Trabajo de Fin de Grado

Modelo de datos eficiente adaptado a la metodología OpenBIM y el estándar IFC

Efficient data model adapted to OpenBIM methodology and IFC standard

Héctor Pedraza Aguilar

La Laguna, 5 de septiembre de 2017

Dña. **Isabel Sánchez Berriel**, con N.I.F. 42.885.838-S profesora Contratada Doctora adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora.

Dña. **Norena Martín Dorta**, con N.I.F. 78.674.114-S profesora Contratada Doctora del área de Expresión Gráfica en la Ingeniería y Arquitectura de la Universidad de La Laguna, como cotutora.

C E R T I F I C A (N)

Que la presente memoria titulada:

"Modelo de datos eficiente adaptado a la metodología OpenBIM y el estándar IFC."

ha sido realizada bajo su dirección por D. **Héctor Pedraza Aguilar**, con N.I.F. 78.636.685-F.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 5 de septiembre de 2017.

Agradecimientos

Aprovecharé este espacio para dar las gracias,

En primer lugar a mis padres y mi novia, por creer en mí y darme las fuerzas para no rendirme, aun cuando no era capaz de ver la salida.

A mis compañeros de trabajo por lo importante que es trabajar con un buen ambiente y por entender mi prioridad hacia mis estudios.

Y, ante todo, a mi tutora (y mi consejera), por seguir dándome oportunidades.

Mil gracias a todos.

Índice General

Índice de figuras

Índice de tablas

Capítulo 1. Introducción

El *Building Information Modeling*[1] (BIM) es un concepto que surge entre finales de los años ’70 y principios de los ’80 y que es frecuente que se use con diferentes sentidos: un tipo de software, un modelo 3D, un proceso de construcción, una base de datos estructural… En cierto modo podemos decir que todos estos conceptos se acercan, pero ninguno lo abarca correctamente. Sin embargo BIM es una metodología para la generación y gestión de datos de edificación que engloba todo el proceso de creación y gestión de un edificio durante su ciclo de vida al completo. Para ello se utiliza software específico de modelado y un modelo de datos que permite abarcar desde la geometría del edificio hasta la información del área geográfica en la que este se encuentra, pasando por las propiedades que tienen cada uno de los elementos que lo componen y las relaciones que se establecen entre ellos.

Partiendo de esto, es necesario un modelo de datos que respalde esta funcionalidad. Debido a esta necesidad nace *Industry Foundation Classes*[2] (IFC) un formato abierto desarrollado por la *International Alliance for Interoperability* a finales del siglo pasado para dar soporte a esta compleja información. La responsabilidad de este proyecto recae sobre su desarrollador, la empresa BuildingSMART[3], quién se encarga de desarrollar el estándar y mejorarlo continuamente para que cada vez dé más respuesta a las necesidades de la metodología BIM.

Una de las cosas que más llama la atención de BIM es que pretende abarcar no sólo la construcción de un edificio, momento en el que, actualmente, se produce una desvinculación total sobre el mismo por parte de los que han trabajado en su creación. BIM pretende ir más allá y abarcar todo el ciclo de vida de las edificaciones, lo que incluye el tiempo posterior a su construcción, es decir, al estado del edificio durante el resto de su vida y las acciones que en él se produzcan, como por ejemplo los mantenimientos del mismo.

La importancia de esta intención es enorme, ya que en la mayoría de los casos parece que el trabajo termina en el momento que se finaliza el proceso de construcción. Esto hace que no se realicen las acciones de mantenimiento que aseguran que el buen estado de un edificio perdure. Se desprecia la utilidad que tiene este tipo de acciones y en la mayoría de los casos no resultan obligatorios, sino meras recomendaciones.

Se aprovechará este trabajo para aportar un pequeño avance en esta fase y facilitar este tipo de labores dada su importancia.

1.1 Objetivos

En todos los comienzos, las primeras preguntas que aparecen son ¿qué es eso? y ¿para qué sirve? Podemos decir que en el caso de BIM, en mayor o menor medida, estas cuestiones tienen respuesta. La información actual acerca de BIM es suficiente para conocerlo y trabajar con él, tanto es así que ya se trabaja en resolver problemas detectados en el uso de este tipo de sistemas.

Por tanto, BIM se encuentra en un punto de transición entre su descubrimiento y la búsqueda de mejoras del sistema. En este punto, son muchas las cuestiones que empiezan a surgir diferenciándose de las primeras en que la respuesta que se busca es mucho más concreta. En este proyecto se han recogido dos de estas cuestiones para intentar arrojar un poco de luz sobre las mismas: ¿Cómo podemos simplificar su uso? y ¿Cuál es la mejor forma de usarlo?

El proyecto estará dividido en dos partes, en la que cada una intentará dar respuesta a una de las preguntas anteriores.

Por un lado, en la búsqueda de simplificación, se investigará la forma de que la estructura BIM se cumpla y funcione de forma totalmente transparente al usuario, eliminando la necesidad de que este tenga conocimientos en estos sistemas para poder operar con ellos.

Por otro lado, para investigar cómo se debe usar de forma eficiente se realizará una investigación de cuál es el modelo de datos que mejor se adapta al funcionamiento de BIM y su estructura, estudiando aspectos como tiempos

de respuesta, facilidad de uso, escalabilidad, consumo de recursos…

En ambos casos se planteará una aplicación final de ejemplo que cumpla la funcionalidad planteada.

Todo el proyecto estará orientado a la gestión de la información implicada en tareas de mantenimiento en edificaciones siguiendo la metodología BIM.

1.2 Alcance

En el presente documento se abordará el estudio del modelo de datos más adecuado a usar en herramientas que sigan la metodología BIM. Para ello se analizará un modelo de datos relacional frente a modelos no relacionales. Una vez que se haya reflejado la estructura IFC en estos modelos, se evaluará su rendimiento desde diferentes aspectos, comparando los pros y contras del uso de uno y otro sistema, y estableciendo un criterio que defina qué modelo de datos es más recomendable usar y bajo qué condiciones (en caso de que las haya).

El proyecto acabará con la implementación de una aplicación Android que sirva como ejemplo de una aplicación de usuario final que haga uso del modelo de datos realizado durante la investigación. Por supuesto, esta permitirá al usuario acceder a los proyectos que se encuentren cargados y registrar los mantenimientos que se hagan a los diferentes elementos del proyecto. El sistema determinará a qué proyectos tiene acceso el usuario y cuáles son los mantenimientos que este puede realizar, así como aportar información acerca de los diferentes mantenimientos que debe recibir cada elemento y la periodicidad en la que deben ser realizados.

De esta forma tendremos una aplicación de gestión de mantenimientos complementada con una aplicación para operarios finales donde consultar y registrar dichos mantenimientos, ambas basadas en la metodología BIM.

1.3 Destinatarios

Los resultados de este proyecto aportarán una solución eficiente a cualquier empresa dedicada a la realización de mantenimientos en edificaciones, tanto por la gestión

de los distintos proyectos que abarquen como por el uso por parte de sus operarios.

Del mismo modo también puede ser de utilidad a cualquier propietario o encargado de una edificación que necesite tener acceso a la información referente a sus propiedades o edificaciones bajo su control. Además, hay mantenimientos de algunos elementos que no requieren de una empresa especializada para realizarlos, por lo que también le sirve como información de cómo debe hacer para mantener dichos elementos en buen estado.

1.4 Antecedentes

Al margen del trabajo de investigación que se desarrolló en la elaboración del TFG, encontramos como antecedentes de la aplicación para la gestión del mantenimiento la herramienta OpenMAINT[4],dedicada a la gestión de edificios, instalaciones, mobiliarios y los correspondientes mantenimientos de los mismos, que además es compatible con las herramientas de diseño que siguen la metodología BIM. OpenMAINT es de libre distribución, bajo licencia AGPL.

Capítulo 2. Modelado eficiente del mantenimiento de edificaciones

Antes de abordar los aspectos tecnológicos y las herramientas utilizadas en el proyecto, debemos estudiar con más detenimiento aquellos aspectos que conforman una parte importante del mismo. En este caso se explicará cómo funcionan las tareas de mantenimiento y el modelo de datos IFC. También, para poner en situación, se comentarán los modelos de bases de datos más utilizados en la actualidad y su funcionamiento general, de forma que se pueda entender su uso posterior en el desarrollo del proyecto.

2.1 Mantenimientos

Antes de comenzar con este apartado hay que comentar que este proyecto no trata de cumplir las normativas referentes a los mantenimientos, ya que éstas son múltiples y de fuentes muy diversas (como especificaremos a continuación). Tampoco, y por los mismos motivos, se han recogido todas las posibilidades de mantenimiento que se contemplan, en algunos casos porque no se ha obtenido dicha información, en otros porque dicha información no se ajusta al modelo de los datos y en otros por simple exceso de información que no supone parte fundamental de la investigación. Aun así se pretende que el estudio refleje lo más fielmente posible el funcionamiento real de una gestión de mantenimientos y que los resultados de este proyecto resulten de la mayor utilidad, aunque son los usuarios los que deben encargarse de la correcta definición y realización de los mantenimientos.

En el caso de los mantenimientos se pueden observar actuaciones muy variadas. Esta gran variedad se debe a que hay varias entidades, a distintos niveles, cuya normativa o recomendación influye directa o

indirectamente en algunos elementos, dependiendo del tipo de elemento que sea.

De esta forma podemos encontrarnos con elementos cuyo mantenimiento es meramente una recomendación realizada por su fabricante o empresa constructora. En otros casos un poco más amplios, los mantenimientos de los distintos elementos pueden venir especificados por una entidad reguladora a nivel local, como por ejemplo la Administración Pública local o un colegio de Ingenieros. Además también es importante que se diferencie en qué casos estas especificaciones son meras recomendaciones o han de cumplirse obligatoriamente (en cuyo caso ha de especificarse también quién ha de encargarse de su cumplimiento y de velar porque así sea).

También tenemos otros elementos que por su importancia tienen un mayor nivel de control. Un ejemplo son todos aquellos elementos relacionados con la seguridad en las edificaciones: alarmas, extintores, conductos de ventilación... En estos casos es normal que nos encontremos con normativas establecidas a nivel estatal y o incluso con normativas a nivel europeo. Teniendo todo esto se hace complicado definir una base de conocimiento de cierta categoría que permita asumir toda esta información, finalidad que no se encuentra como objetivo de este proyecto. Para poder abordar el problema que nos atañe, se ha decidido utilizar un nivel de información que permita entender el funcionamiento que se pretende del sistema que vamos a desarrollar, sin que resulte demasiado complejo en su desarrollo ni demasiado sencillo como para que afecte a la veracidad de la investigación.

