Visualización de Esquemas en Bases de Datos NoSQL basadas en documentos*

Alberto Hernández Chillón, Severino Feliciano Morales, Jesús García Molina, Diego Sevilla Ruiz

Facultad de Informática, Universidad de Murcia
Campus Espinardo, Murcia, Spain
{alberto.hernandez1,severino.feliciano,jmolina,dsevilla}@um.es

Resumen La ausencia de esquema (schemaless) es una de las características más atractivas de las bases de datos NoSQL debido a la flexibilidad que ofrece. Por ejemplo, es posible almacenar datos no uniformes y se facilita la evolución. Sin embargo, los desarrolladores siempre tienen en mente un esquema cuando escriben código para bases de datos NoSQL y muchas utilidades de bases de datos también requieren el conocimiento del esquema para implementar su funcionalidad. Por esta razón, recientemente se han propuesto diferentes enfoques para inferir el esquema ímplicito en los datos NoSQL almacenados. En este trabajo se presenta una herramienta para la visualización de esquemas NoSQL representados como modelos que son obtenidos por medio de un proceso de ingeniería inversa definido por los autores en un trabajo previo. Estos modelos conforman a un metamodelo Ecore que representa esquemas versionados NoSQL. La herramienta es capaz de mostrar diferentes vistas o diagramas de los esquemas que han sido ideados para favorecer la comprensión de algún aspecto del esquema, por ejemplo mostrar un esquema global con todas las versiones de entidades y las relaciones entre ellas.

Keywords: NoSQL Databases, NoSQL schema, NoSQL Schema Visualization, Sirius

1. Introducción

En los últimos años ha crecido el interés por los sistemas de gestión de bases de datos NoSQL y continuamente aumenta el número de empresas que las utilizan o que lo harán en el corto plazo [3]. Las modernas aplicaciones surgidas, principalmente, con la Web 2.0 tienen requisitos exigentes de escalabilidad y rendimiento en el manejo de grandes volúmenes de datos complejos, las cuales no pueden ser satisfechas por los sistemas relacionales. Para satisfacer estos requisitos han ido surgiendo, a lo largo de esta década, más de doscientos sistemas no relacionales [1] que son agrupados bajo el término *NoSQL* ("Not only SQL").

* Work partially supported by the Cátedra SAES of the University of Murcia (http://www.catedrasaes.org), a research lab sponsored by the SAES company (http://www.electronica-submarina.com/).

En realidad estos sistemas son un reflejo de diferentes paradigmas de modelado de datos, entre los que destacan: *clave-valor*, *basados en documentos*, *familias de columnas* y *basados en grafos* [7].

La mayoría de sistemas NoSQL comparten algunas características entre las que destaca que no es necesario definir un esquema que describa la estructura de los datos almacenados, dado que almacenan datos semiestructurados. Esta ausencia de esquema (*schemaless*) justifica la mayor flexibilidad de estos sistemas frente a los relacionales. Entre las ventajas se destaca que (i) es posible tener datos no uniformes (algunos campos de una misma entidad pueden ser opcionales o tener direrentes tipos) y (ii) se facilita la evolución de los datos dado que no es preciso cambiar ningún esquema [7].

Pero la realidad es que, aunque no es preciso la especificación de un esquema, los datos siempre son conformes a un esquema sólo que en este caso está implícito en ellos y en el código que los maneja. De hecho, los desarrolladores siempre deben tener en mente ese esquema implícito cuando escriben sus programas o usan mecanismos como los índices. Mientras los esquemas permiten realizar un control estático de errores, cuando éstos no existen esa responsabilidad recae en los programadores. Por otra parte, muchas utilidades de bases de datos también requieren conocer el esquema de los datos para implementar la funcionalidad ofrecida, por ejemplo motores de consulta y visualizadores de esquemas. Por ello, recientemente se han propuesto algunos enfoques para el descubrimiento y extracción de esquemas NoSQL [6,8,13].

Un informe reciente de Dataversity [3] ha subrayado la importancia del modelado para bases de datos NoSQL y señalado tres tipos de utilidades que serán esenciales en un futuro inmediato: (i) *Visualización de modelos*, bien creados directamente o inferidos de los datos; (ii) *generación automática de código* a partir de esquemas, por ejemplo validadores de datos; y (iii) *manejo de metadatos* en escenarios como la integración de herramientas.

En este trabajo se presenta una herramienta de visualización de esquemas NoSQL inferidos a partir del proceso descrito en [8]. Estos esquemas son obtenidos como modelos conformes a un metamodelo Ecore [11]. La herramienta proporciona un conjunto de diagramas que ofrecen diferentes vistas que facilitan la comprensión de los esquemas inferidos. Cabe destacar que la visualización considera la existencia de versiones de las entidades que componen la base de datos. Para la implementación se ha utilizado *Sirius* que es una herramienta que genera editores visuales a partir de un metamodelo y la definición de la notación gráfica.

La contribución de este trabajo es doble: se identifican un conjunto de tipos de esquema para bases de datos NoSQL y se proponen diagramas para su visualización, los cuales muestran las entidades, versiones de entidades y relaciones entre ellas. En la actualidad, y en la medida que sabemos, solo la herramienta *ER/Studio* ofrece una visualización de esquemas NoSQL, y solo para el sistema *MongoDB*. También *CA ERwin* ha anunciado esta funcionalidad en sus productos pero todavía no es soportada [12]. En cualquier caso, estas herramientas no contemplan la existencia de versiones, lo cual es una diferencia significativa con

nuestro trabajo. Por tanto, entendemos que este trabajo contribuye significativamente a la visualización de esquemas NoSQL.

