Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Este capítulo sirve de introducción a la presente Memoria de Proyecto de Fin de Carrera. En primer lugar se realiza una breve introducción de los conceptos generales del proyecto. La siguiente sección describe conceptos orientados a la Ingeniería de Lenguajes Dirigida por Modelos y al Desarrollo Dirigido por Modelos. La tercera sección describe las motivaciones y objetivos. La última sección está dedicada a describir la estructura del documento.

Índice

1.1. Contexto del proyecto

El principal objetivo de este proyecto es la implementación de un generador de código Cassandra-CQL (Cassandra Query Language) a partir de modelos UML, para ello se utilizaran una serie de técnicas orientadas al desarrollo software dirigido por modelos, estas reglas han sido definidas por los profesores Pablo Sánchez Barreiro y Carlos Blanco Bueno de la Universidad de Cantabria.

Uno de los aspectos más valorados por las empresas dedicadas al desarrollo software hoy en día es la reducción de costes y tiempo en los proyectos de desarrollo software. Orientado a otro ámbito este problema también se planteo Henry Ford cuando propuso el paradigma basado en cadena de montaje a principios del siglo XX (1908). Henry Ford propuso una serie de técnicas y estándares para la automatización de fabricación de sus vehículos, este

método lo puso en marcha con el conocido modelo Ford T. Con esto dio comienzo a la era de la producción industrial en serie.

Años más tarde el término conocido como crisis del software fue concebido por Friedrich Ludwig Bauer [1] en la cumbre de la OTAN en el año 1968, en esta cumbre se debatieron cuales eran los problemas que se iban detectando en los proyectos de desarrollo software, entre ellos podemos encontrar los siguientes: Proyectos que no cumplen los presupuestos, ineficiencia del software, software que no cumple los requisitos, código difícil de mantener y componentes no reutilizables.

El paradigma planteado por Henry Ford basado en la cadena de montaje puede ser aplicado al campo del desarrollo software por lo tanto el objetivo planteado para subsanar algunos de los problemas citados anteriormente en este área fue tratar de trasladar el concepto de la cadena de montaje al desarrollo software, esto es automatizar los procesos de producción para obtener líneas de productos software y poder reutilizar componentes software que ya fueron desarrollados ya que el paradigma tradicional se basa en el rediseño de componentes usados con anterioridad.

A raíz de este concepto surge el denominado "Desarrollo Dirigido por Modelos"(MDD). MDD se puede definir como un enfoque de la Ingeniería del software y de la Ingeniería dirigida por modelos (MDE) que utiliza el modelo para crear un producto. ¿Y que es un modelo?. Un modelo se puede entender como la descripción o representación de un sistema en un lenguaje bien definido. Para entender lo que representa un modelo dentro de la Ingeniería dirigida por modelos (MDE) hay que saber previamente lo que es un metamodelo. Un metamodelo es un modelo usado para especificar un lenguaje, básicamente describe las características del lenguaje. Por lo tanto un modelo se puede entender como la instancia de un metamodelo. Estos conceptos son ampliados en siguientes secciones.

El resultado de la utilización de MDD es traducido en reducción de costes debido a que el recurso humano requerido es menor, aumento de la productividad y reutilización de componentes además se puede aumentar el nivel de abstracción a la hora de realizar el diseño del software.

En las siguientes secciones se desarrollan los siguientes aspectos: El apartado 1.2 expande información sobre la Ingeniería dirigida por modelos y el Desarrollo Dirigido por Modelos, este apartado es vital para entender todo lo relacionado con la memoria presente. El apartado 1.3 presenta la motivación y objetivos del proyecto. Finalmente el apartado 1.4 describe la estructura que tendrá el documento presente.