Como un ejemplo que sirva de punto de partida, en este proyecto se ha decidido utilizar como fuente de información un manual de uso y mantenimiento de edificios desarrollado por el Colegio Oficial de Arquitectos de Canarias, por la completa información que aporta.

En dicho manual se describen los mantenimientos que deben recibir los distintos elementos comunes en una edificación, así como una descripción completa de los mismos, el tipo de persona u operario que debe realizarlo y la periodicidad con la que debe realizarse. En la Figura 2.1 se presenta un extracto que sirve de ejemplo.

Azoteas\ Intransitables\ Tejados

Operación	Responsable	Periodicidad
Limpieza de canalones, limas, cazoletas, rebosaderos y demás elementos de desagüe, comprobando su correcto funcionamiento	O E	6 meses*
Inspección visual de los faldones, longitud de solape entre piezas, fijaciones de mortero entre piezas, puntos singulares como elementos verticales y chimeneas, tejas rotas, tejas de ventilación, ganchos de servicio y elementos de seguridad, reparando todas las anomalías.	O E	1 año
Comprobación de la estanqueidad y posibles deformaciones de faldones, estanqueidad y funcionamiento de los sistemas de desagüe, estado y capacidad de los ganchos de servicio y elementos de seguridad, juntas y lima tesas de encuentros de faldones con paredes chimeneas y canalones, reparando todas las anomalías.	O E	2 años
Comprobación del estado de conservación del tejado	O E	3 años

*En otoño y primavera

Figura 2.1. Estructura de los mantenimientos[5].

2.2 Modelos de bases de datos

Un modelo de datos[6] es una estructura abstracta que define y documenta cómo se va a organizar una determinada información. En el caso de las bases de datos, el modelo de datos determina la estructura lógica de la misma, así como la forma en la que se almacena, organiza, y manipula la información que esta contendrá.

A partir de esta definición, son numerosos los modelos lógicos que se han desarrollado y que sirven como estructura de funcionamiento para las bases de datos actuales. Sin pretender entrar en detalles de todos ellos, se comentarán brevemente aquellos que se han valorado para la elaboración del proyecto.

2.2.1 Modelo de datos relacional

El modelo de datos relacional[7] aparece en 1970 y es el modelo de base de datos más extendido y utilizado en el mundo. Su estructura está basada en la lógica de predicados y en la teoría de conjuntos.

La idea fundamental de este modelo es el uso de relaciones, también llamadas “tablas”. Cada tabla está formada por un conjunto de datos en el que todos los elementos comparten las mismas características, denominadas atributos o “campos”. Cada fila, por su parte, se correspondería con un registro completo de esa tabla, de forma que debe poder identificarse inequívocamente del resto de registros de la misma tabla.

Del mismo modo, podemos tener varias tablas conformando nuestro modelo, cada una cumpliendo con las características mencionadas anteriormente. Además, las

tablas pueden relacionarse entre sí mediante referencias, en las que se indica que un campo de una tabla “A” hace referencia a otro campo de la tabla “B”. Esto es lo que permite que toda la información de la base de datos esté relacionada entre sí.

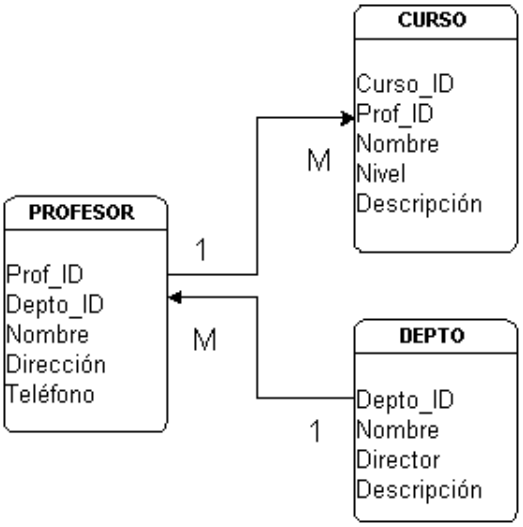

Figura 2.2. Ejemplo de modelo relacional[8].

Como vemos en la Figura 2.2, la tabla “PROFESOR” contiene como campo al “Depto_ID”, relacionando a cada profesor con el departamento al que pertenece. Ocurre de la misma manera entre las tablas “CURSO” y “PROFESOR”.

La gran ventaja de este modelo de datos es la capacidad que tiene para evitar que se produzcan errores en los registros, como información duplicada o datos incorrectos, además de por el sencillo acceso a los datos que permite mediante el uso del lenguaje SQL (Structured Query Language). Por el contrario, esta estructura tan controlada y restringida provoca que las bases de datos relacionales muestren poca eficiencia junto con aplicaciones que utilizan los datos de forma intensiva o que la estructura de los datos que manejan adquiera cierta complejidad.

2.2.2 Modelo de datos orientado a objetos

El modelo de base de datos orientado a objetos[9] está basado en el almacenamiento de la información en forma de objetos completos, lo que incluye tanto su estado como su comportamiento. Este modelo incorpora todos los conceptos importantes del paradigma de objetos y su finalidad

principal es trabajar junto a un lenguaje de programación orientado a objetos (Java, C#, C++…). Pertenece al grupo de modelos de datos denominados como “NoSQL”.

En el modelo, los objetos se almacenan manteniendo la estructura definida para ese objeto en alguno de los lenguajes a los que dé soporte. Esto disminuye el coste del desarrollo y del mantenimiento posterior, por utilizar la misma definición para su uso en la aplicación y en la base de datos. Como contrapartida, no se garantiza la integridad de la información, que debe ser controlada por la aplicación que trabaje con el modelo, pudiendo dar lugar a errores no deseados en la información almacenada.

Además, la rentabilidad de este tipo de sistemas se hace visible cuanto más compleja sea la estructura de los datos que se pretende almacenar, siempre y cuando se adapten a un modelado permitido por la orientación a objetos.

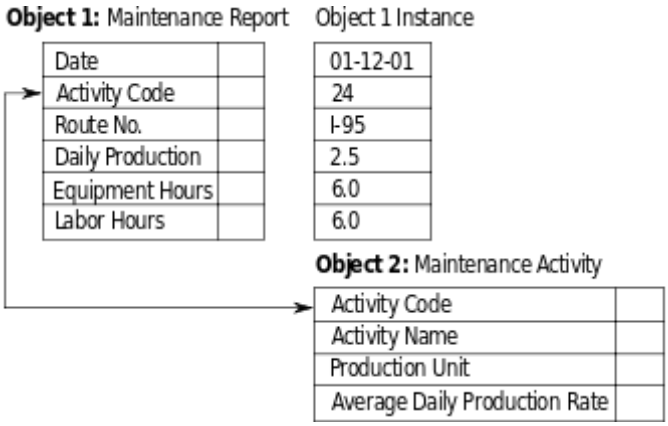

Figura 2.3. Ejemplo de modelo orientado a objetos[10].

Fijándose en la Figura 2.3, se puede ver que se han definido dos objetos que contienen un campo representando la misma información. Se ha instanciado uno de ellos con un valor, mientras que en el segundo objeto no hay ninguna instancia que contenga ese mismo valor. Debe ser la aplicación la que prevenga este tipo de situaciones.

2.2.3 Modelo de datos clave-valor

Otro modelo de base de datos es el denominado clave-valor[11]. Este tipo de base de datos aparece principalmente para dar respuesta a las situaciones en las que una aplicación requiere de un rendimiento

excelente sobre un uso extensivo de la información que se quiere almacenar, que en la mayoría de los casos ocupa un gran tamaño. También se incluye dentro de los modelos de datos “NoSQL”.

El funcionamiento de este modelo se basa en el formato de clave-valor de los datos. Estos pares se organizan dentro de unas estructuras denominadas contenedores, “cabinets” o “clusters”. En cada uno de estos contenedores podemos tener tantos pares clave-valor como queramos y estos pueden estar duplicados o no. Además, los pares dentro de un mismo contenedor pueden tener la misma naturaleza o ser totalmente independientes entre sí, dependiendo de las necesidades de la aplicación.

A cada clave se le asocia un valor, de forma que dentro de ese contenedor tendremos que conocer la clave del dato que queremos obtener para así poder recuperarlo. Si queremos almacenar varios elementos similares, únicamente tendremos que preocuparnos de que la clave de estos no coincida.

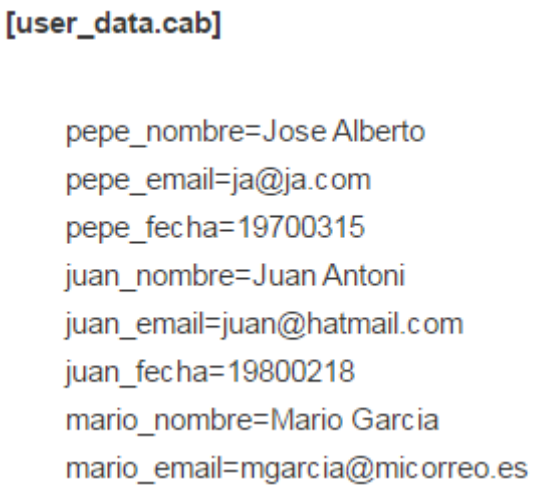

Figura 2.4. Ejemplo de un modelo clave-valor[12].

Tal y cómo se refleja en la Figura 2.4, aunque varios elementos hacen referencia al mismo individuo en la realidad, dentro del modelo clave-valor los datos no tienen ningún tipo de relación más que la otorgada por el significado de la clave. Este tipo de funcionamiento aporta un gran rendimiento a la hora de acceder a la información, además de simplificar su manipulación, pero sacrifica funcionalidades como la integridad de los datos y algunas herramientas de acceso a los mismos. Por ello, no se recomienda utilizar este tipo de modelos para

almacenar datos estructurados o que requieren de un cierto nivel de integridad de los mismos.

2.3 Industry Foundation Classes

El IFC es un modelo de datos que nace en los años 70-80 como respuesta a los requisitos planteados por la metodología BIM. Como ya se comentó en la introducción, BIM persigue un funcionamiento que permita realizar la gestión de las edificaciones, desde que estas se comienzan a diseñar hasta que ya están construidas y perduran en el tiempo. Esto abarca desde la geometría del edificio, representada tanto en 2D como en 3D, hasta información como propiedades de los componentes o incluso información de los costes y recursos consumidos por el mismo. Esto genera el problema de integrar toda la información implicada en cada parte del proyecto.