Este artículo se ha organizado de la siguiente forma. Primero se definen los tipos de esquemas NoSQL que hemos identificado. Luego se describe el metamodelo usado para representar las entidades y versiones en un esquema. A continuación se presentan los diferentes diagramas o vistas que soporta la herramienta creada. Más adelante se comentan los trabajos relacionados más relevantes. Finalmente se presentan las conclusiones y trabajo futuro.

2. Esquemas para bases de datos NoSQL

Un dato semiestructurado está formado por una tupla de uno o más pares *clave-valor*. Nosotros nos referiremos a estas tuplas como *objetos*. Las claves (también denominadas *campos*) denotan propiedades de los datos y los valores pueden ser: (i) atómicos (números, *strings* o *booleanos*), (ii) otros objetos embebidos, (iii) una referencia a otro objeto que será un string o un entero que coincide con el valor de un campo del objeto referenciado (similar a *joins* en bases de datos relacionales), y (iv) un array de valores, que puede ser homogéneo o hetereogéneo. Utilizaremos el término *objeto raíz* para denotar a un objeto que no es embebido en ningún otro.

El término *agregado* se usa normalmente para denotar a una estructura formada por un objeto raíz que tiene objetos embebidos. Por ello, el término "*bases de datos orientadas a la agregacion*" (*aggregation-oriented databases*) ha sido propuesto para nombrar a la categoría de bases de datos NoSQL que almacenan agregados en diferentes formatos (clave-valor, documentos y familia de columnas) [7]. Estos tres tipos de sistemas NoSQL son los más extendidos y en ellos se opta por los objetos embebidos frente a las referencias para facilitar la escalabilidad. Dado que los datos se encuentran distribuidos a través de clústers, las referencias pueden suponer contactar a distintos nodos remotos de un determinado clúster. Este trabajo se centra en base de datos de documentos pero el enfoque podría extenderse para los otros dos tipos.

La Figura 1 muestra un extracto de la base de datos que utilizaremos como ejemplo y que registra películas y directores. Los objetos se han expresado en JSON que es el formato comúnmente usado en bases de datos NoSQL para insertar y extraer datos. Como vemos las entidades *Movie* y *Director* son objetos raíz y *Criticism*, *Prize* y *Media* son objetos embebidos.

Un dato semiestructurado se puede representar como un árbol con la siguiente estructura [4]: (i) los nodos hoja son valores atómicos, (ii) la raíz y los nodos intermedios no son etiquetados y corresponden a objetos embebidos o arrays. Las aristas son etiquetadas con el nombre de los campos. Una raíz o un nodo intermedio tendrán un nodo hijo por cada campo del objeto asociado, y en el caso de arrays un arco por cada elemento contenido que se etiqueta con un entero que denota el índice del array. La Figura 2(a) muestra el árbol para los objetos de la Figura 1 (realmente es un grafo dirigido etiquetado debido a la existencia

```
{ "rows": [
  { "type": "Movie",
    "title": "Citizen Kane",
    "year": 1941,
    "director_id": "123451",
    "genre": "Drama",
    "_id": "1",
    "rating": {
      "score": 8.4,
      "voters": 310768
    },
    "prizes": [
      {
        "year": 1941,
        "event": "Oscar",
        "names": [
          "Best original screenplay"
        ]
      },
      {
        "year": 1941,
        "event": "NY Film Critics",
        "names": [
          "Best screenplay"
        ]
      }
    ],
    "criticisms": [
      {
        "journalist": "R. Brody",
        "media": "The New Yorker",
        "color": "green"
      }
    ]
  },
  { "_id": "2",
    "type": "Movie",
    "title": "Truth",
    "year": 2015,
    "director_id": "345679",
    "genre": "Drama",
    "rating": {
      "score": 6.8,
      "voters": 12682
    },
    "criticisms":[
      {
        "journalist": "Jordi Costa",
        "media": {
          "name": "El Pais",
          "url": "http://elpais.com/"
        },
        "color": "red"
      },
      {
        "journalist": "Lou Lumenick",
        "media": "New York Post",
        "color": "green"
      }
    ]
  },
  {
    "name": "Orson Welles",
    "directed_movies": ["1"],
    "acted_movies": ["1"],
    "type": "director",
    "_id": "123451"
  },
  {
    "type": "director",
    "directed_movies": ["2"],
    "name": "James Vanderbilt",
    "_id": "345679"
  }
]}
```


Figura 1: Fragmento de una base de datos *Movies* de ejemplo.

de referencias). A continuación introduciremos las nociones sobre esquemas de bases de datos NoSQL que hemos considerado en nuestro trabajo.

Esquema o tipo de un objeto. Un *esquema* (o *tipo*) de un objeto semiestructurado se obtiene al reemplazar los valores atómicos del objeto por un identificador que denota su tipo (en nuestro caso un tipo JSON, esto es *Number*, *String* o *Boolean*). Por tanto, la estructura de un esquema sería la misma que la de los objetos de ese tipo. Esto es, un árbol como el mostrado en la Figura 2(b) cuyos nodos hoja son tipos. Un esquema puede describir tanto un objeto raíz como uno embebido.

