumno que limitan el tiempo de su programación personal. También englobará las restricciones implícitas en el estudio de una asignatura.

A continuación, para desgranar un poco el universo de estudio, se pasa a la definición de las propiedades o atributos extrínsecos de cada uno de los objetos. Es posible que existan propiedades comunes a varios objetos pertenecientes a un mismo concepto, en ese caso, se definirán las propiedades en el concepto general y no en el atributo. En la siguiente tabla se muestran las propiedades por objeto o concepto, según el caso.

Fecha	**Asignatura**	**Unidad Didáctica**	**Prueba**	**Programación**
Inicio	Nombre	Nº Créditos	Nombre	Estándar
Final	Nº créditos	Nombre	Fecha Publicación	Estudiante
	Profesor Responsable	Aula	Fecha Entrega	Consultor
	Foro	Material Didáctico	Fecha Calificación	Tiempo
	Tablón	Bibliografía	Fecha Entrega	
	Bibliografía	Ejercicios	Tipo	

Ilustración 10. Elementos del modelo planificación

Relaciones: las relaciones se muestran en el diagrama UML que se muestra a continuación. Algunas de ellas son:

- Unidad didáctica es un prerrequisito para realizar PEC.
- Una guía didáctica está compuesta por unidades didácticas.
- Unidad didáctica está compuesta de material didáctico, y bibliografía.
- Una asignatura tiene una programación estándar.
- Una asignatura tiene varias pruebas.
- Una prueba tiene varias fechas: fecha de publicación o inicio, fecha de entrega, fecha de calificación.
- Una prueba puede ser de varios tipos (herencia disjunta).
- Una programación puede ser estándar o personalizada por el estudiante (herencia disjunta).
- La programación del estudiante se construye a partir de la programación estándar.
- Unidad didáctica tiene ejercicios.

Restricciones:

- Para realizar la PV tiene que estar aprobada la evaluación continua y la parte práctica.
- Para realizar el EX tiene que estar aprobada la parte práctica.
- Para cursar una asignatura, es posible que haya que tener aprobada una o varias asignaturas determinadas.
- Para realizar una determinada PEC, es necesario haber adquirido previamente los conocimientos que se desprenden de las unidades didácticas que engloban dicha PEC.
- El estudiante tiene un tiempo finito para dedicarlo a la asignatura, lo que constituirá un conjunto de restricciones temporales.
- La programación personal del estudiante debe ceñirse a las fechas que dicta la programación estándar.

Para personalizar la programación y adaptarla a las necesidades de un estudiante particular, es necesario considerar las restricciones temporales del estudiante; esto es, el tiempo que el estudiante tiene disponible para dedicárselo a la asignatura. Este tiempo vendrá determinado por varios factores de la vida personal del estudiante, pero no cabe su discusión en el problema que se abarca. Sin embargo, sí será necesario considerar de manera distinta el tiempo definido en la planificación estándar. Este sub-tiempo será como un subconjunto más reducido del tiempo considerado. Esta partición se puede formar imponiendo intervalos de tiempo específicos dentro del plan de tiempo inicial. Esto podría pasar por determinar franjas horarias dentro de un día, y días concretos dentro de una semana. Así se obtiene una nueva materialización y medida del tiempo, introduciendo las horas y los días. De este modo, se llegan a considerar 3 dimensiones temporales distintas: duración, días y horas.

Mediante un diagrama UML se muestran las entidades más representativas del modelo y las relaciones entre ellas:

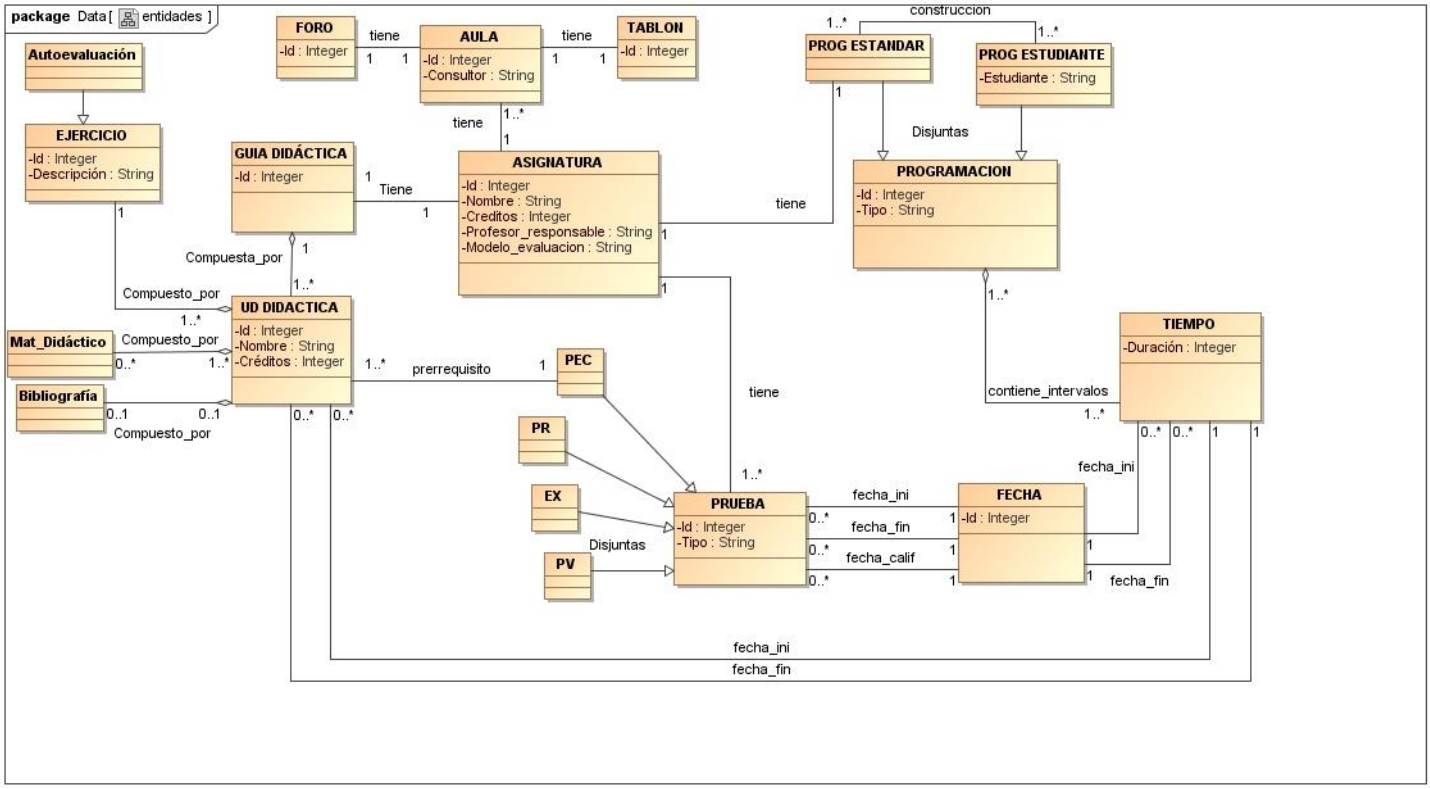


Ilustración 11. Ampliación del diagrama UML del modelo de planificación

2. DISCUSIÓN DEL MODELO

2.1. BD ORIENTADA A GRAFO

Las BD orientadas a grafos proporcionan una estructura de datos sólida para la representación del universo causa de estudio. Esta estructura puede ser de gran utilidad; en los nodos quedarían almacenadas las distintas entidades (asignatura, PEC, PR, EX, PV, fecha, unidad didáctica, ejercicio, programación estándar, programación estudiante, calificación, tiempo, etc.) y en las aristas las relaciones entre entidades (una asignatura tiene distintas unidades didácticas, cada asignatura tiene distintas pruebas que tendrá que realizar el estudiante, cada prueba tiene unas determinadas fechas de publicación, entrega y calificación, una programación contiene una forma de tiempo, la programación del estudiante tiene restricciones temporales, etc.) formando un grafo en el que cada nodo tiene un puntero a sus nodos adyacentes. Además, este modelo permite darle mucha importancia a las relaciones, e incluso permite definir atributos de las mismas, algo que facilitará mucho la planificación, al conseguir unas consultas más rápidas y eficientes gracias también a la utilización de la teoría de grafos.