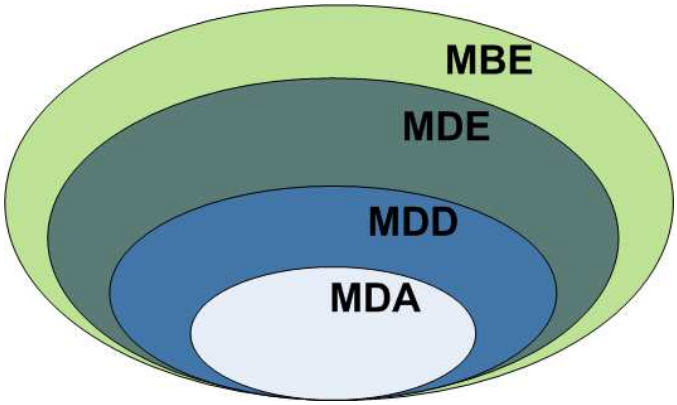

Figura 1.1: Capas del modelado de lenguajes

1.2. Antecedentes: Ingeniería Dirigida por Modelos y Desarrollo Dirigido por modelos

Según [2] la Ingeniería Dirigida por Modelos o MDE puede ser definida como la *"técnica que hace uso, de forma sistemática y reiterada, de modelos como elementos primordiales a largo de todo el proceso de desarrollo. MDE trabaja con modelos como entradas al proceso y produce modelos como salidas del mismo"*. El concepto de Desarrollo Dirigido por Modelos (MDD) es como se puede observar en la (Figura 1.1, etiqueta 1). un subconjunto de la Ingeniería Dirigida por Modelos (MDE). A diferencia de MDD,[3] MDE va más allá de las actividades de desarrollo de puros y abarca otras tareas basadas en el proceso de modelado por ejemplo, aplicar ingeniería inversa a un modelo.

Tanto la Ingeniería Dirigida por Modelos como el Desarrollo Dirigido por Modelos aportan varias ventajas respecto a métodos de desarrollo tradicionales, entre estos podemos encontrar las siguientes ventajas [4]:

1. Tiempo de desarrollo menor: Esto se puede interpretar de la siguiente manera. Al trabajar con modelos lo único que hace falta desarrollar es el generador de código por lo que el tiempo de desarrollo a tener en cuenta es el dedicado al generador de código.

2. Reutilización de componentes en distintos sistemas debido a que trabajamos con modelos.

3. Si se centran los esfuerzos en el generador de código se puede mejorar la calidad del software, además reducimos la cantidad de errores así como el tiempo incurrido en testeos.

4. Eleva el nivel de abstracción pues los problemas que se trabajan son cercanos al dominio del problema.

Otros beneficios que se pueden dar utilizando MDD son descritos por [5] en el ámbito de los repositorios institucionales, los beneficios más importantes que se pueden extrapolar a otras áreas son los siguientes:

1. Un menor número de líneas de código escritas, ya que los niveles de abstracción de MDD a través de los modelos y meta-modelos diseñados fomentan el reusó del código y de los modelos.

2. Alto nivel de abstracción para escribir aplicaciones y artefactos de software a través de la arquitectura de niveles del meta-modelado y las capas de modelado de MDD. Este beneficio favorece diseñar una aplicación partiendo de lo mas general a lo más concreto, es decir, son independientes de la tecnología.

3. Especificación de requisitos de usuario a varios niveles obteniendo un sistema flexible a los cambios.

Una vez entendidos los conceptos anteriores y los beneficios que nos proporciona este enfoque de desarrollo se presentan los conceptos básicos que nos ayudaran a comprender como funciona MDD.

El enfoque del Desarrollo Dirigido por Modelos parte del metamodelo [6], el cual es la representación del sistema en sintaxis abstracta. Un lenguaje de modelado está formado de sintaxis y semántica. La sintaxis son el conjunto de normas o reglas de escritura del lenguaje. La semántica refleja el significado de la sintaxis.

La definición de la sintaxis abstracta es el primer paso en el proceso de diseño de un lenguaje. Especifica la forma del metamodelo. Clases y relaciones de dominio son los principales elementos que forman parte de un metamodelo. Los metamodelos son considerados instancias de estos lenguajes de modelado. Un ejemplo de metamodelo es la definición de UML [7]. En UML por ejemplo elementos como las clases, paquetes o atributos se pueden usar dentro del propio UML.

La definición de la sintaxis concreta es el siguiente paso, este paso consiste en la representación de los elementos bien definidos en la sintaxis abstracta. Véase la creación de un modelo. Un modelo es la representación concreta de los elementos descritos en la sintaxis abstracta o metamodelo. Por lo tanto un modelo siempre posee las propiedades y cumple las restricciones de su metamodelo. En la (Figura 1.2, etiqueta 2) se pueden observar representaciones de los elementos descritos.