Entidades y Versiones de Entidad. Una base de datos almacena datos sobre entidades del mundo real (objetos físicos o conceptos). Nosotros hemos usado el término *entidad* para referirnos a todos los objetos almacenados que representan información de una misma entidad (por ejemplo, *Movie*, *Director* o *Prize*). La ausencia de esquema permite que puedan almacenarse objetos con diferente esquema para una misma entidad. Nosotros nos referiremos como *versión de una entidad* al conjunto de objetos de una entidad que corresponde a un particular

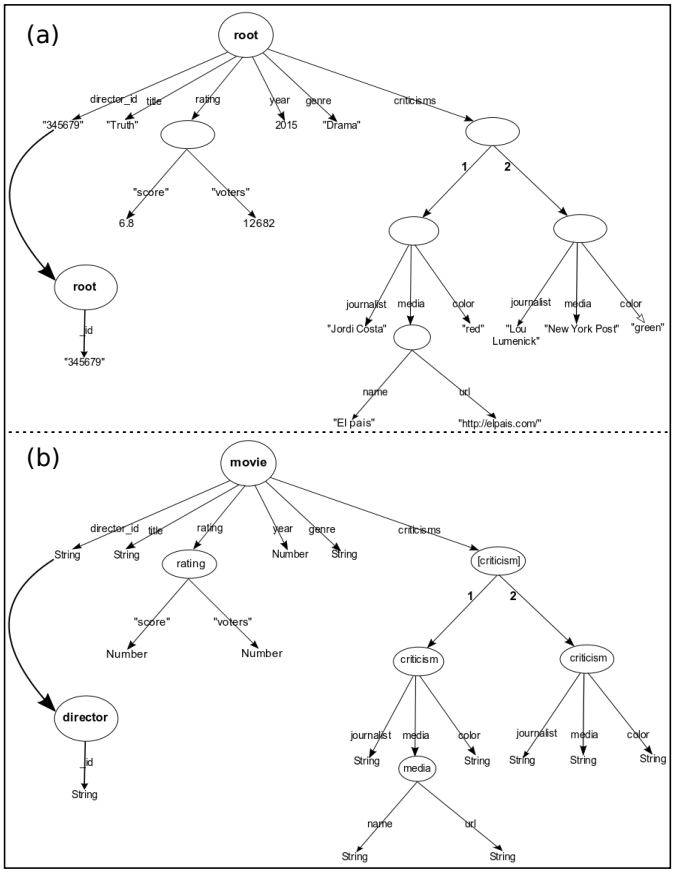

Figura 2: (a) Estructura de árbol para un objeto Movie; (b) estructura de arbol indicando las entidades.

esquema. Por ejemplo en la Figura 1 hay 2 versiones de *Movie*, 2 de *Director*, 2 de *Criticism*, 1 de *Prize* y 1 de *Media*. También hablaremos de entidades raíz (*Movie* y *Director*) y embebidas (*Criticism*, *Prize* y *Media*). Nos referiremos a las versiones de una entidad como el nombre de la entidad seguido del símbolo "_" y un número de versión, por ejemplo *Movie_1* y *Director_2*.

Esquemas versionados. Como hemos visto cada versión de una entidad tendrá su propio esquema que llamaremos *esquema de una versión* para distinguirlo de un *esquema de una entidad*. Nos referiremos a ambos como *esquemas versionados*. En los esquemas de una version los tipos de los campos son otros esquemas de versiones (esto es, versiones de entidad). Por ejemplo, el esquema que corresponde a objetos como Movie_2 es representado en 3.

```
{
    "title": "String",
    "year": "Number",
    "director_id": "ref(Director)",
    "genre": "String",
    "rating": "Rating_1",
    "criticisms": [
        "Criticism_1",
        "Criticism_2"
    ]
}
```

Figura 3: Esquema de la versión *Movie_2*.

```
{
    "title": "String",
    "year": "Number",
    "director_id": "ref(Director)",
    "genre": "String",
    "rating": "Rating_1",
    "criticisms": [
        "Criticism_1",
        "Criticism_2"
    ],
    "prizes": [
        "Prize_1"
    ]
}
```

Figura 4: Esquema de unión de entidad para *Movie*.

Como vemos un *esquema de versión* involucra cuatro clases de tipos: (i) primitivos, (ii) agregación cuando el campo agrega objetos de otra entidad, (iii) referencias (son distinguidas con "ref") cuando el campo referencia objetos de otra entidad, y (iv) arrays que es el único tipo colección considerado (un array de valores atómicos se llamará *tupla*). Usaremos el término *relación* para referirnos a agregaciones y referencias. Nótese que estos dos tipos causan que un esquema tenga referencias directas o indirectas a otros. En la Figura 2(b) se pueden observar estas relaciones para el caso de una entidad *Movie*.

Un *esquema de entidad* está formado por el conjunto de esquemas de sus versiones. Algunas veces interesa tener una única "vista" de todas las versiones de una entidad. Esto se puede conseguir con un esquema unión de una entidad que resulta de la unión de todos los esquemas aplicando las siguientes reglas: (i) se incluyen directamente aquellas propiedades que sólo aparecen en una versión o aquellas que aparecen en más de una versión pero con el mismo tipo, y (ii) cuando aparecen en más de una versión pero el tipo difiere entonces se añade el tipo unión formado por el conjunto de todos los tipos. Por ejemplo, la Figura 4 muestra el esquema unión de la entidad Movie de nuestra base de datos.