Otras de las características de este modelo que lo hacen idóneo para la planificación de una asignatura es la flexibilidad que ofrece esta estructura de datos (los grafos son aditivos), pudiendo modificarla sin mucho esfuerzo en el transcurso de su vida (añadir o suprimir tanto nodos como relaciones). Esto es de vital importancia ya que la educación es una ciencia viva, y cualquier actividad de aprendizaje o prueba podría sufrir modificaciones en un momento determinado, sin penalizar el rendimiento y accesibilidad del sistema global. Por otra parte, los datos en este modelo están fuertemente conectados y requieren de análisis continuos y exhaustivos, punto fuerte de las BD orientadas a grafos por su propia topología.

Sin embargo, las BD orientadas a grafos no son muy adecuadas para entornos descentralizados y con acceso en tiempo real, ya que la partición de la estructura de datos en subgrafos puede conllevar una falta de integridad en los datos. La gran cantidad de planificaciones estándares para las distintas asignaturas, y por otra parte las planificaciones personales de cada estudiante, puede hacer inviable el almacenamiento en un entorno centralizado. Se podrían estudiar distintos diseños de fragmentación (por ejemplo, se podría tener una BDG distinta y separada para cada una de las asignaturas), sin embargo ese aspecto no es ámbito de este trabajo.

2.2. ONTOLOGÍA

Para la creación del universo objeto de discusión, sería interesante comenzar por un desarrollo dirigido por ontología, lo cual ayudaría a la definición y clasificación de entidades, relaciones y demás elementos del universo. También hay que considerar la capacidad de complementación que tienen estas estructuras con las BD orientadas a grafos, lo que resultaría una baza interesante para combinar ambos modelos.

Al crear una ontología, se materializa la clasificación de elementos y relaciones que aparecen en el modelo de conocimiento, facilitando de este modo la estructura general del modelo, y la formulación de las consultas sobre los elementos del modelo. De este

modo, mediante el uso de ontologías se podrían definir el conjunto de clases, relaciones, funciones, atributos y constantes para el dominio de planificación de una asignatura. Básicamente, el papel que desempeñará la ontología es facilitar la construcción de un modelo de dominio, el cual proveerá un vocabulario de términos y relaciones con las cuales se puede modelar el dominio.

El diseño conceptual se realizará mediante ontologías conforme al siguiente esquema:

2.2.1. Determinar el dominio y alcance de la ontología

Se comenzará definiendo un modelo ontológico que represente nuestro universo de discusión. Para ello, se comienza con la definición del dominio para el cual se crea la ontología, es decir, el uso que se le dará a la ontología. El dominio de esta ontología es el modelo para la planificación de una asignatura de la UOC. Los conceptos que se definen son los detallados en el apartado "**Error! Reference source not found.**".

La ontología que se está diseñando será utilizada en la etapa de especificación del diseño conceptual, tanto para el modelo de planificación estándar como para el modelo de planificación del estudiante, y deberá incluir el detalle de todos los elementos anteriormente descritos (asignaturas, unidades didácticas, ejercicios, pruebas de validación, pruebas de evaluación continua, etc.).

Una de las formas de determinar el alcance de la ontología es formulando una serie de preguntas que la base de conocimiento basada en la ontología debería ser capaz de responder, estas son las llamadas preguntas de competencia. Tales preguntas podrían ser las siguientes:

- Número de créditos de la asignatura.
- Número de pruebas que contiene la asignatura.
- Número de créditos de cada unidad de aprendizaje (determinarán la carga lectiva de cada una).
- Duración temporal para la realización de cada actividad de aprendizaje.
- Superación o no de una prueba.
- Número de asignaturas en que se ha matriculado el estudiante (en caso de que se contemplaran programaciones de estudiante de varias asignaturas combinadas).
- Número de horas libres al día que dispone el estudiante para dedicarlas a la asignatura.
- Días de la semana que el estudiante puede dedicar una franja horaria a la asignatura.

2.2.2. Considerar la reutilización de ontologías existentes.

Existen multitud de ontologías que se pueden reutilizar, refinando y adaptando los recursos que ofrezca la más afín al modelo que se desea representar. Sin embargo, es muy probable que el formalismo en el que está expresado la ontología sea muy distante a las necesidades requeridas por el modelo a representar, en cuyo caso el

trabajo de adaptación es mayor que el de realizarla desde el principio. Existen numerosos repositorios de ontologías ya creadas, como por ejemplo la librería DAML (http://www.daml.org/ontologies/) donde se pueden buscar ontologías por muchos criterios.