Según [6] estos son los pasos que hay que realizar para la creación de un nuevo lenguaje de modelado:

1. En primer lugar hay que establecer cuál va a ser la sintaxis de nuestro lenguaje de modelado. Para ello hay que crear un metamodelo el cual como se comentaba anteriormente establece cuales son las reglas que han de seguir los modelos. El metamodelo ha de ser construido

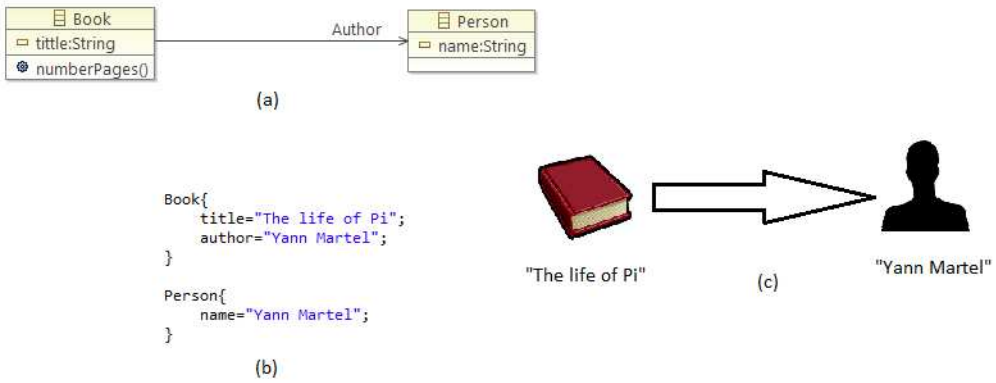

Figura 1.2: (a) Metamodelo, (b) Sintaxis concreta, (c) Sintaxis grafica

utilizando un lenguaje de metamodelado, el lenguaje de modelado utilizado en el presente proyecto es ECore [8]. Además de ECore hay otros lenguajes de modelado como puede ser MOF [9].

2. El siguiente paso consiste en la definición de la sintaxis concreta de nuestro lenguaje de modelado. La creación de un modelo bien definido a partir del metamodelo creado en el paso anterior.

3. La definición de la semántica del lenguaje de modelado es el último paso, por ejemplo crear un generador que transforme los elementos del modelo en elementos de un lenguaje distinto bien definido.

La herramienta con la que se va a trabajar es EMF un plugin de eclipse que nos permite crear y trabajar con lenguajes de modelado. MDE proporciona las transformaciones de modelos como principal herramienta para trabajar con modelos en el proceso de desarrollo software.

1. *Model to model (M2M).* Dado un modelo de entrada esta transformación produce otro modelo como salida. Este tipo de transformación se puede utilizar de varias formas, la primera transformar un modelo dándole propiedades que no posee el original. La segunda es la que se utilizara en el presente proyecto, dado un modelo origen en UML realizar una transformación a un modelo destino escrito en Cassandra (Figura 1.3, etiqueta 3). Esta información es ampliada en el siguiente capítulo.

2. *Model to text (M2T).* Es la ultima transformación que se realiza en la etapa de desarrollo para generar el código de la aplicación. Estas transformaciones no generan un modelo de salida sino una representación textual del modelo que se desea transformar. Se puede utilizar para generar por ejemplo una representación del modelo transformado en formato HTML.