Con las herramientas CAD tradicionales cada uno de los equipos participantes en el proyecto dispone de las específicas en su campo que le permite realizar su parte, dejando al margen el resto de elementos. Por ejemplo, se puede diseñar los planos del edificio, pero no hay forma de especificar que una pared es de hormigón o que su construcción va a suponer un coste determinado. Del mismo modo, una herramienta que sí puede especificar eso dispone de su propia estructura de información que no permite almacenar los datos estructurales y geométricos.

Con la aparición del formato IFC se pretende resolver este problema: un único formato, estandarizado para todas aquellas herramientas relacionadas con proyectos de edificación, del que todos pueden interpretar, extraer y almacenar la información que necesitan sin que interfiera con la información que han almacenado las demás herramientas participantes en el proyecto. Esto equivale a aportar un lenguaje común para transferir información entre distintas aplicaciones que sigan el formato BIM, logrando así la interoperabilidad de éstas.

Pero a pesar de las enormes puertas que BIM e IFC nos abren, entender el modelo de datos no resulta trivial para aquellos que no son expertos en esta tecnología, siendo muy larga su curva de aprendizaje. Por ello, hay varios aspectos que se deben comentar del modelo antes de entrar de lleno en los datos de la investigación, con el fin de que su entendimiento resulte más sencillo.

2.3.1 Versión de IFC

A la hora de comenzar a trabajar con el modelo de datos IFC, se advirtió que los ejemplos disponibles tenían ligeras variaciones entre sí. Por ejemplo, un mismo elemento en un proyecto tenía una estructura diferente que en otro. En ese momento parecía un problema de modelado incorrecto, pero posteriormente se aclaró que la diferencia era debida a la versión de IFC en la que estaban desarrollados unos modelos y otros.

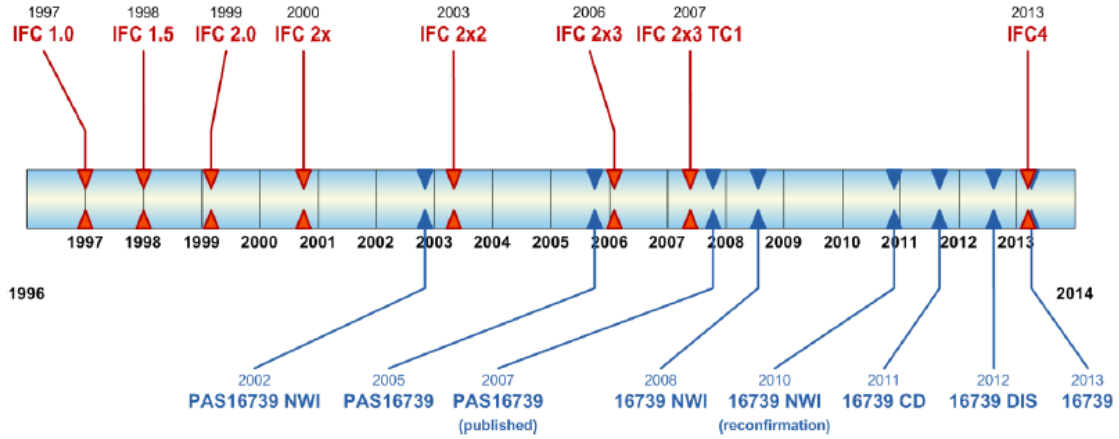

Figura 2.5. Evolución del formato IFC[13].

Como se puede ver en la Figura 2.5, a lo largo de su vida el formato IFC ha recibido varias actualizaciones y mejoras que se han visto reflejadas en diferentes versiones de lanzamiento. Las más relevantes han sido dos de ellas. En primer lugar la versión IFC2x lanzada en octubre del 2000, cuya última corrección de las 5 que recibió fue la IFC2x3-TC1, lanzada en julio de 2007. Al inicio de este trabajo esta corrección sigue siendo la versión de IFC de mayor uso en el mundo.

Y en segundo lugar tenemos la versión IFC4, lanzada en marzo de 2013 tras 6 años de perfeccionamientos (alrededor de 1200 mejoras y correcciones) y documentación. Es la versión destinada a ocupar el estándar de BIM por los próximos años.

Lo primero que llama la atención es la enorme diferencia de calidad entre la documentación de una versión y de otra, lo que ayuda en gran medida a aquellos que realizan una primera toma de contacto con esta tecnología. Además de esto, el cambio de versión supone varias mejoras de las cuales algunas pueden afectar al

proyecto. Entre las más destacadas para este caso, se encuentra la aparición de numerosas entidades que no se contemplaban en la versión anterior o no tenían una representación propia como entidad y que además son susceptibles de recibir mantenimiento. Es el caso de los elementos relacionados con la seguridad del edificio (alarmas, extinción de incendios…), los sistemas eléctricos o los sistemas de ventilación. También se han modificado la mayoría de los elementos estructurales, diferenciándolos entre el propio elemento y su versión estándar (p. ej. Wall y WallStandardCase).

Otro cambio relevante para este proyecto es el realizado en los espacios. Con la nueva versión es posible definir un espacio concreto dentro de otro, sin necesidad que sea un habitáculo definido por 4 paredes, o incluso se puede definir espacios exteriores situados por fuera de la edificación.

También cabe mencionar la posibilidad incorporada en IFC4 de que los ficheros ifc se generen de forma que la representación geométrica del proyecto se base en la geometría por coordenadas, en vez de fijarse a la geometría de sólidos típica de las herramientas de desarrollo CAD. Esto supone un gran avance en la sincronización entre la parte arquitectónica e informática de un proyecto.

Del mismo modo hay mejoras menores que no afectan al alcance de este proyecto pero sí pueden resultar interesantes en trabajos futuros sobre los resultados del mismo.

Debido a todas estas mejoras realizadas y a la vigencia prevista para la versión IFC4, se ha decidido utilizarla como referencia durante el desarrollo del proyecto. Esta decisión ha resultado un acierto, debido no sólo a las mejoras que la nueva versión supone, sino también al hecho de que al comienzo del proyecto la versión más extendida era la IFC2x3, y esta se ha visto desplazada gradualmente por la versión IFC4, cuyo uso es el que se ha extendido actualmente.

2.3.2 Estructura del fichero IFC

Los ficheros del formato IFC siguen una estructura determinada. En primer lugar aparece una cabecera con información relativa al usuario que creó el documento y

las herramientas que utilizó para crearlo. Estos datos no tienen ningún impacto sobre el proyecto que se ha modelado, son meramente informativos.

A partir de la cabecera comienzan las líneas con los elementos que contiene el modelo. Estas líneas siguen una estructura fija, independientemente del elemento al que haga referencia. Comienza con un número de línea, que no ha de ser correlativo. Le sigue en la definición el nombre de la entidad a la que hace referencia la línea. Por último, entre paréntesis y separados por comas, se indican todos los parámetros que definen a esa entidad.

Estos parámetros pueden ser de diferentes tipos: un valor literal, una referencia a otra línea del presente documento (#XXXX), una lista de elementos (literales o referenciados por su línea) un valor enumerado, un símbolo “$”, lo que indica que ese atributo no ha sido especificado en el modelo (equivalente a null) o un símbolo “*”, el cual indica que dicho atributo viene heredado del padre de ese objeto. En la Figura 2.6 que sigue, se puede ver un ejemplo del formato de algunos elementos relevantes para el proyecto.

```
#100= IFCPROJECT('30cdREjkr1sBa6yCk$B9cO',#41,'Project Number',$,$,'Project Name','Project Status',(#92),#87);
#2349= IFCSITE('30cdREjkr1sBa6yCk$B9cQ',#41,'Default',$,'',#2348,$,$,.ELEMENT.,(51,30,0,549316),(0,-7,-34,-450321),0.,$,$);
#110= IFCBUILDING('30cdREjkr1sBa6yCk$B9cP',#41,'',$,$,#32,$,'',.ELEMENT.,$,$,#106);
#119= IFCBUILDINGSTOREY('30cdREjkr1sBa6yCj0qsTV',#41,'Level 0',$,$,#117,$,'Level 0',.ELEMENT.,0.);
#145= IFCSPACE('3nBMDoF1j2gOj$aRM4aoyG',#41,'1',$,$,#128,#141,'Room',.ELEMENT.,.INTERNAL.,$);
#175= IFCSPACE('3nBMDoF1j2gOj$aRM4aoyJ',#41,'2',$,$,#162,#173,'Room',.ELEMENT.,.INTERNAL.,$);
#216= IFCWALLSTANDARDCASE('3nBMDoF1j2gOj$aRM4aoa_',#41,'Basic Wall:Wall-Ext_22Rdr-100Blk-50Air-30Ins-100LBlk-12P:285442',$,'Basic
Wall:Wall-Ext_22Rdr-100Blk-50Air-30Ins-100LBlk-12P:45420',#189,#214,'285442');
#362= IFCWALLSTANDARDCASE('3nBMDoF1j2gOj$aRM4aoaK',#41,'Basic Wall:Wall-Ext_22Rdr-100Blk-50Air-30Ins-100LBlk-12P:285480',$,'Basic
Wall:Wall-Ext_22Rdr-100Blk-50Air-30Ins-100LBlk-12P:45420',#342,#360,'285480');
#396= IFCWALLSTANDARDCASE('3nBMDoF1j2gOj$aRM4aobb',#41,'Basic Wall:Wall-Ext_22Rdr-100Blk-50Air-30Ins-100LBlk-12P:285529',$,'Basic
Wall:Wall-Ext_22Rdr-100Blk-50Air-30Ins-100LBlk-12P:45420',#376,#394,'285529');
#430= IFCWALLSTANDARDCASE('3nBMDoF1j2gOj$aRM4aob5',#41,'Basic Wall:Wall-Ext_22Rdr-100Blk-50Air-30Ins-100LBlk-12P:285561',$,'Basic
Wall:Wall-Ext_22Rdr-100Blk-50Air-30Ins-100LBlk-12P:45420',#410,#428,'285561');
#726= IFCDOOR('3nBMDoF1j2gOj$aRM4aodf',#41,'Doors_ExtDbl_Flush:1510x2110mm:285653',$,'1510x2110mm',#2576,#720,'285653',2.11,1.51);
```

Figura 2.6. Estructura de los ficheros .ifc.

A pesar de esta estructura determinada, la información realmente se encuentra almacenada como texto plano, por lo que es legible desde cualquier editor de texto. Aun así, dada la cantidad tan grande de elementos de los que hace uso IFC para reflejar el proyecto completo, un fichero que represente un modelo simple puede llegar a ocupar miles de líneas, sobre todo cuanta mayor complejidad geométrica tengan los elementos que en él se reflejan.