En esta ocasión, se considera más práctico no partir de ningún modelo existente, dada la particularidad de los elementos que forman parte del modelo, y puesto que el modelo no es muy denso en extensión.

2.2.3. Enumerar términos importantes para la ontología

Es conveniente realizar un listado con todos los términos que intervienen en el dominio de estudio. Se parte de una lista tipo lluvia de ideas sin importar las propiedades o relaciones que pueden tener, para posteriormente mediante refinamientos sucesivos se van detallando y clasificando los conceptos. Algunos de los conceptos pertenecientes a la programación de una asignatura son enumerados y definidos en el apartado "**Error! Reference source not found.**":

Aprobado	Apto	Área	Asignatura	Autoevaluación
Bibliografía	Capítulo	Carrera	Ciclo	Concepto
Contenido	Créditos	Cuatrimestre	Definición	Día semana
Dificultad	Duración	Ejercicio	Estricta	Evaluación
Examen	Final	Flexible	Glosario	Guía didáctica
Foro	Restricciones temporales			
Hora	Índice	Inicio	Introducción	Limitaciones
Matricula honor	No apto	Notable	Número páginas	Objetivos
Periodo	Prueba de evaluación continua	Prueba de validación	Requisitos previos de conocimiento	Sección
Sobre saliente	Suspenso	Tablón	Tiempo requerido	Unidad didáctica

Ilustración 12. Enumeración de Términos

Tras la lluvia de ideas, se puede crear un glosario de los términos principales inherentes al modelo que se está representando. A continuación se muestra un ejemplo de definición de aluno de los conceptos:

Nombre	Descripción
Contenido	Conjunto secuencial de los temas de la asignatura organizados por capítulo y sección
Bibliografía	Cita de las fuentes consultadas (libros , trabajos, o enlaces electrónicos) para la elaboración del temario de la asignatura
Créditos	Unidad de medida de la carga académica de la asignatura. Cada crédito se corresponde con 25 horas de trabajo por parte del estudiante
Ejercicio	Enunciado cuyo desarrollo permite la práctica y comprensión de la teoría.

Evaluación	Elemento educativo cuyo fin es verificar los conocimientos adquiridos por el estudiante en una determinada materia
Glosario	Definición de términos
Guía Didáctica	Documento que expone y facilita la comprensión del material didáctico por parte del estudiante, y orienta y organiza el trabajo integrando los elementos didácticos para el estudio de la asignatura.

Ilustración 13. Glosario de Términos

Por tanto, las Ontologías proporcionan un vocabulario y organización de conceptos que representan un marco de trabajo conceptual para el análisis, discusión o consulta de información de un domino. Sin embargo, existe la necesidad de realizar tareas de razonamiento en ellas, por lo cual se debe integrar módulos o herramientas de razonamiento de acuerdo al desarrollo ontológico realizado.

2.3. LÓGICA TEMPORAL

Para realizar estas tareas de razonamiento en Ontologías, se propone el otro modelo expuesto, la lógica temporal. La planificación es eminentemente temporal y cambiante, y necesita de un modelo que recoja las consecuencias inequívocas de entornos cambiantes durante el paso del tiempo. También será de gran utilidad en la definición de las restricciones temporales, tanto las del propio estudiante como las del modelo en sí. Las restricciones temporales del modelo se pueden definir como un conjunto de reglas escritas en un lenguaje de lógica descriptiva temporal.

Esto es, tanto las restricciones inherentes a la asignatura en sí (por ejemplo, antes de cursar la asignatura *Inteligencia Artificial II* el alumno tienen que tener superada la asignatura *Inteligencia Artificial I*) como las restricciones temporales del estudiante (por ejemplo, es posible que el estudiante solo pueda dedicarle tiempo a la asignatura los lunes, miércoles y fines de semana). Otro ejemplo lo constituye la secuenciación de asignaturas; en ocasiones, para cursar una asignatura es necesario tener aprobada otra u otras asignaturas. Estos y otros forman el conjunto de las restricciones temporales. Todas estas comprobaciones y razonamiento serán factibles de implementar con la lógica descriptiva temporal.