Finalmente, definimos el *esquema global de la base de datos* como aquel que consiste en el conjunto de esquemas versión raíz y que incluirá todos los esquemas versión de la base de datos.

3. Metamodelo de esquemas NoSQL

Los modelos que se visualizan son conformes al metamodelo *NoSQL_Schema* presentado en [8] que se muestra en la Figura 5. Un esquema (metaclase *NoSQLSchema*) está formado por una colección de entidades (*Entity*) y cada entidad tiene una o más versiones (*EntityVersion*). Una versión tiene una o más propiedades (*Property*) que pueden ser de tipo *Attribute* o *Association*, dependiendo de si la propiedad representa un tipo (*PrimitiveType* o *Tuple*) o bien una relación entre dos entidades, respectivamente. Una asociación puede ser un agregado

(*Aggregate*) o una referencia (*Reference*). La cardinalidad de la asociación se almacena en los atributos *lowerBound* y *upperBound*, y puede tomar valores 0, 1 ó −1 (este último valor usado para denotar cardinalidad superior a 1).

Una agregación se conecta con una o más versiones (*[1..*]refTo*) porque un objeto embebido puede agregar un array con objetos de distintas versiones. Por el contrario una referencia está conectada a una entidad (*[1..1]refTo*) debido a que se necesita almacenar que una versión referencia a cierta entidad.

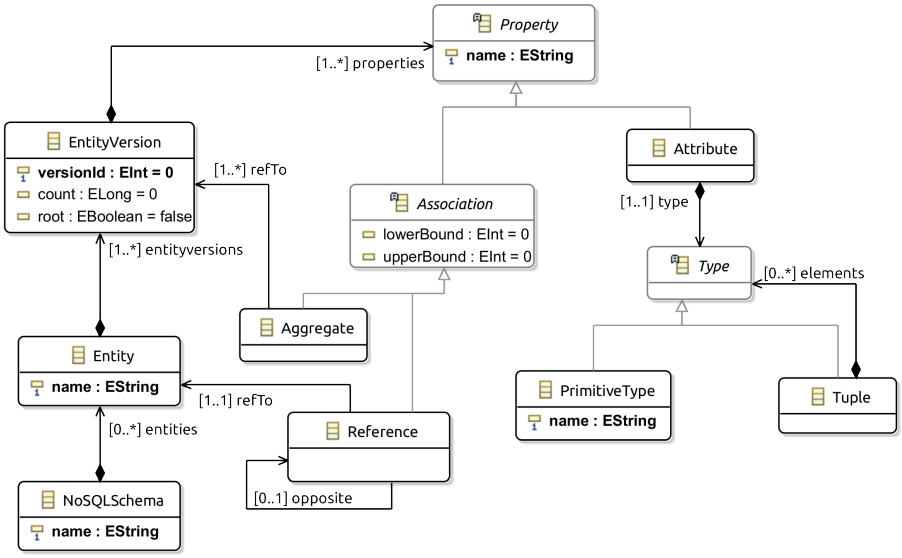

Figura 5: Metamodelo *NoSQL_Schema* que representa esquemas NoSQL.

4. Visualizacion de esquemas NoSQL

La estrategia aplicada para visualizar esquemas NoSQL ha sido crear un editor de modelos gráficos utilizando *Sirius* [2]. Esta solución reduce considerablemente el esfuerzo de desarrollo en comparación a crear la herramienta de visualización desde cero. Dado un metamodelo Ecore, Sirius permite definir de forma sencilla una notación gráfica y los elementos de la paleta de un editor. A partir de esta información, Sirius genera un editor de modelos que pueden ser exportados en formato XMI. Debido a su robustez y potencia, Sirius se ha convertido en la herramienta más usada para crear editores de modelos y lenguajes específicos de dominio (DSL) gráficos en el contexto de EMF (*Eclipse Modeling Framework*). Sirius nos ha permitido crear una herramienta de visualización fácil de usar y con una gran variedad de vistas de los esquemas.

La visualización de esquemas requiere ejecutar primero nuestra herramienta de inferencia que obtiene el modelo *NoSQL_Schema* a partir de los objetos JSON de la base de datos. Esos modelos se pueden cargar en la herramienta de visualización para ir accediendo a sus diferentes vistas y navegando por sus elementos. A continuación comentaremos las principales características de las distintas vistas o diagramas que hemos definido. La herramienta está accesible desde la galería de Sirius[1].

4.1. Vista general de árbol

Como muestra la Figura 6, esta vista consiste de un árbol que tiene tres ramas etiquetadas como *Schema*, *Inverted Index* y *Entities*. *Schema* lista todas las entidades raíz con sus esquemas de versión, y para cada uno de ellos se listan los esquemas agregados o referenciados. En la Figura 6 se muestra que *Movie* tiene 5 versiones y para cada una se muestran los esquemas referenciados desde su esquema versión. *Inverted Index* contiene un índice inverso de versiones, de modo que dado un esquema de version raíz o embebido (por ejemplo, *Director_1*) es posible acceder a todos los *esquema de versión raíz* en los que es referenciado (por ejemplo, *Movie_1*); *Entities* muestra todas las entidades existentes tanto las raíz como las embebidas, y para cada una de ellas lista sus versiones, y para cada versión muestra sus propiedades y tipos. Por tanto esta rama visualiza lo que hemos denominado *esquema de versiones completo de la base de datos*.