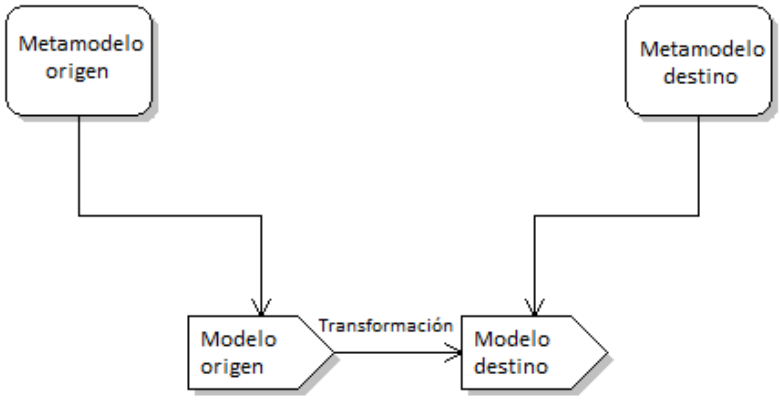

Figura 1.3: Proceso de transformación de un modelo

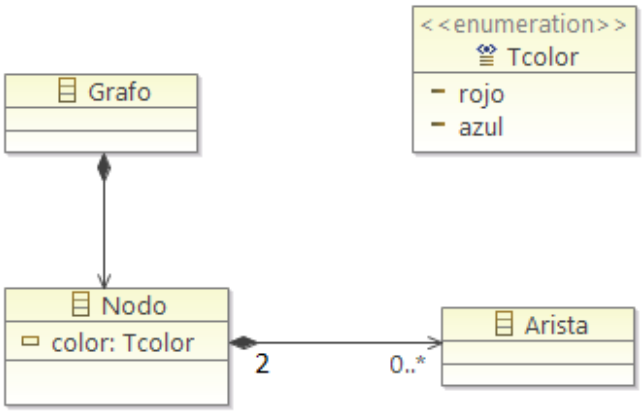

Figura 1.4: Metamodelo de un grafo

Para explicar todo este proceso se presenta un sencillo ejemplo donde se explican cómo se utilizan brevemente las técnicas de desarrollo dirigido por modelos, Model to Model (M2M) y Model to Text (M2T). Para aplicar M2M nos harán falta dos metamodelos (uno origen y uno destino) y para aplicar M2T se ha diseñado un sencillo generador de código HTML.

La (Figura 1.4, etiqueta 4) muestra la representación de la sintaxis abstracta o metamodelo del lenguaje. El metamodelo consiste en un sencillo grafo el cual está compuesto de un número indefinido de nodos los cuales están conectados entre sí mediante aristas. Estos nodos como se ve en la imagen tienen un atributo de tipo "Tcolor"por lo que un Nodo puede ser de color rojo o de color azul. Por lo tanto un modelo bien definido de este metamodelo será un modelo que contenga un grafo con un numero cualquiera de Nodos conectados dos a dos entre sí mediante aristas.

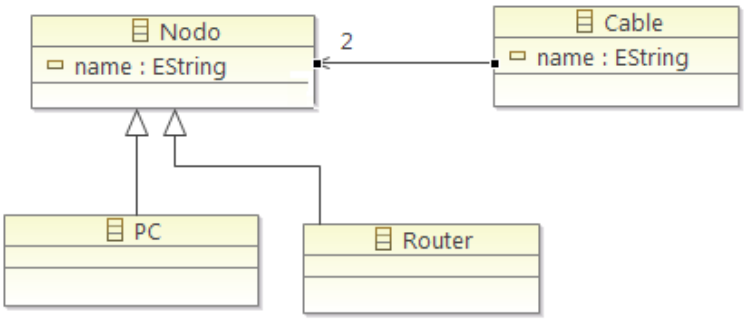

Figura 1.5: Metamodelo de una red

Conexiones

A1	PC1	Router1
A2	PC2	PC3
A3	Router2	PC4

Figura 1.6: Resultado transformación HTML

A continuación se presenta otro metamodelo como se ve en la (Figura 1.5, etiqueta 5). Este metamodelo consiste en la representación de una red de computadores que está compuesta de PC's y Routers estos análogamente al anterior metamodelo están conectados indistintamente entre sí mediante cables. Tanto los nodos como los cables tiene un atributo de tipo "String"para definir el nombre.

Nuestro objetivo es la transformación de un modelo de tipo Grafo a un modelo de tipo Red, dependiendo del color del Nodo el Nodo será transformado en un PC o en un Router (Rojo-PC, Azul-Router) para realizar esta transformación se utilizan técnicas M2M.

Una vez obtenido el modelo de tipo Red deseamos representar textualmente el modelo generado para ello utilizamos técnicas M2T. En este caso la representación del modelo es realizada en formato HTML. La representación del código generado utilizando un modelo de tipo grafo y aplicando la transformación entre modelos es el que se puede observar en la (Figura 1.6, etiqueta 6).

El proceso que se ha seguido es explicado más detalladamente en capítulos posteriores, tanto las técnicas de transformación de modelos como la realización del generador de código así como la creación de los modelos.

1.3. Motivación y Objetivos

Como hemos visto a lo largo de la introducción el enfoque que proporciona el Desarrollo Dirigido por Modelos así como la Ingeniería Dirigida por Modelos nos proporciona múltiples ventajas respecto al desarrollo tradicional por lo que la utilización de este enfoque es apropiada para la realización del generador de código así como para las transformaciones entre modelos.

La utilización de modelos UML respecto a modelos escritos en Cassandra a la hora de especificar una base de datos no relacional proporciona una abstracción para aquellos usuarios que no estén muy familiarizados con el modelado de bases de datos no relacionales. Por lo que la utilización de modelos escritos en UML es ventajosa ya que UML es un lenguaje de modelado bien conocido por toda la comunidad de diseñadores.