La estructura de este modelo descriptivo temporal se puede adaptar fácilmente al problema de estudio (conceptos, roles e individuos formados a partir de los constructores, y el conjunto de axiomas para dar aserciones acerca de esos conceptos, roles e individuos), y además este tipo de lógicas proveen servicios propios de inferencia (procedimientos de decisión sólidos y completos para la resolución de los problemas).

2.4. CONCLUSIÓN

Tras lo expuesto, se puede concluir que las tres técnicas de representación del conocimiento presentadas (BD orientadas a grafos, lógica descriptiva temporal y ontologías) no se excluyen entre sí, sino que enfatizan distintas formas de enfocar el

dominio que se ha de representar. Es por ello, que se propone hacer un uso combinado de las tres, en el cual se aproveche las bondades de cada uno de los modelos, para aplicarlos en distintas fases o componentes del domino. De esta forma, se propone el uso de ontologías para definir y representar el esquema general del modelo de estudio durante la fase de diseño conceptual, las BD orientada a grafos para la creación y el almacenamiento de las estructuras de datos, y la lógica descriptiva temporal se utilizará para la definición de las restricciones temporales tanto del estudiante, como las inherentes a la asignatura.

3. REPRESENTACIÓN DEL MODELO

3.1. BD ORIENTADAS A GRAFOS

Tal y como se comentó en un capítulo anterior (“Discusión del modelo”), se hará uso de las BD basadas en grafos para crear la estructura de datos que soportará el modelo. En los nodos se almacenarán las entidades anteriormente descritas, juntos con sus propiedades o slots, y las relaciones serán almacenadas en las aristas.

El diagrama UML de las entidades del modelo mostrado en la Ilustración 11 es la base para crear la BDG.

Neo4j sería un software ideal para la creación de esta BDG, ya que es de software libre (Java), ampliamente utilizado para redes sociales y sistemas de recomendación, (el modelo objeto de estudio comparte muchas características con ambos modelos).

3.2. ONTOLOGÍA

Ya ha quedado de manifiesto en el apartado anterior el uso que se hará de las Ontologías: servirá para la etapa de especificación durante el diseño conceptual, ya que son tremendamente útiles para detectar y definir conceptos relevantes, jerarquías, relación entre los conceptos, propiedades, etc.

Además, las ontologías proporcionan un vocabulario rico con el que poder expresar el conocimiento, usando distintos niveles de detalle. Esta capacidad de razonamiento abstracto permite la realización de distintas inferencias a partir del conocimiento disponible. También permite plantear problemas de planificación abstractos cuyas soluciones pueden aplicarse a multitud de problemas concretos, algo que resulta especialmente interesante en el caso cuestión de estudio.

Se propone como lenguaje ontológico el OWL, recomendación del W3C para especificar ontologías, y que permite definir las condiciones necesarias y suficientes para definir la pertenencia a una clase. OWL Lite permite establecer restricciones simples sobre cómo las instancias pueden utilizar las propiedades de una clase. OWL hace uso de las lógicas descriptivas para definir un conjunto de restricciones, las cuales además permiten el uso del razonador. Entre las principales tareas de inferencia con lógica descriptiva se encuentran dos de gran utilidad para la jerarquía de clases: subsunción (comprobar si una categoría es subconjunto de otra) y clasificación (comprobar si un objeto pertenece a una categoría).

3.3. LÓGICA TEMPORAL

La lógica temporal se usará en el presente modelo desde una doble perspectiva; por una parte, definirá los axiomas lógicos. Esto es, expresiones lógicas que tendrán que ser siempre verdaderas en la ontología. En realidad, pera este propósito no es necesario una lógica temporal, siendo suficiente una lógica de primer orden. Sin embargo, con el fin de conectar con los modelos propuestos anteriormente, resulta

muy interesante el uso de lógicas descriptivas (DL) por dos motivos principales: son ampliamente utilizadas en lenguajes ontológicos (OWL) y además son relevantes en la representación y el razonamiento de modelos de bases de datos (los diagramas UML se integran eficazmente dentro de las lógicas descriptivas, y los razonadores DL pueden utilizarse para verificar su consistencia (Lutz, C., Wolter, F., Zakharyashev, M., 2008).

El principal reto a la hora de usar las DL's es la búsqueda de algoritmos de decisión para los problemas de inferencia tradicionales, como son la satisfactibilidad, subsumición y consistencia en las aserciones entre individuos para lógicas de descripción expresivas y con la menor complejidad posible.