2.3.3 Estructura del modelo

El formato IFC está basado en una jerarquía de clases en forma de árbol, partiendo este de un nodo raíz del cual heredan todas las demás entidades, directa o indirectamente. Este nodo raíz se denomina *IfcRoot* y actúa como superclase de todas las demás entidades.

A partir de dicho elemento se va desplegando el árbol de jerarquía, comenzando con la diferenciación de los 3 grandes tipos de entidades que recoge IFC: definición de objetos, definición de propiedades y relaciones. Cada uno de estos nodos actúa como definiciones abstractas de los elementos que lo heredan. Con esto, a cada nivel que vamos avanzando por la jerarquía, se van añadiendo atributos a los heredados del padre. Así, cuando llegamos a los nodos hoja, que son los productos en sí mismos, tenemos la definición completa que lo compone.

En la Figura 2.7 se muestra el árbol de herencia de un extintor en IFC. Cómo podemos ver, el número de niveles desde el nodo raíz hasta el propio elemento es amplio. En cada uno de estos niveles por los que pasa la definición hasta llegar el producto se aporta algo al mismo. De IfcRoot hereda los 4 elementos gráficos de todo producto: el id único y global, la referencia con el propietario que creó ese elemento, el nombre y la descripción. En el nivel de la definición de objeto obtienen una serie de relaciones; y así hasta que llegamos al propio objeto, en el que obtiene una diferenciación por tipo.

Inheritance Graph:

```
ENTITY IfcFireSuppressionTerminal
  ENTITY IfcRoot
    GlobalId                        : IfcGloballyUniqueId;
    OwnerHistory                    : OPTIONAL IfcOwnerHistory;
    Name                            : OPTIONAL IfcLabel;
    Description                     : OPTIONAL IfcText;
  ENTITY IfcObjectDefinition
  INVERSE
    HasAssignments                  : SET OF IfcRelAssigns FOR RelatedObjects;
    Nests                           : SET [0:1] OF IfcRelNests FOR RelatedObjects;
    IsNestedBy                      : SET OF IfcRelNests FOR RelatingObject;
    HasContext                      : SET [0:1] OF IfcRelDeclares FOR RelatedDefinitions;
    IsDecomposedBy                  : SET OF IfcRelAggregates FOR RelatingObject;
    Decomposes                      : SET [0:1] OF IfcRelAggregates FOR RelatedObjects;
    HasAssociations                 : SET OF IfcRelAssociates FOR RelatedObjects;
  ENTITY IfcObject
    ObjectType                      : OPTIONAL IfcLabel;
  INVERSE
    IsDeclaredBy                    : SET [0:1] OF IfcRelDefinesByObject FOR RelatedObjects;
    Declares                        : SET OF IfcRelDefinesByObject FOR RelatingObject;
    IsTypedBy                       : SET [0:1] OF IfcRelDefinesByType FOR RelatedObjects;
    IsDefinedBy                     : SET OF IfcRelDefinesByProperties FOR RelatedObjects;
  ENTITY IfcProduct
    ObjectPlacement                 : OPTIONAL IfcObjectPlacement;
    Representation                  : OPTIONAL IfcProductRepresentation;
  INVERSE
    ReferencedBy                    : SET OF IfcRelAssignsToProduct FOR RelatingProduct;
  ENTITY IfcElement
    Tag                             : OPTIONAL IfcIdentifier;
  INVERSE
    FillsVoids                      : SET [0:1] OF IfcRelFillsElement FOR RelatedBuildingElement;
    ConnectedTo                     : SET OF IfcRelConnectsElements FOR RelatingElement;
    IsInterferedByElements          : SET OF IfcRelInterferesElements FOR RelatedElement;
    InterferesElements              : SET OF IfcRelInterferesElements FOR RelatingElement;
    HasProjections                  : SET OF IfcRelProjectsElement FOR RelatingElement;
    ReferencedInStructures          : SET OF IfcRelReferencedInSpatialStructure FOR RelatedElements;
    HasOpenings                     : SET OF IfcRelVoidsElement FOR RelatingBuildingElement;
    IsConnectionRealization         : SET OF IfcRelConnectsWithRealizingElements FOR RealizingElements;
    ProvidesBoundaries              : SET OF IfcRelSpaceBoundary FOR RelatedBuildingElement;
    ConnectedFrom                   : SET OF IfcRelConnectsElements FOR RelatedElement;
    ContainedInStructure            : SET [0:1] OF IfcRelContainedInSpatialStructure FOR RelatedElements;
  ENTITY IfcDistributionElement
  INVERSE
    HasPorts                        : SET OF IfcRelConnectsPortToElement FOR RelatedElement;
  ENTITY IfcDistributionFlowElement
  INVERSE
    HasControlElements              : SET [0:1] OF IfcRelFlowControlElements FOR RelatingFlowElement;
  ENTITY IfcFlowTerminal
  ENTITY IfcFireSuppressionTerminal
    PredefinedType                  : OPTIONAL IfcFireSuppressionTerminalTypeEnum;
END_ENTITY;
```

Figura 2.7. Jerarquía hereditaria de un extintor[14].

Estos atributos que componen a una entidad pueden ser de dos tipos: directos o inversos. Los atributos directos contienen información guardada directamente en la instancia de esa entidad, mientras que los inversos (que por lo general son relaciones) significan que hay otra entidad que puede tener como atributo al propio extintor.

Esta estructura es común para todas las entidades IFC. Por supuesto, en cada uno variarán los atributos que tiene y las relaciones que pueden contenerlo, pero en esencia siguen el mismo criterio.

Pero además de entender la estructura del modelo, en este caso también es relevante entender cuáles son las entidades más significativas para el proyecto que pretendemos desarrollar.

2.3.4 Entidades relevantes

De entre los centenares de entidades de las que dispone IFC, para nuestro proyecto solamente requerimos de unas pocas, que se especifican a continuación[15]:

IfcElement: esta clase abstracta representa una generalización de todos los componentes y productos AEC (“Architecture”, “Engineering”, ”Construction”). Es superclase de todos los productos que modelaremos en nuestro sistema, los cuales se listan a continuación:

Entidad IFC	Representación
IfcBeam	Viga, soportes horizontales.
IfcColumn	Columna, soportes verticales.
IfcCovering	Techo, cubierta.
IfcDoor	Puerta.
IfcFireSuppressionTerminal	Elemento antiincendios.
IfcFurniture	Mobiliario.
IfcLightFixture	Lámpara.
IfcMember	Estructura de soporte genérica.
IfcRailing	Barandilla.
IfcRamp	Rampa.
IfcRampFlight	Tramo de rampa.
IfcRoof	Tejado.
IfcSlab	Suelo, losa, pavimento.
IfcStair	Escalera.
IfcStairFlight	Tramo de escalera.
IfcSystemFurnitureElement	Componentes del mobiliario.
IfcWall	Muro.
IfcWallStandardCase	Muro estándar (sin componentes ni formas complejas).
IfcWindow	Ventana.

Tabla 2.1. Relación de entidades IFC y los productos reales que representan.

IfcSpatialStructureElement: esta clase abstracta representa a todos aquellos elementos espaciales que pueden utilizarse para definir una estructura espacial. Es superclase de los 4 espacios estructurales que se pueden definir:

Entidad IFC	Representación
IfcSite	Terreno o parcela
IfcBuilding	Edificio
IfcBuildingStorey	Planta de un edificio
IfcSpace	Habitáculo

Tabla 2.2. Relación de entidades IFC y los espacios reales que representan.

IfcRelContainedInSpatialStructure: esta relación es la que marca en qué espacio estructural está contenido cada uno de los productos. Es de suma importancia para nuestro trabajo, ya que nos permite establecer una distribución de los productos y separarlos por áreas.

IfcRelAggregates: esta entidad de relación se encarga de enlazar un elemento con otro que lo contiene o que se ensambla con él. Es la otra entidad de relación que necesitamos analizar para nuestro modelo. Por un lado nos permite montar la jerarquía de espacios, pudiendo ubicar cada espacio en cada piso; identificar a qué edificio pertenece cada piso (ya que puede haber más de uno en el proyecto) y cada edificio a qué terreno. Por otro lado, nos permite distinguir aquellos elementos que no son individuales, sino que forman parte de otro más grande. Por ejemplo, un pasamano o un descansillo que forman parte de una escalera.

IfcActor: es la entidad que representa a los actores que forman parte del proyecto. Puede representar a una persona individual, a una organización o a ambas. En nuestro caso se aplica a las personas que actúan en el proyecto y, en la aplicación, a los operarios de mantenimiento.

La importancia de recurrir a las clases abstractas en algunos casos se verá claramente más adelante, cuando entremos en más detalles con los problemas obtenidos y cómo se han ido resolviendo.

Capítulo 3. Tecnologías empleadas

Una vez planteada la situación inicial y los problemas a resolver, toca hablar de las herramientas que se han utilizado y algunas decisiones previas que se han tomado para determinar cómo abordarlo. En algunos casos ha habido que tomar decisiones importantes sobre las herramientas a utilizar por ser determinante para el desarrollo del proyecto, como qué bases de datos utilizar; pero en otros han sido únicamente herramientas de apoyo a la investigación, como los visores de IFC. Vamos a entrar en más detalle de cada uno de los casos relevantes.

Como se podrá observar, la mayoría de estas herramientas disponen de código abierto y acceso completo sin restricciones, salvo algunos casos que especificaremos concretamente.

3.1 Bases de datos a utilizar

En un proyecto destinado a estudiar el rendimiento de dos modelos de datos es natural que aparezca esta cuestión como la primera. Las opciones que aparecen son múltiples, y ha habido varias opciones valoradas como posibles. En un primer lugar se valoró utilizar dos bases de datos relacionales para realizar la comparación de rendimiento entre las mismas, pero posteriormente se pudo observar la viabilidad de utilizar bases de datos NoSQL frente a la relacional y se decidió apostar por ello.

Una vez tomada esta decisión, lo siguiente a decidir era los sistemas gestores de bases de datos que se van a utilizar.

3.1.1 Modelo de datos relacional

Para la base de datos relacional se valoraron dos opciones: MariaDB y PostgreSQL, ambas opciones de código abierto y con soporte continuado. Aunque PostgreSQL ha aumentado su número de usuarios en gran medida, sobre

todo desde que se parara el proyecto MySQL, finalmente se ha decidido apostar por utilizar la herramienta derivada de este último **MariaDB**.

MariaDB[16] es el descendiente natural de MySQL, desarrollado incluso por el mismo autor, y mantenido por él mismo y la comunidad de desarrolladores de software libre continuando con la licencia GPL que corría peligro desde que el proyecto original fue adquirido por Oracle. El proyecto de MariaDB se ha centrado en pulir aspectos clave de MySQL, como los motores de búsqueda o la optimización de consultas, mientras asegura que al menos una parte del proyecto MySQL original se mantiene con código abierto y licencia GPL.