Como representación visual se ha optado por diseñar un estilo compuesto por una etiqueta descriptiva y un icono distintivo con el fin de que el usuario pueda, de un vistazo, identificar el elemento que visualiza.

- Las entidades se representan con un icono púrpura pálido con la letra *'E'* y como etiqueta el nombre de la entidad.
- Para las versiones se ha creado un icono amarillo con las iniciales *'EV'* y como etiqueta el nombre de la entidad a la que pertenece la versión seguido del número de versión.
- Para los *esquemas de versión* se ha creado un icono que simboliza una raíz y una etiqueta que informa de qué versión es la *versión raíz* de dicho esquema.
- Para los agregados y las referencias se han diseñado unos iconos con forma de flecha y colores azul y púrpura oscuros, con etiquetas que informan tanto del nombre de la asociación como del destino de la misma.
- Para los atributos primitivos y tuplas se han creado unos iconos rosáceos de pequeño tamaño y etiquetas que denotan el tipo o tipos primitivos a los que se hace referencia, y el nombre de los mismos.

La escala cromática y las etiquetas se han conservado en todas las vistas desarrolladas para facilitar la uniformidad de las mismas y el proceso de aprendizaje al usuario. A partir de la *vista de árbol general* se puede acceder al resto de diagramas a partir de menús contextuales. A continuación comentaremos cada uno de ellos.

[1] `https://eclipse.org/sirius/gallery.html`

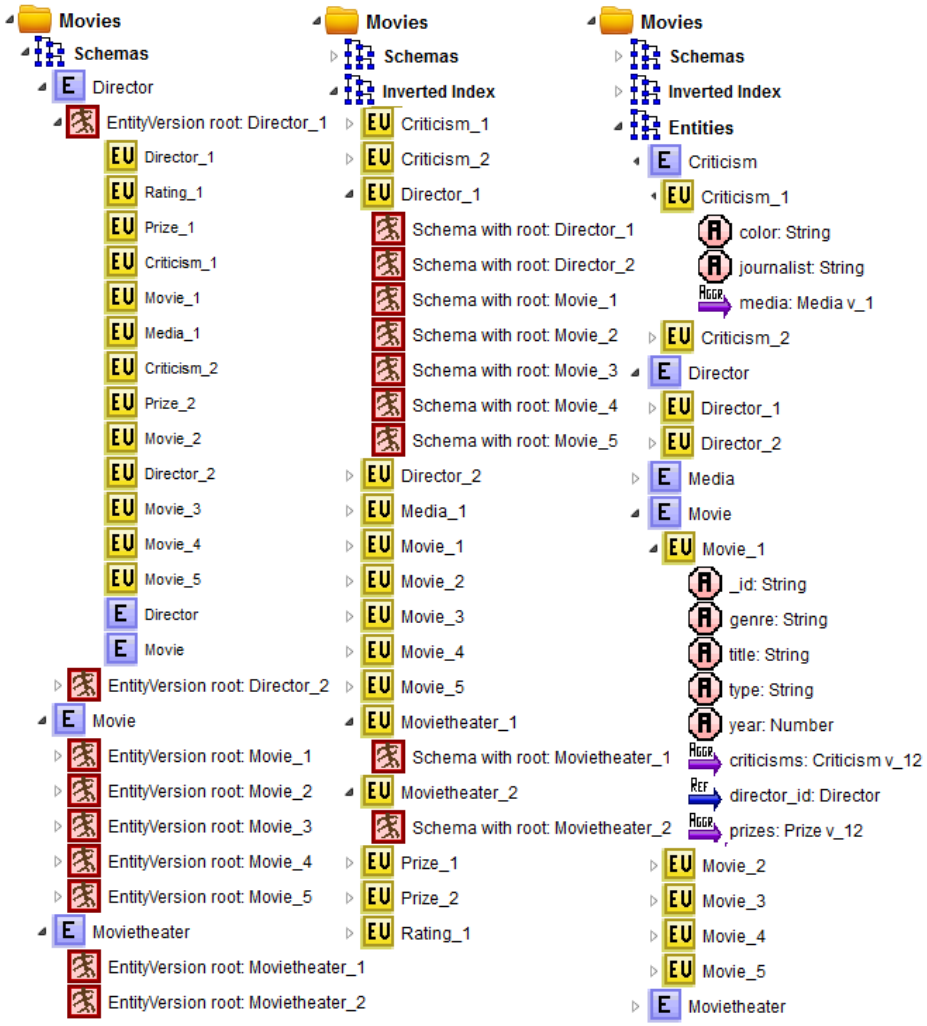

Figura 6: Vista general de árbol sobre *Movies.xmi*.

4.2. Diagrama del esquema global

El *diagrama del esquema global* recoge la información de la rama *Entities* del árbol general. Por tanto, este diagrama muestra el esquema de versiones completo. La forma del diagrama es similar a los diagramas de clases de UML, como se observa en la Figura 7. Su objetivo es mostrar de un vistazo qué entidades incluye un esquema, cuántas versiones existen de cada una de estas entidades, qué atributos tiene cada versión y cómo se relacionan las versiones a través de las relaciones de referencia y agregación.