Como se comenta al inicio del documento, el objetivo de este proyecto de fin de carrera es la implementación de un generador de código que transforme un modelo UML en su correspondiente código ejecutable en Cassandra. Previa implementación del generador de código habrá que transformar el modelo UML a un modelo escrito en Cassandra. Para ello necesitaremos contar con el metamodelo de Cassandra-CQL y el metamodelo de UML. Dicho generador y dicha transformación se desarrollaran utilizando el enfoque y las técnicas que proporcionan el Desarrollo Dirigido por Modelos. Esta implementación será realizada con el framework que proporciona eclipse de modelado llamado EMF.

Como resultado del proyecto se genera un código escrito en el lenguaje de Cassandra (CQL-Cassandra Query Language) que puede ser ejecutado en cualquier herramienta que soporte dicho lenguaje.

1.4. Estructura del Documento

Tras este capítulo de introducción, la presente memoria del Proyecto Fin de Carrera se estructura tal y como se describe a continuación: El Capitulo 2 describe en términos generales todos los conceptos y tecnologías que son necesarios para comprender el contenido de esta memoria conceptos tales como Cassandra o EMF por ejemplo. El Capitulo 3 describe como se ha realizado la transformación de modelos así como la metodología de transformación de modelos (M2M). El Capítulo 4 describe como se ha realizado el generador de código así como las técnicas y tecnologías utilizadas. Finalmente el Capítulo 5 presenta mis conclusiones tras la realización del proyecto así como posibles mejores de la herramienta. (Puede cambiar).

Capítulo 2

Antecedentes y Planificación

Este capítulo describe las tecnologías y técnicas utilizadas en el desarrollo del presente Proyecto de Fin de Carrera. La primera sección está dedicada a introducir el caso de estudio del PFC presentado, el cual consiste en la realización de un generador de código Cassandra que transforma modelos UML a modelos escritos en Cassandra. La siguiente sección está dedicada a describir en qué consiste EMF el lenguaje utilizado para desarrollar lenguajes de modelado. A continuación se describe la herramienta Epsilon que se ha utilizado para desarrollar el generador de código. La siguiente sección está dedicada a explicar de manera breve algunos conceptos de Cassandra. Finalmente se expone la planificación que ha seguido el proyecto para su realización, desde formación hasta desarrollo.

Índice

2.1. Caso de estudio: Generador de código CQL-Cassandra

2.2. EMF

Eclipse Modeling Framework (EMF) [2.2] es un framework de modelado que nos proporciona la base para la elaboración de lenguajes de modelado.

Para la creación de metamodelos EMF [2.3] utiliza dos modelos de metadatos: Ecore y Genmodel. Ecore contiene la información sobre las clases que

se han definido. Genmodel contiene información adicional para la generación del código, por ejemplo la ruta y la información del archivo. Utilizando EMF se pueden crear metamodelos de forma gráfica muy similar a los diagramas de clases en UML .

EMF permite crear un metamodelo a través de diferentes medios, por ejemplo, XMI, anotaciones Java, XML o UML. EMF proporciona un framework para almacenar la información del modelo.

Un ejemplo sencillo de la utilidad de EMF es el siguiente: Imaginemos que deseamos construir una aplicación para manipular mensajes escritos en XML. El primer paso que daríamos sería empezar definiendo el schema del mensaje sin embargo con EMF se puede trabajar ignorando este nivel . Con EMF podemos crear plugins que generen por ejemplo un diagrama de clases UML a partir de este mensaje XML o directamente generar el código Java que implemente las clases del mensaje XML.

2.3. Epsilon

Epsilon es una familia de lenguajes y herramientas para el desarrollo de software dirigido por modelos, entre estas podemos encontrar: la transformación de modelos, validación de modelos, generación de código entre otras funcionalidades. Epsilon es distribuido a través de la plataforma de modelado de lenguajes de Eclipse.

Como se comentaba anteriormente Epsilon proporciona multitud de lenguajes y herramientas para trabajar con modelos. Los lenguajes utilizados en este proyecto son EOL (Epsilon Object Language), ETL (Epsilon Transformation Language) y EGL (Epsilon Generation Language) también ha sido utilizado EUnit como herramienta para validar el código escrito por el generador de código Cassandra-CQL. Estos lenguajes y herramientas son descritos a continuación.