Los diferentes tipos de lógica se distinguen entre sí por el nivel de expresividad que alcanzan, lo cual es inversamente proporcional a la complejidad: cuanto más expresivo es el lenguaje, más complicado será el razonar en esa lógica. Es por ello que habrá que buscar un equilibrio entre expresividad y complejidad.

Volviendo al problema objeto de estudio, para la definición de las restricciones inherentes al modelo se combinará una lógica descriptiva ALC (la DL más simple) con una lógica temporal lineal (TLC), obteniendo de este modo una lógica descriptiva temporal TDL. El tiempo será usad en su faceta de intervalos. Los operadores de TLC deberán ser aplicados sobre los roles de la ALC, puesto que el carácter temporal afectará sobre todo al modo en que pueden relacionarse los conceptos (clases o entidades).

IV. CONCLUSIONES

El presente trabajo propone un modelo de representación del conocimiento necesario para la planificación óptima y eficaz de las asignaturas de la UOC. Dicha planificación tiene por objetivo inicial la superación de los contenidos académicos por parte del estudiante, adquiriendo los conocimientos necesarios, realizando las pruebas requeridas dentro de las fechas solicitadas, compatibilizando con el resto de las actividades académicas del estudiante y con sus limitaciones temporales personales y mejorando al fin la gestión del tiempo del alumno.

Para dicha representación, se han estudiado tres modelos distintos que a priori podían encajar en la materialización del modelo. A continuación se exponen las conclusiones obtenidas tras el estudio de cada uno de los modelos.

Las BDG ofrecen las siguientes características que resultan apropiadas al modelo de planificación de una asignatura:

- Una BDG facilita la exploración de datos al disponer de una estructura de grafo, especialmente cuando las relaciones entre esos datos son tan significativas como los datos mismos.
- El rendimiento de una BDG es muy bueno ya que se beneficia de la teoría de grafos para el recorrido de la BD.
- Resultan muy útiles para almacenar la información en modelos con múltiples relaciones, como redes y conexiones sociales.
- Tienen una gran facilidad para modelar y adaptarse a modelos cambiantes, bondad que se podrá aprovechar en el modelado de las distintas asignaturas, puesto que no todas tienen porqué seguir el mismo patrón.

Por su parte, las ontologías juegan un papel crucial en la compartición del conocimiento entre los diferentes actores de un determinado dominio, por la disponibilidad del conocimiento almacenado y por los mecanismos que dispone para gestionar y tener accesible la información.

Las ontologías intentan formular un esquema conceptual exhaustivo y riguroso de un dominio dado con la finalidad de facilitar la comunicación, reusar y compartir información. De este modo, se puede hacer uso de la ontología para definir un vocabulario común que incluya además la interpretación de los conceptos básicos del dominio y sus relaciones en la especificación del diseño conceptual.

Por último, la lógica temporal permite formalizar una acción en momentos diferentes, gracias a los operadores modales. Ordena los predicados temporalmente, base fundamental para la planificación. Haciendo uso de los razonadores automáticos se pueden inferir predicados válidos y coherentes para una planificación eficaz.

La lógica temporal, aunque sea muy complicada de representar, representa un acercamiento muy intuitivo a la forma de pensar de las personas, se aproxima a lo que se denomina “pensar en el futuro” y “recordar”.

Tras el estudio de los tres modelos de representación, se opta por un uso combinado de los mismos, haciendo uso de las bondades de cada uno de los modelos. Se propone el uso de ontologías para definir y representar el esquema general del modelo

de estudio durante la fase de diseño conceptual, las BD orientada a grafos para la creación y el almacenamiento de las estructuras de datos, y la lógica descriptiva temporal se utilizará para la definición de las restricciones temporales tanto del estudiante, como las inherentes a la asignatura.

En el presente trabajo se ha definido caso de estudio para la aplicación del esquema propuesto con el uso de los distintos modelos. En trabajos futuros, se podría continuar por la implementación de la planificación en cada uno de los modelos, y la programación software posterior.