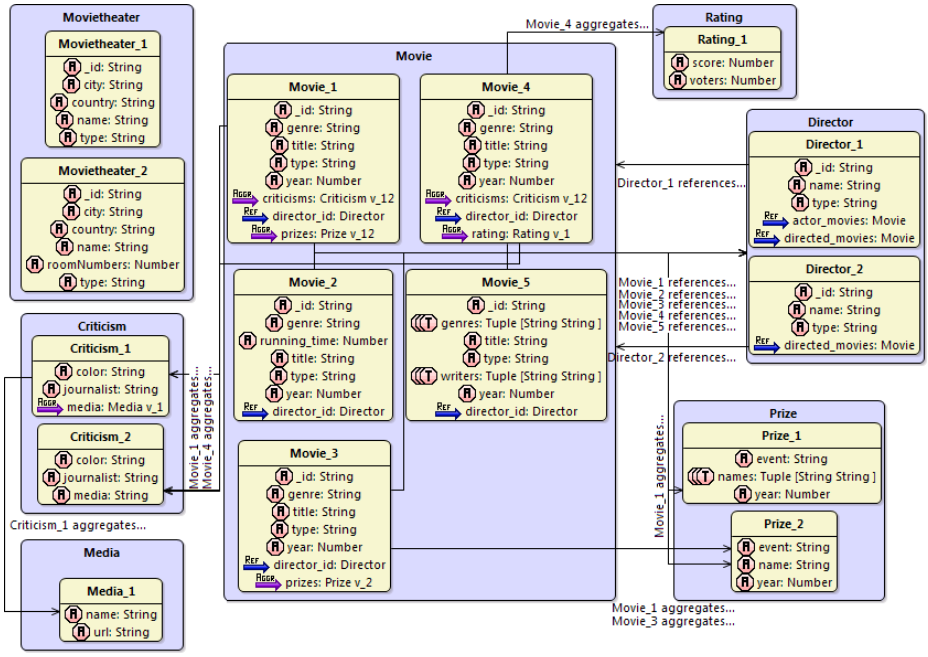

Figura 7: Visualización del esquema global de *Movies.xmi*.

Para el diseño de esta vista se ha recurrido a una escala cromática familiar para el usuario extraída a partir de los iconos de la vista de árbol. En lugar de recurrir a iconos se ha optado por formas geométricas rectangulares de bordes redondeados con un color, una etiqueta y un contenido.

Desde un *diagrama de esquema global* se puede acceder a cualquiera de los dos diagramas de *esquemas de versiones* (plano o anidado), y mediante el menú contextual de alguna *Entitad*, al *diagrama de entidad* de dicha entidad.

4.3. Diagrama de esquema de versión

Se ofrecen dos tipos de diagramas para mostrar los esquemas de una versión raíz: *plano* y *anidado*. Las dos representaciones difieren en cómo se representan relaciones entre los esquemas directamente relacionados. En el *diagrama plano* (Figura 8) estas relaciones se muestran visualmente como flechas, de una forma similar al diagrama global del esquema. En el *diagrama anidado* (Figura 9), en cambio, los esquemas que corresponden a una relación de agregación en el *esquema raíz* se muestran anidados en el interior de éste. Para el diseño de estos diagramas se ha optado por aplicar muchas decisiones de diseño del diagrama anterior. Por ejemplo, se ha mantenido la misma representación para entidades, un color violeta pálido y la etiqueta de cada entidad se forma con su nombre.

Mientras en el diagrama plano las relaciones se distinguen etiquetando las flechas con *references* o *aggregates*, en el diagrama anidado las relaciones se

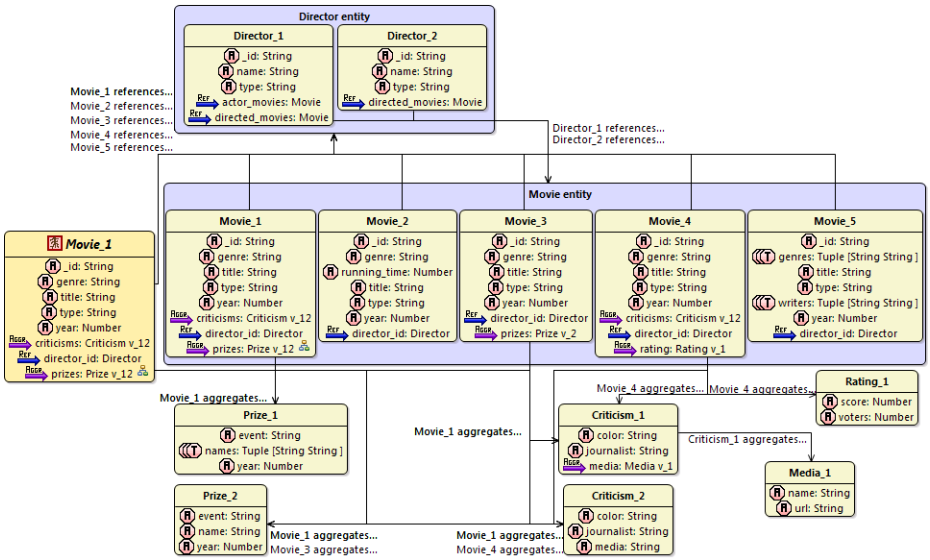

Figura 8: Un diagrama de versión de esquema plano de *Movies.xmi*.