2.3.1. Epsilon Object Language

[2.1] capitulo EOL EOL (Epsilon Object Language) es un lenguaje de programación imperativo utilizado para crear, consultar y modificar los modelos EMF. Según [libro epislon] se puede considerar el lenguaje como una mezcla de Javascript y OCL, que combina lo mejor de ambos lenguajes. Como tal, proporciona todas las características habituales imperativas que se encuentran en Javascript (por ejemplo, la secuencia de la declaración, las variables, bucles for y while, etc) y todas las características interesantes de OCL como las operaciones sobre las colecciones (por ejemplo Sequence1..5.select (x — x¿3)).

Para entender mejor el funcionamiento de EOL se expone el siguiente ejemplo. Se ha definido el metamodelo mostrado en la figura 2.1 el cual consiste en la representación de una casa y las personas que viven en ella.

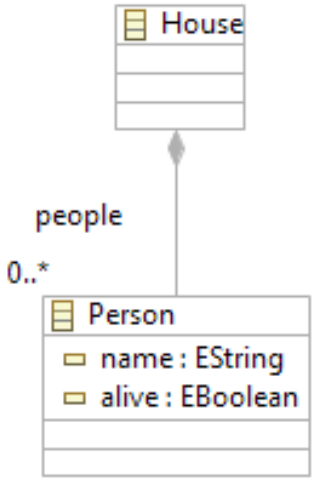

Figura 2.1: Ejemplo metamodelo casa

```
------------------------------------------------------
for (person in Person.all){
  if (person.alive == true) {
    person.name.println();
  }
}

//Podemos realizar lo mismo de la siguiente manera:

Person.all.select(r|r.alive==true).name.println();

------------------------------------------------------
```

Figura 2.2: Ejemplo código EOL

Las personas tienen un nombre y un atributo booleano que representa si una persona está viva o no. Existe una relación de agregación para reflejar que la casa contiene personas.

Una vez creado el metamodelo podemos crear una instancia de ese metamodelo. Un ejemplo de cómo funciona EOL puede ser el siguiente: deseamos saber que personas habitan en la casa y están vivas. La sintaxis sería de la siguiente manera (figura 2.2).

Como vemos la sintaxis es muy similar a cualquier lenguaje orientado a objetos, podemos manipular y consultar los objetos del modelo, sin embargo EOL no nos permite la definición de clases.

```
pre {
    "Running ETL".println();
    var Red : new Red!Red;
    Red.nameRed="Red1";
    var iPC:new Integer=1;
    var iRouter:new Integer=1;
}

rule Arista2Cable
transform a : Grafo!Arista
to r : Red!Cable {
    r.nameCable = a.nombreArista;
    if (a.parent.isDefined()) {
        r.parent=Red;
        for (nodoArista in a.children) {
            if(nodoArista.color=TColor#R){
                var PC : new Red!PC;
                PC.nameNodo="PC"+iPC;
                r.children.add(PC);
                iPC=iPC+1;
            }
            else{
                var Router : new Red!Router;
                Router.nameNodo="Router"+iRouter;
                r.children.add(Router);
                iRouter=iRouter+1;
            }
        }
    }
}
```

Figura 2.3: Ejemplo código ETL

2.3.2. Epsilon Transformation Language

[2.1] capitulo etl ETL (Epsilon Transformation Language) es un lenguaje de transformación modelo a modelo basado en reglas. ETL proporciona las características estándar de un lenguaje de transformación, también nos permite manipular los modelos de entrada y salida así como su código fuente. ETL tiene su propia sintaxis sin embargo utiliza el lenguaje EOL como base.

Recordando el ejemplo de la Red de computadores y el Grafo detallado en el capitulo anterior (sección 1.2). Deseamos realizar la transformación de un modelo de tipo grafo a un modelo de tipo red para ello utilizaremos ETL. En primer lugar hemos de contar con un modelo de un Grafo para poder realizar la transformación a un modelo de tipo Red. La figura 2.3 muestra el código que realiza el proceso de transformación de un Grafo a una Red.

La sección de código inicial define algunas variables previa ejecución de las reglas. En ella se inicializan variables que se utilizaran posteriormente.