3.1.2Modelo de datos orientado a objetos.

Para la base de datos no relacional se han dado varios pasos a seguir. En primer lugar había que determinar qué tipo de base de datos NoSQL encajaría mejor con la estructura de la información que hay que manejar con IFC. Debido a la naturaleza de las entidades IFC con sus atributos, como primera opción se valoró una base de datos de tipo clave-valor, con Cassandra como opción más viable; u orientada a documentos, con MongoDB como mejor candidato. Posteriormente se vio que las entidades podían entenderse como clases, y cada elemento del fichero IFC como instancias de las respectivas clases. Esto conllevó valorar como gran opción una base de datos orientada a objetos, para lo que se eligió **ObjectDB**.

ObjectDB[17] es una base de datos orientada a objetos destinada al lenguaje de programación Java. Se puede utilizar tanto en modo cliente-servidor como en modo incrustado, aunque en nuestro caso hay que descartar esta última opción. Al estar dedicado a Java, se puede integrar en varios tipos de aplicaciones, desde las destinadas al funcionamiento web hasta servidores de aplicaciones Java EE. También se pueden desplegar en contenedores de servlets. Tiene soporte continuo por la empresa ObjectDB Software y, aunque no es de código abierto ni de uso gratuito, sí ofrece facilidades de uso completo sin coste para estudiantes y proyectos de investigación.

En este caso se ha decidido optar por ObjectDB por parecer el mejor candidato de base de datos orientada a

objetos disponible. Además concuerda con el resto de decisiones sobre las herramientas a utilizar, que comienzan a tender hacia el lenguaje de programación Java.

3.1.3 Modelo de datos clave-valor.

Como sistema gestor para este tipo de bases de datos se ha elegido utilizar **Cassandra**.

Apache Cassandra[18] es una base de datos NoSQL basada en el modelo de datos clave-valor. Fue desarrollada en 2008 dentro de Facebook, aunque en 2009 pasó a ser un proyecto Apache y en 2010 se graduó como proyecto de alto nivel. Está escrita en Java y es de código abierto bajo licencia Apache License 2.0.

Cassandra se caracteriza principalmente por perseguir la escalabilidad lineal y la disponibilidad, permitiendo almacenar enormes volúmenes de datos de forma distribuida sin que su uso y gestión se vea afectado por ello. Aunque los múltiples nodos replicados que conforman la red de datos soportan una redundancia máxima, la sincronización entre los centros de datos se realiza de forma asíncrona, provocando que Cassandra ofrezca también un rendimiento muy elevado pese a la gran cantidad de información que almacena y lo distribuida que se encuentre.

Cassandra está muy extendida en sistemas con grandes volúmenes de datos, como por ejemplo Instagram, Netflix, Reddit o GitHub.

En este proyecto se ha optado por Cassandra como herramienta basada en el modelo de datos clave-valor por ser la herramienta de este tipo más extendida. En este caso, el lenguaje Java no ha marcado la diferencia, ya que el segundo candidato, el sistema Redis, también lo soporta Java (aunque está escrito en ANSI C).

3.2 Lector del formato IFC

Uno de los principales problemas que apareció al comenzar a trabajar con IFC fue el acceso a la información en el fichero. Se comenzó a buscar la mejor manera de acceder a los datos, valorando seriamente la posibilidad de implementar un intérprete o *parser*

específico para este tipo de ficheros que permitiera rescatar esa información.

Al poco tiempo de comenzar en su desarrollo se comienza a buscar alguna alternativa ya implementada. La respuesta se halló en el framework IFC TOOLS PROJECT[19], una librería creada por la empresa APSTEX, participante importante en el proyecto de BuildingSMART. La librería está orientada a Java, lo que concuerda con el resto de decisiones tomadas acerca de este lenguaje, y contiene todas las clases que refleja el esquema IFC, además de todos los métodos de acceso y trabajo con cada una de ellas. El nombre de la librería es **Java Toolbox IFC4** y su uso es totalmente libre.

Utilizando esta librería se resuelven todos los problemas de recuperación de los datos, al estar en perfecta relación con la estructura del esquema IFC4.

3.3 Lenguaje de programación y framework

Con la decisión de usar una base de datos destinada a Java, y el hecho de que la mejor herramienta encontrada para el acceso a un modelo IFC esté escrita en este mismo lenguaje, parece claro que el lenguaje a elegir para realizar el desarrollo de la aplicación será **Java**. Además, estas decisiones están respaldadas por el objetivo de desarrollar la aplicación cliente en plataforma Android.

El proyecto consta de dos desarrollos independientes, por lo que los abordaremos por separado.

3.3.1 Aplicación de testeo

Los requisitos para un sistema que permita realizar el estudio que pretendemos sobre las bases de datos no son muy elevados. El núcleo de funcionamiento de la aplicación estará en el paso de los datos obtenidos del modelo a la base de datos, lo cual no requiere de un entorno gráfico complejo ni elaborado. Por ello se ha decidido utilizar la herramienta **NetBeans** en su versión 8.1 para desarrollar dicha aplicación.

NetBeans[20] es un proyecto de código abierto (Licencias CDDL y GPL 2) desarrollado y principalmente mantenida pos Sun Microsystems (actualmente administrado por Oracle).

La principal ventaja de Netbeans, aparte de la gran comunidad de usuarios y de soporte que la respalda, es que ya incluye la mayoría de módulos necesarios para comenzar un proyecto en Java, lenguaje para el que está orientado principalmente. Es un entorno multiplataforma.

Para la conexión con las bases de datos se ha utilizado la librería específica de uso en el caso de ObjectDB mientras que para acceder a MariaDB se ha utilizado la API de MariaDB con JDBC.

3.3.2 Aplicación cliente en Android

Para el desarrollo de la aplicación cliente en Android se ha decidido utilizar la herramienta **Android Studio**[21], en su versión 2.1, un entorno de desarrollo integrado específico para desarrollar en la plataforma Android. Fue desarrollado por Google a mediados de 2013 como sustituto de Eclipse como IDE oficial para el desarrollo de aplicaciones Android. Su distribución es totalmente gratuita (Licencia Apache 2.0) y es multiplataforma.

Las librerías que debemos usar son las mismas que para la aplicación de estudio: IfcToolBox y ObjectDB o MariaDB en función de los resultados obtenidos en el estudio.

3.4 Visores de IFC

Otro conjunto de herramientas que hemos utilizado son los denominados visores de IFC. Estas herramientas permiten visualizar la información contenida en el fichero con una mejor presentación que la visualizada anteriormente en texto plano. Como mínimo, nos permiten acceder a una representación 3D del modelo, pudiendo tener una imagen clara de lo que representa el proyecto y los elementos que contiene.

En este apartado hemos pasado por varios ejemplos, de los cuales los más destacados son **BimVision**, **BimSurfer** o **IfcViewer**. Pero todos ellos aportan una funcionalidad muy básica, prácticamente limitada a la representación 3D del proyecto, por lo que siempre se acabaron descartando.

Finalmente se opta por **Solibri Model Viewer**[22], desarrollado por la empresa Solibri y de uso gratuito. Esta herramienta no solo permite visualizar el modelo 3D de una forma sencilla, sino que además refleja

perfectamente la estructura jerárquica establecida en los elementos del modelo, permitiendo acceder a la información completa de cada uno de ellos y analizarlos individualmente. Además, la disposición de la información es muy clara y directa, facilitando en gran medida su uso.

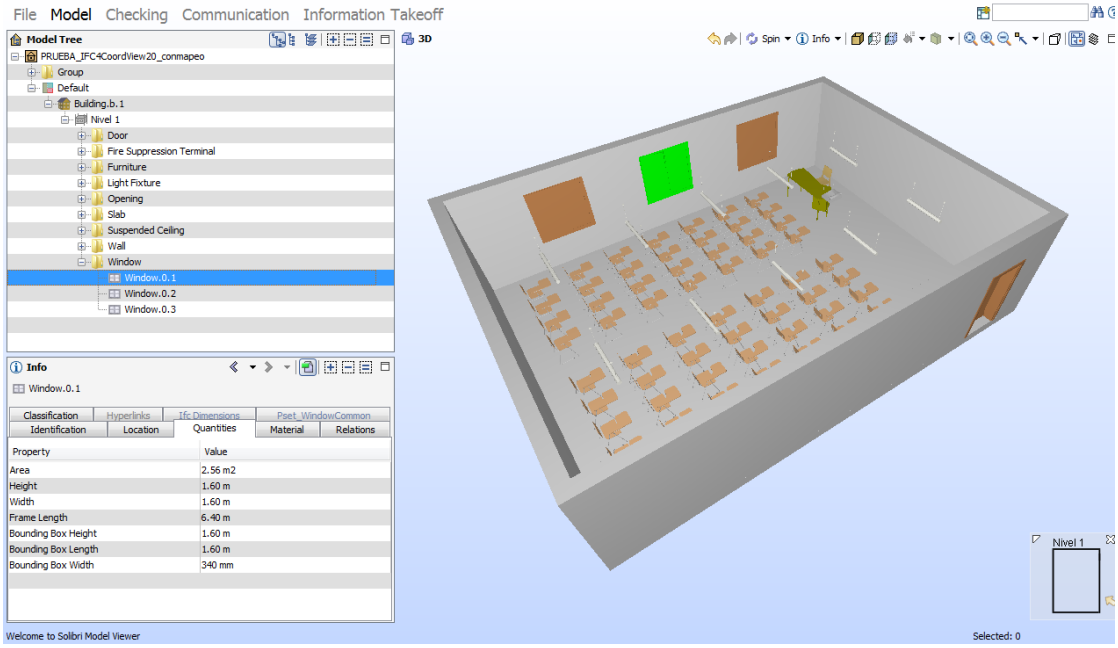

Figura 3.1. Modelo IFC visualizado con Solibri Model Viewer

La utilidad de estas herramientas ha sido de gran valor durante la fase de aprendizaje del proyecto, pudiendo decir que ha sido fundamental a la hora de entender la estructura jerárquica de los elementos y la cantidad de información que se puede almacenar y obtener de cada uno de ellos.

3.5 Herramientas de diseño

Aunque no han sido utilizadas para la realización del proyecto, las herramientas de diseño y modelado son fundamentales para que este tenga sentido. Estas herramientas son las encargadas de generar el fichero IFC a partir del cual cargamos todos los datos en las bases de datos, por tanto debemos aclarar un par de cuestiones acerca de las mismas.