APÉNDICE A

Metodología	Uso	Desarrollador	Fases
Uschold y Grüninger	Enterprise Ontology	Universidad de Edimburgo junto a IBM Lloyds y Unilever	• Identificación del propósito • Construcción de la ontología • Evaluación • Documentación
Gruninger y Fox	Ongología TOVE	Toronto Virtual Enterprise	• Captura de escenarios • Formulación de preguntas • Especificar terminología en un lenguaje formal • Establecer condiciones para la determinación de una ontología completa
Amaya	Reúso conocimiento en procesos técnicos complejos	Proyecto ESPRIT KACTUS	• Especificación de la aplicación • Diseño preliminar con nivel abstracción general • Refinamiento • Estructuración
SENSUS	Base de conceptos abstractos identificados en SENSUS		• Descartar términos no relevantes de Ontología SENSUS • Obtener los términos relevantes de SENSUS • Añadir términos relevantes
Methontology	Permite construcción ontologías en un nivel de conocimiento	Ontoloty Group de la Universidad Politécnica de Madrid	• Especificación de requisitos • Adquisición de conocimiento • Conceptualización • Formalización • Integración • Implementación • Evaluación • Documentación

BIBLIOGRAFÍA

Aranda, G. N., Ruíz, F. (2005). "Clasificación y ejemplos del uso de ontologías en Ingeniería del Software". Universidad Nacional del Comahue, Argentina. Universidad de Castilla-La Mancha, España. Disponible en http://sedici.unlp.edu.ar/handle/10915/23076

Boehm, B. (1984). "Verifying and validating software requirements and design specifications"*, IEEE Software.*

Borst, W. N. (1997). "Construction of Engineering Ontologies for Knowledge Sharing and Reuse", Ph.D. thesis. *Centre for Telematics and Information Technology*. University of Twente, Enschede, The Netherlands.

Burrieza Muñiz, A., Pérez de Guzman Molina, I., Valverde Ramos, A. "Lógica para la Computación". *Universidad de Málaga*. Disponible en http://www.matap.uma.es/profesor/guzman/web/Bienvenida_files/5-Logica-Abductiva-Paraconsistente-Web.pdf

Camacho, E. (2010). "NoSQL la evolución de las bases de datos". Disponible en http://sg.com.mx/revista/42/nosql-la-evolucion-las-bases-datos

Casas Roma, J. (2013). "Lógica temporal: perspectiva histórica y relación con los flujos de tiempo". *Trabajo fin de Master*. Universidad de Salamanca.

Chandrasekaran B., Josephson J.R., Benjamins V. (1999). "What Are Ontologies, and Why Do We Need Them?". *IEEE Intelligent Systems*. Disponible en http://www.csee.umbc.edu/courses/771/papers/chandrasekaranetal99.pdf

Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks.

Corcho, O., Fernandez-Lopez, M., Gomez-Perez, A. (2003). "Methodologies, tools and languages for building ontologies. Where is their meeting point?". *Data & Knowledge Engineering*. Disponible en http://oa.upm.es/2637/1/JCR02.pdf

Domenjoud, M. (2012). "Graph databases: an overview". *Blog octo-talks!.* Disponible en http://blog.octo.com/

Flores, D. (2014). "Neo4j – Una guía rápida". Disponible en http://devniel.com

Garcete, A. "Base de Datos Orientadas a Columnas". *Universidad de Paraguay.*

García Jiménez, A. (2004). "Instrumentos de representación del conocimiento: tesauros versus ontologías"**.** Universidad Rey Juan Carlos I. Disponible en http://revistas.um.es/analesdoc/article/view/1691

Genesereth, M., Fikes, R. (1992). "Knowledge Interchange Format". *Technical Report Logic*. Stanford University. Disponible en http://logic.stanford.edu/kif/hypertext/kif-manual.html

Gómez-Pérez, A. (Coordinadora.). (2002). "A survey on Ontology Tools". *OntoWeb Deliverable 1.3*. Disponible en http://icc.mpei.ru/documents/00000826.pdf

Gómez-Pérez, A., Moreno, A., Pazos, J., Sierra-Alonso, A. (2000). "Knowledge Maps: An Essential Technique for Conceptualization", *Data & Knowledge Engineering.* Disponible en http://oa.upm.es/6482/1/Knowledge_maps_An_essential.pdf

Gruber, T. R. (1993). "A Translation Approach to Portable Ontology Specifications", disponible en http://tomgruber.org/writing/ontolingua-kaj-1993.htm