En este código encontramos solo una regla. Esta regla transforma aristas del grafo a cables de la red. La primera instrucción copia el nombre de la arista al cable. A continuación la primera condicional cuestiona si esa arista tiene un padre definido en caso afirmativo asigna el cable a la red. A continuación por cada nodo se crea o bien un PC o un router dependiendo del color del nodo que se esté analizando (rojo-PC, azul-Router). En siguiente lugar se asigna el nodo creado al cable correspondiente y finalmente se añade a la red. Este proceso se repite por cada arista del grafo. Una vez ejecutado este código dado un modelo de entrada de tipo Grafo obtenemos un modelo de tipo Red que cumple las reglas definidas en el metamodelo.

2.3.3. Epsilon Generation Language

[2.1 capitulo egl] [2.4] Epsilon Generation Language (EGL) es un lenguaje utilizado para la generación de código basado en la transformación de modelos (Model to Text-M2T). EGL puede ser utilizado para transformar los modelos en varios tipos de lenguajes, por ejemplo código ejecutable Java, código HTML o incluso aplicaciones completas que comprenden el código en varios lenguajes (por ejemplo, HTML, Javascript y CSS).

Cada plantilla de EGL contiene varias secciones. Cada sección puede ser estática o bien dinámica. Una sección estática contiene texto que aparecerá en la salida generada por la plantilla. Una sección dinámica comienza con la secuencia '[%'y termina con la secuencia '%]'. La sección dinámica contiene el lenguaje (EOL, Epsilon Object Language).

La figura 2.4 muestra como se realiza la generación de código HTML utilizando para ello el modelo generado de una red a partir de un grafo (ver sección anterior).

2.4. Cassandra

http://www.nosql-database.org/ [chalmers] Desde que nació SQL en el año 1974, SQL ha sido el modelo de base de datos relacionales utilizado por excelencia. En los últimos años ha surgido otra vertiente denominada NoSQL la cual surge por la necesidad de manejo de grandes volúmenes de información no estructurada, distribuida con la mayor rapidez posible. NoSQL es usado en plataformas como Facebook o Twitter. Podemos encontrar varios tipos de bases de datos no relacionales; Bases de datos documentales, bases de datos clave-valor, orientadas a grafos y mas tipos.

Las principales características de NoSQL que difieren de SQL son:

1. NoSQL no garantiza las propiedades ACID (atomicidad, coherencia, aislamiento y durabilidad).

2. No utiliza SQL como lenguaje de consultas. Algunas bases de datos no relacionales utilizan SQL como lenguaje de apoyo sin embargo la

```
------------------------------------------------------
[%
    var red: Red := Red.allInstances().at(0);
%]

<html>
    <head>
        <title> Red </title>
    </head>
    <body>
        <h1>Conexiones</h1>
        <table  border="1">
            <col style="width: 200px" />
            <col style="width: 100px" span="3" />
            [% for (conexiones in red.conexiones){%]
            <tr>
                <th scope="row">[%=conexiones.nameCable%]</th>
                [% for (nodos in conexiones.children){%]
                    <td>[%=nodos.nameNodo%]</td>
                [% }%]
            </tr>
            [% }%]
        </table>
    </body>
</html>

------------------------------------------------------
```

Figura 2.4: Ejemplo código EGL

mayoría utilizan su propio lenguaje de consultas como por ejemplo Cassandra que utiliza CQL.

3. No está permitido el uso de joins ya que al manejar grandes volúmenes de información una consulta con un join puede llegar a sobrecargar el sistema.

4. Escalan horizontalmente y trabajan de manera distribuida por lo que la información puede estar en distintas maquinas y el añadir nodos mejora el rendimiento.

5. Resuelven problemas de altos volúmenes de información

En este proyecto de fin de carrera se utiliza Cassandra, Cassandra es una distribución de bases de datos no relaciones (NoSQL). Dentro del mundo de las bases de datos no relaciones Cassandra pertenece a la familia de bases de datos denominada çlave-valor". Las bases de datos clave-valor son aquellas que asocian los valores a una determinada clave esto permite la recuperación y escritura de información de forma muy rápida y eficiente. Cassandra reúne las tecnologías de sistemas distribuidos de Amazon Dynamo y el modelo de

datos BigTable de Google. Al igual que Dynamo, Cassandra es consistente. Como BigTable, Cassandra proporciona un modelo de datos basado en ColumnFamily siendo considerado el sistema basado en clave-valor más popular.