En este caso, la herramienta que se ha utilizado para generar los modelos sobre los cuales se ha trabajado es

Revit. Revit[23] es una herramienta de modelado 3D propiedad de Autodesk orientado principalmente hacia la metodología BIM. Esta herramienta es una de las dos más utilizadas para proyectos basados en BIM, compartiendo esta categoría con ArchiCAD, propiedad de la empresa Graphisoft.

La utilidad de Revit y ArchiCAD con respecto a este proyecto es que ambos permiten exportar cualquier proyecto con este software al formato IFC, a partir del cual se puede extraer la información necesaria de ese modelo. Aun así han surgido algunos problemas que afectan en gran medida al funcionamiento rutinario tanto del desarrollo que se plantea en este proyecto como en las demás aplicaciones basadas en el uso de IFC.

Estas aplicaciones se nutren de un banco de datos interno de la propia aplicación, en el cual se encuentran los modelos de los diferentes elementos que se pueden agregar a un proyecto de edificación (paredes, ventanas, luces…). Los diseñadores agregan desde su banco de datos estos elementos y los representan en el modelo, agregando los parámetros correspondientes a cada uno de los elementos individuales. Los bancos de datos no están regulados por el estándar IFC, por lo que es muy posible que el objeto que se haya modelado no cumpla con el mismo.

Debido a esto, es importante que en el momento de realizar el modelo se sigan estrictamente los esquemas definidos por el estándar IFC y se mantenga un riguroso cuidado en que el modelado inicial se realice bajo los parámetros correctos. En caso de no ser así, los errores que surjan se expandirán a todos los sistemas que se alimenten del modelo generado. Se debe tener en cuenta que el correcto funcionamiento de las aplicaciones que trabajan con este tipo de modelos depende en su totalidad de la correcta realización de los mismos.

Aun así, no toda la culpa recae sobre la persona que realiza el modelo. Como se ha comentado, estas herramientas se nutren de un banco de datos que contienen los objetos a representar en el modelo. Estos objetos pueden venir predefinidos de forma estándar, pero la mayoría de ellos se importan desde cualquiera de los numerosos bancos de datos disponibles en internet, de los cuales destaca **BIMObject**[24] como el más extenso de todos ellos.

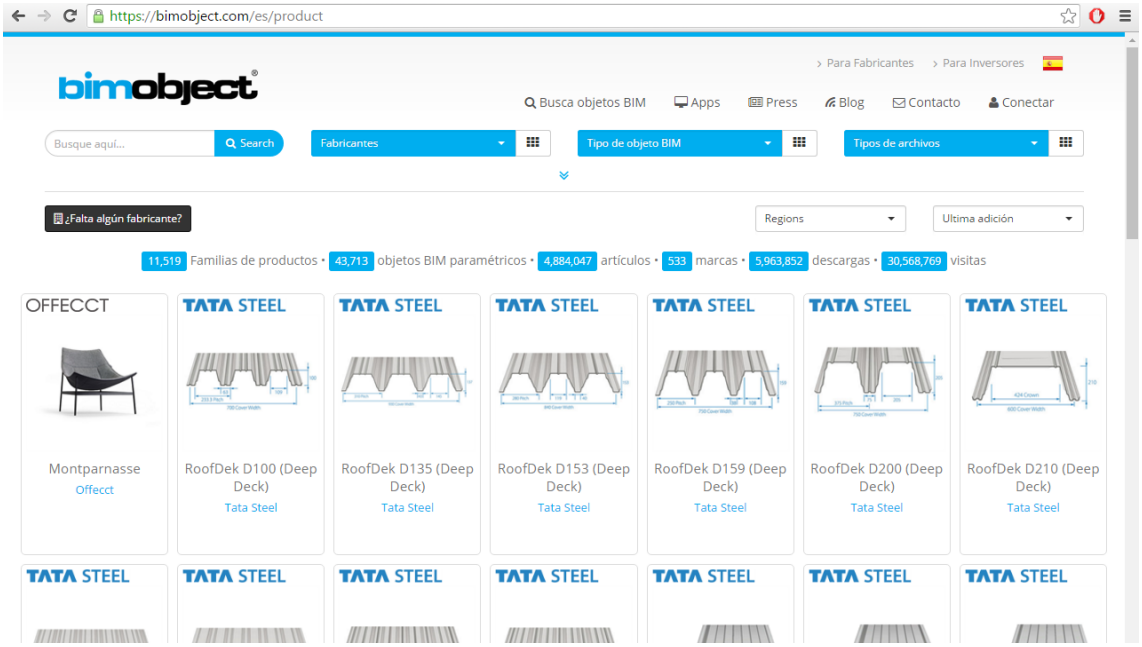

Figura 3.2. Captura del banco de datos BIMObject.

Estos bancos de datos contienen modelos que los fabricantes de distintos elementos añaden al banco para que los usuarios de BIM los utilicen para sus proyectos. Es una forma muy interesante y llamativa, para los fabricantes, de ofrecer sus productos y presentarlos al mercado, por lo que se entiende que su acceso sea totalmente gratuito. El problema viene cuando ese modelo no está correctamente definido. Si un fabricante añade el modelo de su nueva lámpara pero no lo parametriza correctamente, el diseñador que lo incluya en su herramienta Revit o ArchiCAD y lo modele arrastrará ese error.

Por tanto, es importante que se comience a trabajar no solo en el desarrollo del estándar IFC, sino en el cuidado de su correcto cumplimiento.

3.6 Clientes de bases de datos

Para poder trabajar con un motor de base de datos necesitamos de alguna herramienta que nos permita acceder al sistema gestor y diseñar, montar y visualizar las bases de datos que van a sustentar nuestro proyecto.

Para la base de datos MariaDB disponemos del ya conocido MySQL Workbench, perfectamente válido para trabajar con la nueva versión de este motor sin que

produzca ningún tipo de problema. Aun así, la instalación de MariaDB viene acompañada de un cliente distinto, **HeidiSQL**[25], en su versión 9.1. Este cliente supone un total desconocimiento a priori, pero tras unas pocas pruebas, se decide mantener como herramienta de acceso a la base de datos por lo ligero y sencillo que resulta de usar. Aunque funcionalmente se encuentre en clara desventaja con respecto del Workbench, es más que suficiente para lo que se requiere de él en este proyecto.

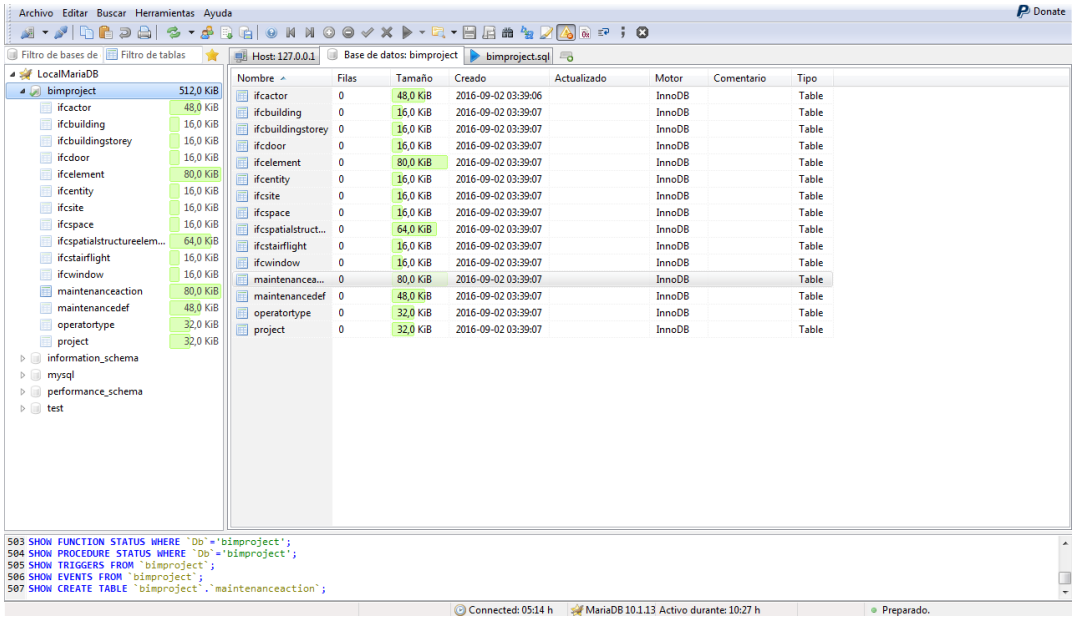

Figura 3.3. Cliente de base de datos HeidiSQL.

En el caso de la base de datos ObjectDB ha sido más sencillo. Dentro del paquete con las librerías viene incluida una herramienta cliente de la base de datos denominada **Explorer**. Su funcionalidad es extremadamente básica, pero suficiente para las funciones que debe cumplir.

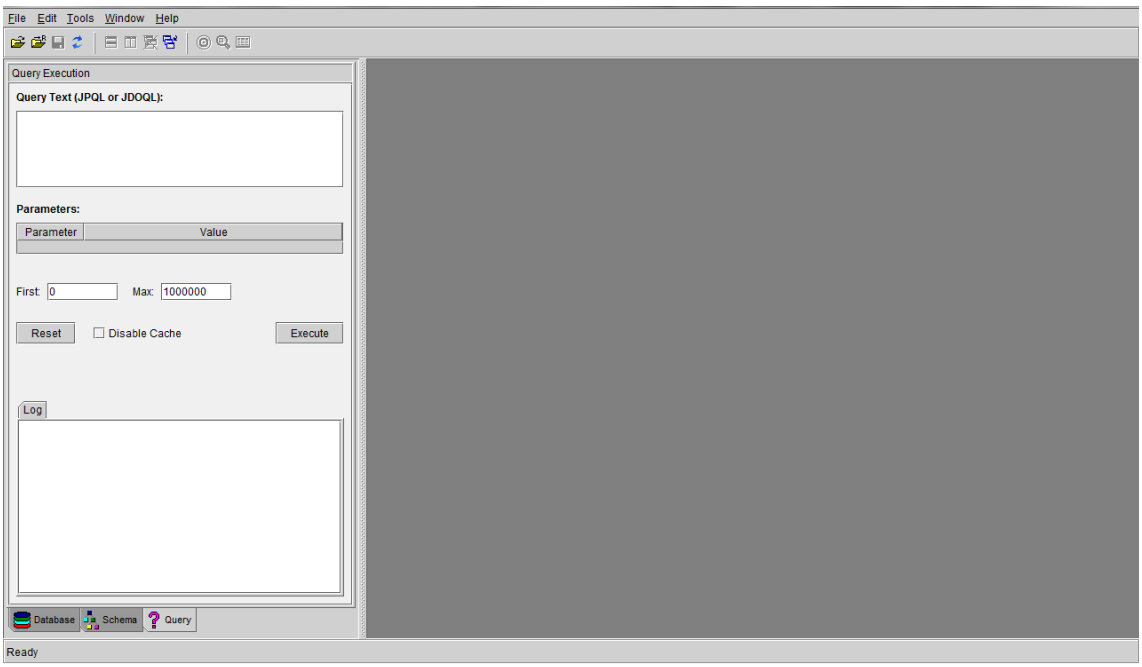

Figura 3.4. Cliente para ObjectDB Explorer.