Guarino, N. (1998). *Formal Ontology in Information Systems*. Proceedings of FOIS'98, Trento, Italy, IOS Press, Amsterdam.

http://cala.unex.es/cala/epistemowikia/index.php?title=Ontolog%C3%ADas#Lenguajes_de_ontolog.C3.ADas

http://elies.rediris.es/elies18/531.html

http://en.wikipedia.org/wiki/OrientDB

http://ontologia.net/

http://protege.stanford.edu

http://www.orientechnologies.com/orientdb/

http://www.w3.org/TR/owl-features/

http://www.w3.org/TR/rdf-primer/

Huertas, M. A. (2006). "Lógicas para la red". *Universidad Oberta de Cataluña.*

Jurisica I., Mylopoulos J., Yu E. (1999). "Using ontologies for knowledge management: An information systems perspective". *Proceedings of 62nd Annual Meeting of the American Society for Information Science* (ASIS99).

Kifer, M., Lausen, G., Wu, J. (1995). "Logical Foundations of Object-Oriented and Frame-Based Languages." *Journal of the ACM*. Disponible en http://www.csee.umbc.edu/courses/771/papers/flogic.pdf

Kollegger, A. (2010). "What's new in Neo4j 2.0". *Neo Technology*. Boston,

La Web Semántica, disponible en http://www.hipertexto.info/documentos/web_semantica.htm

Lenat, D. B., Guha R. V. (1990). *Building large knowledge-based systems. Representation and inference in the Cyc Project*. Boston, Addison-Wesley.

Lógica para la Computación. IV) Lógicas modales temporales. Alfredo Burrieza Muñiz, Inmaculada Pérez de Guzmán Molina, Agustín Valverde Ramos.

Lutz, C., Wolter, F., Zakharyaschev, M. (2008), "Temporal Description logic: A Suvery", *computational-logic.org*

Mahesh, K. (1996): *Ontology Development for Machine Translation: Ideology and Methodology.* New Mexico State University, Las Cruces, NM: Computing Research Laboratory.

Martínez Comeche, J. A., "Tutorial ontologías". *Universidad Complutense de Madrid.*

Motta, E. (1999). *Reusable Components for Knowledge Modelling*. Amsterdam. IOS Press.

Ramos, E., Nuñez, H. (2007). "ONTOLOGÍAS: Componentes, Metodologías, Lenguajes, Herramientas y Aplicaciones". Universidad Central de Venezuela. Disponible en http://lia.ciens.ucv.ve/LIA/publicaciones.php

Rodríguez Gonzalez, M. E., Conesa i Caralt, J. "Modelo en Grafos". *Universidad Oberta de Cataluña*.

Rodríguez Gonzalez, M. E., Conesa i Caralt, J., Urbón Bayes, P. "Modelos de Agregación: Características Generales". *Universidad Oberta de Cataluña*.

Sánchez-Cuadrado, S., Morato-Lara,J., Palacios-Madrid,V., Llorens-Morillo, J. Moreiro-Gónzalez J.A. (2007). "De Repente, ¿Todos Hablamos de Ontologías?". Universidad Carlos III de Madrid. Disponible en http://www.elprofesionaldelainformacion.com/contenidos/2007/noviembre/03.pdf

Studer, R., Benjamins, V. R. & Fensel, D. (1998), "Knowledge engineering: Principles and methods". *Data and Knowledge Engineering*. Disponible en http://www.hubscher.org/roland/courses/hf760/readings/studer98knowledge.pdf

"The Neo4j Manual" (2014). Disponible en http://www.neo4j.org/

Torres Soler, L. C., "Anexo C. Lógica Temporal". *Universidad de Colombia.*

User Manual Sparksee by Sparsity Technologies.

Van Fraassen, B. C, (1978). *Introducción a la filosofía del tiempo y del espacio*. Barcelona. Labor.

Van Heijst, G., Schereiber, A.T. Y Wielinga, B.J. (1996). "Using Explicit Ontologies in KBS Development". *International Journal of Human and Computer Studies*.

Weigand, H. (1997). "Multilingual Ontology - Based Lexicon for News Filtering - The TREVI Project", en K. Mahesh (1997).

World Wide Web Consortium (2006), disponible en http://www.w3c.es/divulgacion/guiasbreves/websemantica