Las bases de datos basadas en Cassandra son soluciones utilizadas cuando es necesaria escalabilidad y alta disponibilidad sin comprometer el rendimiento. Escalabilidad lineal y tolerancia a fallos o infraestructura en la nube lo convierten en la plataforma perfecta para datos de misión critica. Cassandra proporciona gran estabilidad en cuanto a la replicación de datos a través de múltiples datacenters, consiguiendo una menor latencia para sus usuarios y la tranquilidad de que no existan perdidas de datos ante caídas. Cassandra utiliza un lenguaje llamado CQL (Cassandra Query Language) con una sintaxis muy similar a SQL aunque con menos funcionalidades.

Fue diseñado por Avinash Lakshman (uno de los creadores de Amazon's Dynamo) y Prashant Malik (Ingeniero de Facebook). Cassandra está en producción para Facebook sin embargo aun está en fase de desarrollo.

A continuación se explican algunos términos que hay que tener en cuenta cuando se trabaja con Cassandra. En Cassandra un KeySpace es el equivalente a una base de datos en los sistemas de bases de datos relacionales. El conocido término de tabla en las bases de datos relacionales tiene su equivalente en Cassandra llamado Column Family. Por lo tanto un KeySpace puede contener varias Column Families y una Column Family a su vez contiene varias columnas. Una columna es la unidad de almacenamiento básica, está formada de tres campos: Nombre, un valor y un timestamp. El nombre y el valor se almacena como una matriz de bytes sin procesar y pueden ser de cualquier tamaño. Un ejemplo: Nombre de la columna ̈Username ̈Nombre de usuario ̈Ignacio ̈Timestamp ̈123456789 ̈La sintaxis del lenguaje CQL es muy similar a SQL, CQL contiene sintaxis ya conocida de SQL como INSERT, DELETE, UPDATE, INSERT, ... Un ejemplo de código CQL ejecutable es el siguiente (Figura 2.5)

2.5. Planificación

Como se ha comentado en el presente documento, el objetivo de este proyecto de fin de carrera es la implementación de un generador de código Cassandra a partir de modelos UML. El proceso de desarrollo así como el de aprendizaje que se ha seguido para la realización del proyecto queda reflejado en la figura [figuraProceso].

La primera tarea como es evidente consistió en adquirir los conocimientos necesarios para el desarrollo del proyecto, en primer lugar todo lo relacionado con el proceso de modelado de un lenguaje y transformación de lenguajes [kleppe]. También fueron necesarios conocimientos sobre la Ingeniería y el Desarrollo Dirigido por Modelos, también sintaxis y arquitectura de Cas-

```
DROP KEYSPACE twitter;

CREATE KEYSPACE twitter
WITH replication = {'class':'SimpleStrategy', 'replication_factor':2};

USE twitter;

CREATE TABLE Tweet(
       UUID text,
       usernameTw text,
       body text,
       PRIMARY KEY(UUID)
);

CREATE TABLE User(
       username text,
       password text,
       followers set<text>,
       followings set<text>,
       tweets_written list<text>,
       PRIMARY KEY(username)
);
```

Figura 2.5: Ejemplo código CQL

sandra.

A continuación se comenzó a trabajar con la herramienta Epsilon, sus lenguajes EOL, EGL, ETL y EUnit como herramienta para las pruebas de los modelos y código generados. Así como con lenguaje para la definición de lenguajes de modelado EMF.

Para conocer cómo funcionaban estos lenguajes se desarrollaron una serie de casos prácticos para familiarizarse con los métodos de transformación así como con la herramienta y creación de casos de prueba.

Una vez conocido estos conceptos se estudiaron las reglas de transformación de un modelo UML a un modelo Cassandra, estas reglas fueron propuestas por [pabloCassandra].

Tras estas tareas de adquisición de conocimientos se empezó a trabajar en el generador de código Cassandra empezando por la transformación de modelos UML a modelos Cassandra. Una vez finalizada la transformación se comenzó a trabajar en el generador de código Cassandra. Tras realizar dicha tarea se realizaron una serie de casos de prueba para verificar si el resultado del generador de código era el esperado utilizando la herramienta EUnit.

Una vez desarrollado el generador de código se realizaron distintos casos de prueba sobre Cassandra para probar el correcto funcionamiento del generador de código.

Pruebas integración

2.6. Sumario

Durante este capítulo se han descrito los conceptos necesarios para lograr comprender el ámbito y el alcance de este proyecto , se ha descrito el caso de estudio planteado en el proyecto también se ha hablado sobre tecnologías implicadas en el desarrollo del generador de código, Cassandra y su arquitectura, Epsilon los lenguajes utilizados y herramientas utilizadas. El siguiente capítulo describe