Por último, en el caso de la base de datos Cassandra, no hay herramientas cliente específicas para esta base de datos, pero podemos utilizar cualquier herramienta genérica de conexión con bases de datos. Un ejemplo válido sería la herramienta DBeaver o RazorSQL (esta última con licencia no gratuita). Además de esto, Cassandra dispone de un lenguaje propio denominado CQL (Cassandra Query Language), muy similar a SQL y que dispone de una interfaz vía comando como la que se usa con MySQL u ORACLE, denominada CQLSH. En este caso únicamente se ha utilizado esta herramienta de comandos para la configuración y gestión de Cassandra.

Aunque Cassandra está desarrollado en Java, el cliente CQLSH está hecho en Python, por lo que únicamente tendremos que asegurarnos de tener instalada la versión de Python correspondiente a la versión CQLSH descargada (en este caso se ha usado la versión 5.0.1, con Python 2.7.13) para poder disponer de una herramienta cliente de una base de datos Cassandra. Además, viene incluido en cualquier descarga de Cassandra.

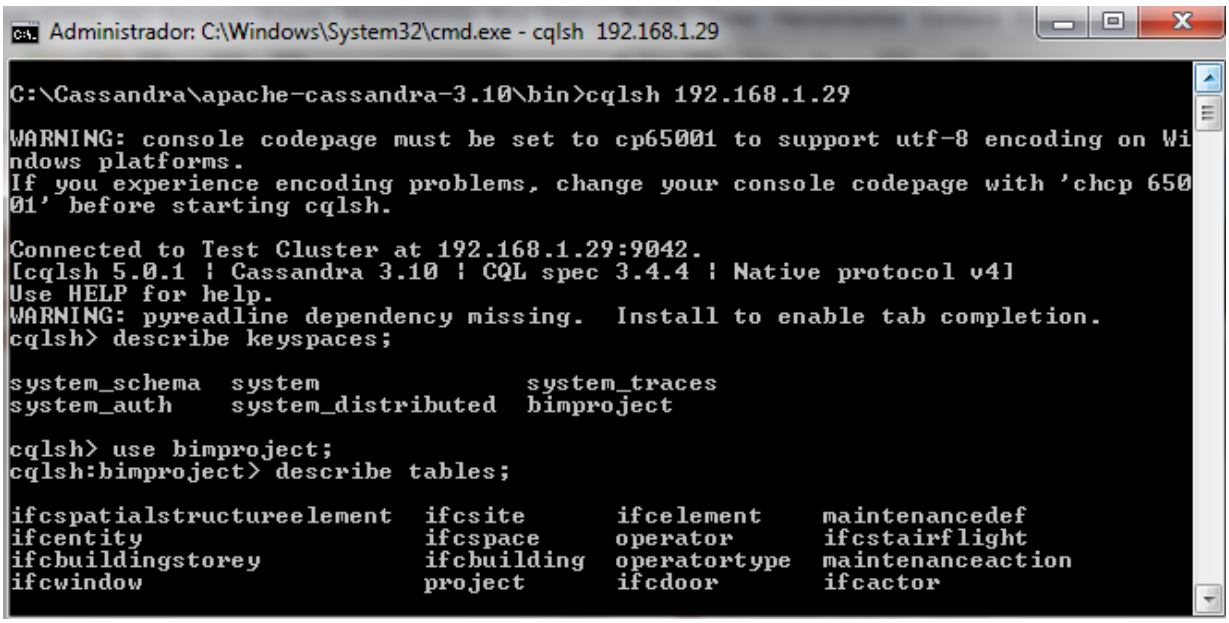

Figura 3.5. Interfaz de la herramienta de línea de comandos CQLSH.

Capítulo 4. Implementación

4.1 Descripción.

Una vez que hemos tomado todas las decisiones previas necesarias para poder arrancar con el proyecto, procedemos a realizar las implementaciones de las herramientas del mismo. Hay que recordar que, en nuestro caso, hay dos implementaciones totalmente individuales.

Por un lado se encuentra la aplicación encargada de testear los modelos de datos, destinada pura y exclusivamente a este fin, sin ningún uso válido posterior salvo el reciclado de los paquetes de clases que la conforman. Por ello, se detallará más adelante todos los diagramas de clases que conforman la herramienta.

Por otro lado tenemos la aplicación en plataforma Android. Esta aplicación sí tiene como objetivo representar un prototipo de un caso real, por lo que se detallará concretamente todos los casos de uso que pueden darse en la misma.

Aunque antes de todo esto hay que comentar el punto más importante (en este caso) de la implementación: el diseño de la base de datos.

4.2 Descripción de la base de datos

Se pretende desarrollar un modelo de datos cuya estructura respete el funcionamiento del formato IFC en la medida de lo posible y permita almacenar los diferentes elementos que en él se modelan.

El funcionamiento principal será destinado a tareas de mantenimiento, por lo que el modelo debe recoger la información necesaria para realizarlos y los objetos que deben recibirlos, así como la propia acción de mantenimiento.

Este mantenimiento debe almacenar el elemento sobre el que se realiza, la persona encargada de su realización,

el tipo de mantenimiento que se ha realizado y la fecha de actuación, con el fin de mantener un control de cuándo debe realizarse el siguiente mantenimiento.

Además, los elementos deben estar organizados por habitaciones, de forma que sea sencillo localizar el elemento concreto sobre el que se va a actuar o se puedan revisar todos los elementos ubicados en el mismo espacio en una única vez.

Por último, el modelo debe permitir almacenar la información de diferentes proyectos, cada uno representado mediante un fichero en formato IFC diferente.

Bajo este planteamiento, la solución a la que se llega se corresponde con el esquema entidad-relación reflejado en la Figura 4.1.

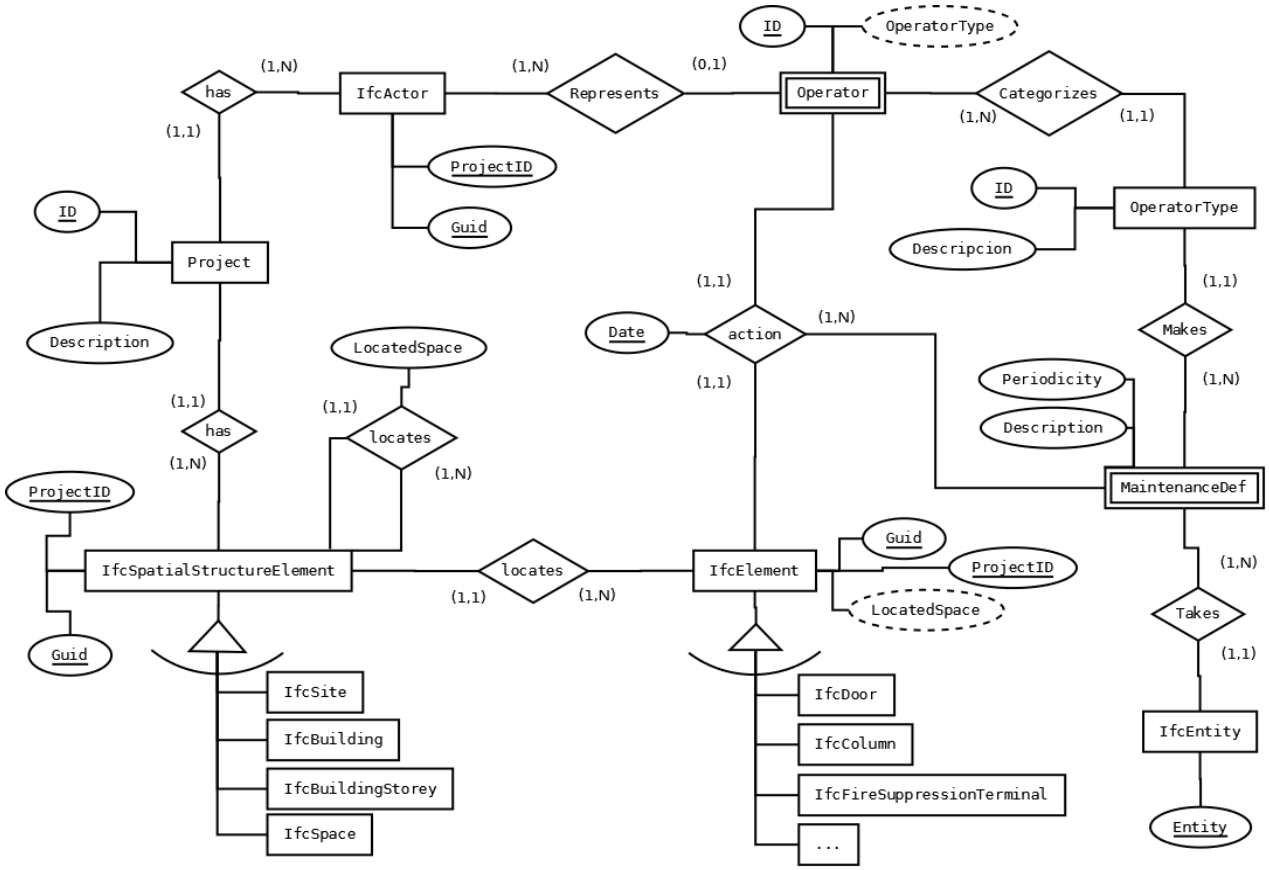

Figura 4.1. Modelo Entidad-Relación.

Dada la gran cantidad de atributos de las entidades IFC se han omitido los atributos menos significativos, aunque se verán reflejados todos en el modelo relacional y en los diagramas de clases respectivamente.

El esquema parece sencillo a primera vista, pero incluye una serie de ciclos redundantes que hay que analizar con detenimiento.

El bucle entre los Operator, los OperatorType y los mantenimientos es uno de ellos. Las acciones de mantenimiento requieren que se defina el tipo de operario que debe encargarse de realizarla. Al mismo tiempo, el operario necesita tener asignado un tipo de operario válido, a fin de que pueda asegurarse de que, cuando vaya a realizar un mantenimiento, esté autorizado a realizarlo. Por tanto, ambas relaciones son necesarias. La conexión entre Operator y MaintenanceDef también es necesaria, ya que la relación Action recibe una información distinta de una entidad que de la otra (de Operator recibe el tipo y de MeintenanceDef su código).