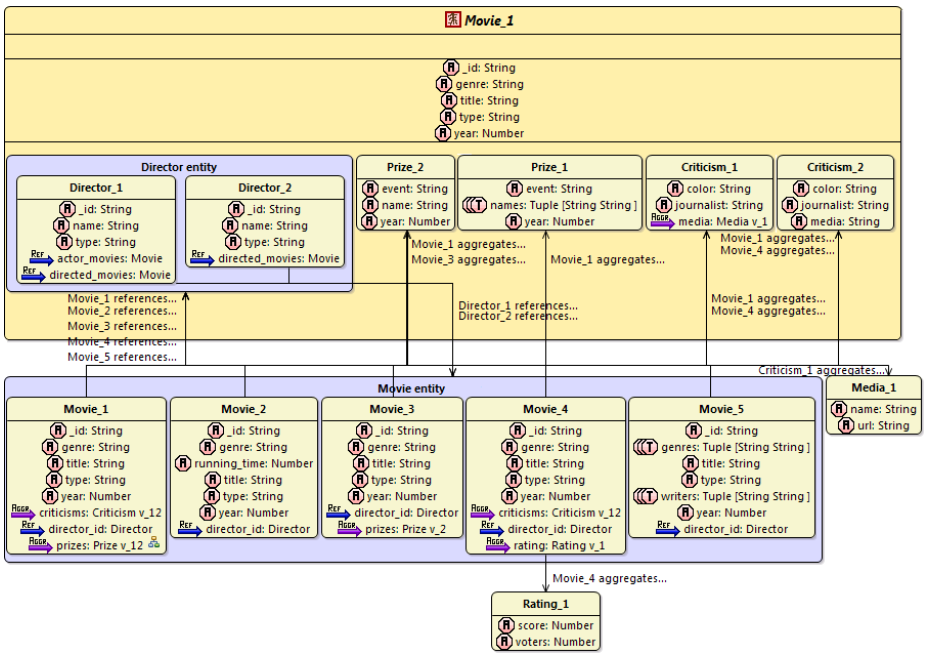

Figura 9: Un diagrama de versión de esquema con anidación de *Movies.xmi*.

expresan físicamente en el interior de la versión raíz que las referencia. Esto hace que la vista anidada resulte más compacta y muestre más claramente el nivel de anidamiento de los elementos asociados.

4.4. Diagrama de entidad

Un *diagrama de entidad* muestra visualmente los esquemas versión de una entidad como rectángulos anidados dentro de un rectángulo que representa la entidad, como muestra la Figura 10. En ella se observa que la entidad *Criticism* tiene dos versiones y se muestran sus esquemas, por ejemplo *Criticism_ 1* agrega un esquema *Media*. Por tanto, estos diagramas representan lo que hemos definido como un *esquema de entidad*.

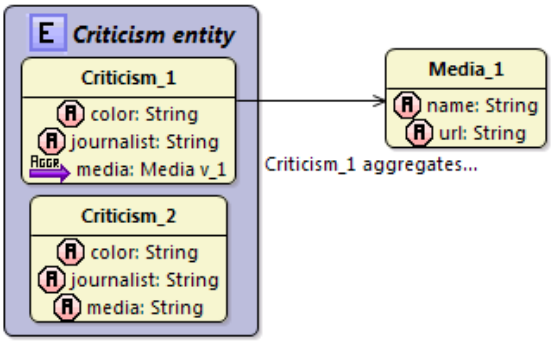

Figura 10: Un diagrama de entidad de *Movies.xmi*.

5. Trabajo relacionado

ER/Studio[2] es una herramienta comercial para modelado de datos que ha sido la primera en soportar modelos de datos NoSQL. En concreto, desde mediados de 2015 incluye la capacidad de mostrar diagramas de esquemas MongoDB que obtiene a partir de un proceso de ingeniería inversa sobre una base de datos. Sin embargo, no considera la existencia de diferentes versiones de entidades y sólo soporta un tipo de diagrama.

CA ERwin[3]es otra conocida herramienta de modelado de datos y de acuerdo con [12] sus desarrolladores trabajan en el soporte del modelado de datos NoSQL. Se utiliza un modelo de datos unificado definido para ERwin (*Unified Data Modelling*) para representar los esquemas de bases de datos relacionales y NoSQL. En el caso de sistemas NoSQL se realiza un proceso de inferencia y se

[2] `https://www.idera.com/er-studio-enterprise-data-modeling-and-architecture-tools`.
[3] `http://erwin.com/products/data-modeler`.

muestran los diagramas de los esquemas obtenidos, aunque no se proporcionan detalles de cómo se llevará a cabo la inferencia y la visualización. Tan solo se menciona que se aplicarán diferentes tipos de técnicas para la ingeniería inversa como *machine learning*, inferencia guiada por el usuario a través de interfaces gráficas y mejora continua de la precisión del proceso de inferencia.

A diferencia de ER/Studio, el enfoque propuesto en este trabajo abarca cualquier sistema NoSQL basado en agregación, y tiene en cuenta versiones de entidades y referencias. En cuanto a ERwin, cabe destacar que el proyecto aquí desarrollado coincide con los objetivos planteados en [12], aunque hay algunas diferencias significativas. Nuestro metamodelo para representar esquemas es más expresivo que el *Unified Data Model* utilizado en ERwin (por ejemplo, permite capturar absolutamente todas las variaciones de tipos, y por lo tanto, toda la información de metadatos de la base de datos), y además permite crear soluciones basadas en la tecnología Model-Driven Engineering basadas en metamodelos y transformaciones de modelos. Por ejemplo, el propio uso de la herramienta Sirius para la visualización.

JSON Discoverer es una herramienta útil para inferir esquemas a partir de documentos JSON [5]. En los esquemas inferidos se tienen en cuenta las relaciones de agregación, pero se obvian las referencias entre entidades. Tampoco se tiene en cuenta la existencia de distintas versiones para una misma entidad. Los esquemas se muestran como diagramas de clases UML, pero la aplicación no está enfocada para ser una herramienta de visualización.

6. Conclusiones y trabajo futuro

Este trabajo ha presentado una de las primeras propuestas para visualizar esquemas de bases de datos NoSQL teniendo en cuenta la existencia de versiones de las distintas entidades almacenadas. Una implementación del proceso de inferencia y de la herramienta de visualización NoSQL se pueden encontrar en un repositorio GitHub[4]. Se han definido varios tipos de esquemas así como diagramas y vistas para su visualización. Cada vista o diagrama está destinado a mostrar información del esquema que interesa al desarrollador, desde un esquema global con todas las entidades y relaciones entre ellas a otros que muestran esquemas de versiones de entidades. El uso de Sirius ha permitido desarrollar la herramienta con un coste menor a su creación desde cero ya que el editor se genera a partir de metamodelo y permite definir la notación gráfica y propiedades del editor de un modo sencillo.

Cabe destacar que esta herramienta iría principalmente destinada a aquellas bases de datos con un número reducido de versiones, como sucede en la mayoría de aplicaciones de negocio. Si existiesen cientos o miles de versiones, como se supone en [13], los diagramas resultarían demasiado grandes y complejos, y por tanto díficiles de manejar.

Para finalizar, se proponen algunos posibles trabajos futuros. La herramienta presentada puede servir de punto de partida para desarrollar una que soporte el

[4] `https://github.com/catedrasaes-umu/NoSQLDataEngineering`.

diseño y visualización de esquemas NoSQL. Esta herramienta incluiría un sencillo lenguaje de consultas para versiones de entidades y capacidades avanzadas de navegación por los esquemas. Además, ofrecería facilidades de generación de código a partir de esquemas (por ejemplo, código para ODM mappers o validadores de datos). También debemos considerar bases de datos clave-valor y familias de columnas. Una evaluación sistemática con desarrolladores NoSQL nos permitirá evaluar nuestra propuesta y definir más líneas de investigación futuras.

En realidad el trabajo realizado aquí es parte de un proyecto más amplio destinado a definir un conjunto de herramientas para aplicar *NoSQL Data Engineering*, como se explica en [9]. En línea con las conclusiones del informe [3], se pretende desarrollar un *toolkit* de herramientas para la visualización de esquemas y datos, la generación de código y el manejo de metadatos. La visualización se ha abordado en este trabajo; la inferencia de esquemas en [8] y la generación de código para usar el *mapper ODM* `Mongoose`[5] en [10].

Referencias

1. Página nosql-databases.org. `https://nosql-databases.org`
2. Página principal de Sirius. `https://eclipse.org/sirius/`
3. Bacvanski, V., Roe, C.: A Dataversity Report. Insights into Modeling NoSQL (2015)
4. Buneman, P.: Semistructured data. In: Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. pp. 117–121. ACM (1997)
5. Izquierdo, J.L.C., Cabot, J.: Jsondiscoverer: Visualizing the schema lurking behind JSON documents. Knowl.-Based Syst. 103, 52–55 (2016)
6. Klettke, M., Störl, U., Scherzinger, S.: Schema Extraction and Structural Outlier Detection for JSON-based NoSQL Data Stores. In: BTW'2105. pp. 425–444 (2015)
7. Sadalage, P., Fowler, M.: NoSQL Distilled. A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley (2012)
8. Sevilla Ruiz, D., Morales, S.F., Molina, J.G.: Inferring Versioned Schemas from NoSQL Databases and its Applications. In: ER, International Conference on Conceptual Modeling. pp. 467–480 (September 2015)
9. Sevilla Ruiz, D., Morales, S.F., Molina, J.G.: Model Driven NoSQL Data Engineering. In: JISBD. Santander (September 2015)
10. Sevilla Ruiz, D., Morales, S.F., Molina, J.G.: A MDE Approach to Generate Schemas for Object-document Mappers. In: Modelsward (February 2017)
11. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: Eclipse Modeling Framework. Addison-Wesley, 2nd. edn. (2008)
12. Wang, A.: Unified Data Modeling for Relational and NoSQL Databases. `https://www.infoq.com/articles/unified-data-modeling-for-relational-and-nosql-databases` (February 2016)
13. Wang, L., Hassanzadeh, O., Zhang, S., Shi, J., Jiao, L., Zou, J., Wang, C.: Schema Management for Document Stores. In: VLDB Endowment. vol. 8 (2015)

[5] `http://mongoosejs.com/`.