Por el otro lado se produce un caso similar. Tanto los espacios como los actores necesitan estar ubicados en un proyecto concreto. Del lado de los espacios, esta información pasa también a los elementos, que deben estar en el mismo proyecto que el espacio que los contiene. La conexión de MaintenanceAction tanto con Operator como con IfcElement es necesaria, ya que el primero recibe la información identificativa de los otros dos, que no consiste solo en el proyecto en el que se encuentran. Además sirve como método de comprobación de que tanto el elemento a mantener como la persona encargada pertenecen al mismo proyecto.

Una vez resueltos los conflictos, el verdadero problema surge en la representación de las relaciones IS_A del modelo tanto en una base de datos como en otra. En este caso, se ha diferenciado el modo de resolverlas entre un modelo de datos y otro.

En el modelo relacional se definirán las tablas padre, con un campo extra que permitirá identificar el tipo de elemento/espacio al que refiere cada registro. Esto es viable gracias a que todos los hijos tienen atributos comunes. Para los atributos no comunes, se especificarán en una tabla aparte dedicada a ese tipo de elemento concreto.

El motivo de estructurar la información de esta manera viene de que las consultas a realizar sobre los espacios y los elementos tienen como objetivo sus atributos comunes, filtrando por algún criterio independiente del tipo de elemento o estructura espacial que sea. Aunque si

seguimos el esquema IFC deberíamos reflejar cada entidad como una tabla aparte, para nuestro caso resulta poco operativo.

El resultado de las adaptaciones al modelo relacional se refleja en la Figura 4.2.

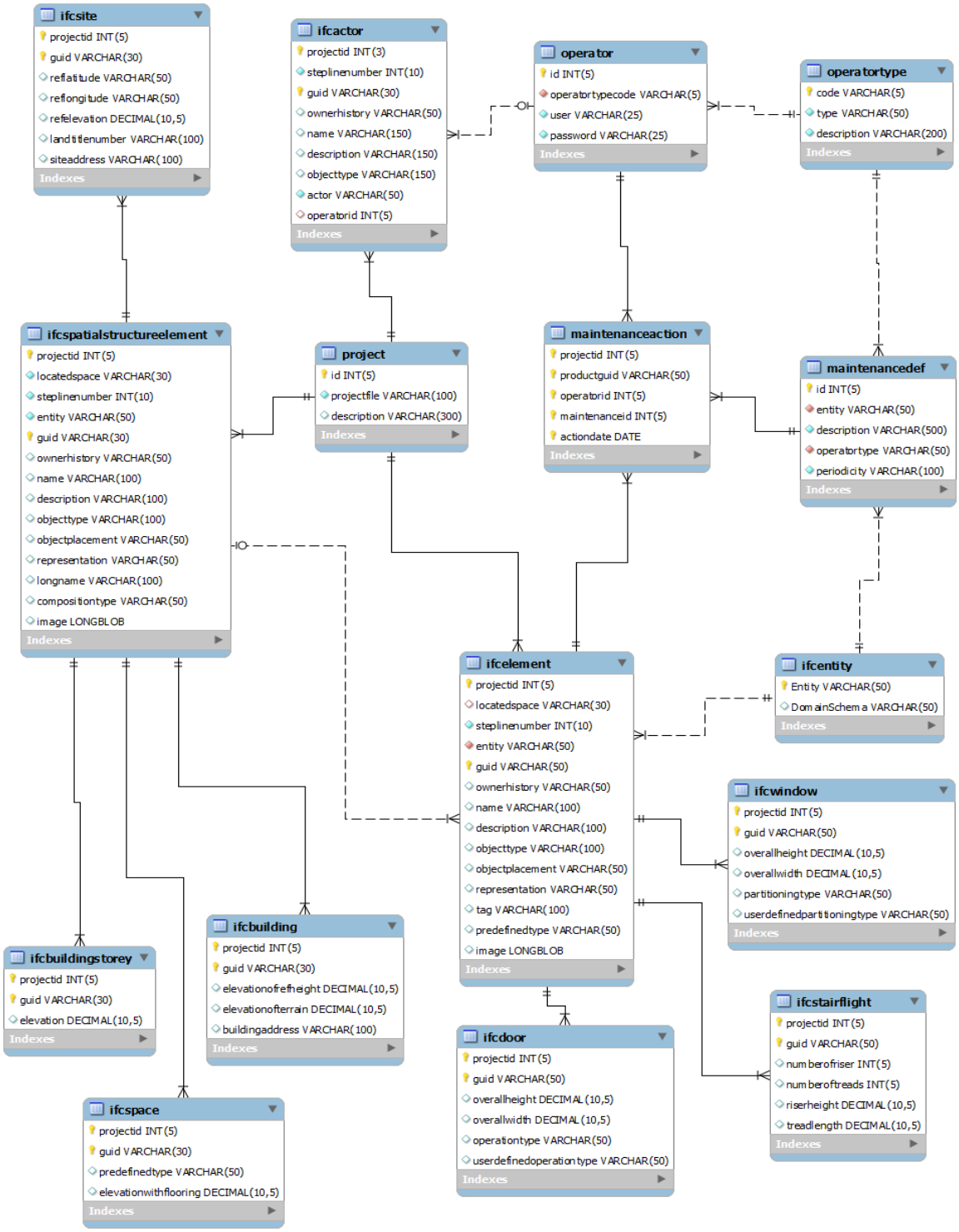

Figura 4.2. Modelo relacional.

En el caso del modelo orientado a objetos, las relaciones IS_A se resolvieron generando una entidad para

cada uno de los distintos elementos que se van a modelar. De esta forma, el modelo sigue la misma estructura que el esquema IFC. En este caso esta implementación resulta viable gracias a una propiedad del paradigma de objetos que es el núcleo de funcionamiento del esquema IFC: la herencia jerárquica.

Siguiendo esto, se ha definido la entidad padre de las IS_A como abstracta, de forma que sirva únicamente para que cada uno de los hijos hereden los atributos y métodos que son comunes a todos ellos. La ventaja de este funcionamiento es que la herencia funciona muy eficientemente en ObjectDB, por lo que podremos buscar entre todos los diferentes elementos de la base de datos haciendo referencia a la entidad padre.

Esta estructura se verá reflejada a continuación, cuando se definan los diagramas de clases.

4.3 Diagramas de clases

En la implementación que se ha realizado se ha fragmentado el código en 4 paquetes separados (sin contar la clase que recoge el entorno gráfico), recogiendo en cada uno clases con un objetivo común. A continuación se mostrarán los diagramas de clases separados por paquetes. Recordemos que estos diagramas se corresponden con la herramienta de testeo de las bases de datos, no de la aplicación cliente.

4.3.1 Paquete *model*

En este paquete se encuentran las clases encargadas de comunicarse con las bases de datos. Contiene 3 clases en total, cada una de ellas con un objetivo muy definido: *Initializer* se encarga de inicializar las bases de datos con la información que no es aportada por el modelo IFC; *Reader* se encarga de recoger el modelo IFC y pasarlo a las bases de datos; y *Tester* se encarga de realizar las pruebas o test con las bases de datos.

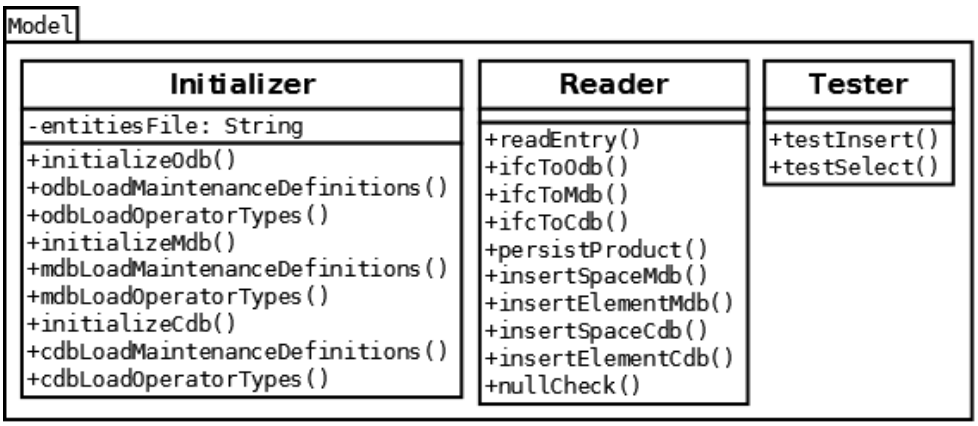

Figura 4.3. Diagrama de clases del paquete Model.

4.3.2 Paquete *sources*

El paquete sources está destinado a contener todas aquellas clases que tienen funcionalidad estricta de comunicación entre las demás clases del programa. Está formado por 2 únicas clases: la clase *Main* que ejecuta el entorno gráfico de la aplicación y la clase *StatsRecord*, que se encarga de almacenar los datos recopilados por la clase *Tester* y pasárselos al frame para que los muestre.

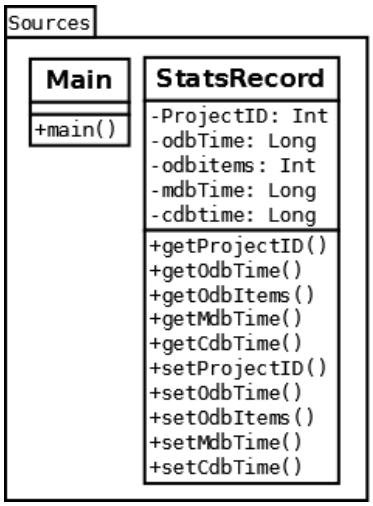

Figura 4.4. Diagrama de clases del paquete Sources.

4.3.3 Paquete *entities*

Para concluir, el paquete *entities*. En este paquete se encuentra el grueso del proyecto, ya que incluye las declaraciones de todas aquellas entidades que serán persistentes en nuestra base de datos orientada a objetos. Contiene un total de 32 entidades, de las cuales 2 son abstractas. Por motivos de espacio se simplificarán las funciones “getter” y “setter” de las clases en el siguiente diagrama.

Como se puede observar en la Figura 4.5, todas las entidades que representan elementos de IFC se han denominado por el nombre de su entidad seguido de la palabra “Entity”. El motivo de esta nomenclatura no es otro que distinguir las clases propias del programa que representan a los elementos IFC en nuestra base de datos orientada a objetos de las propias clases del modelo IFC, contenidas todas en la librería IfcToolBoxIFC4.

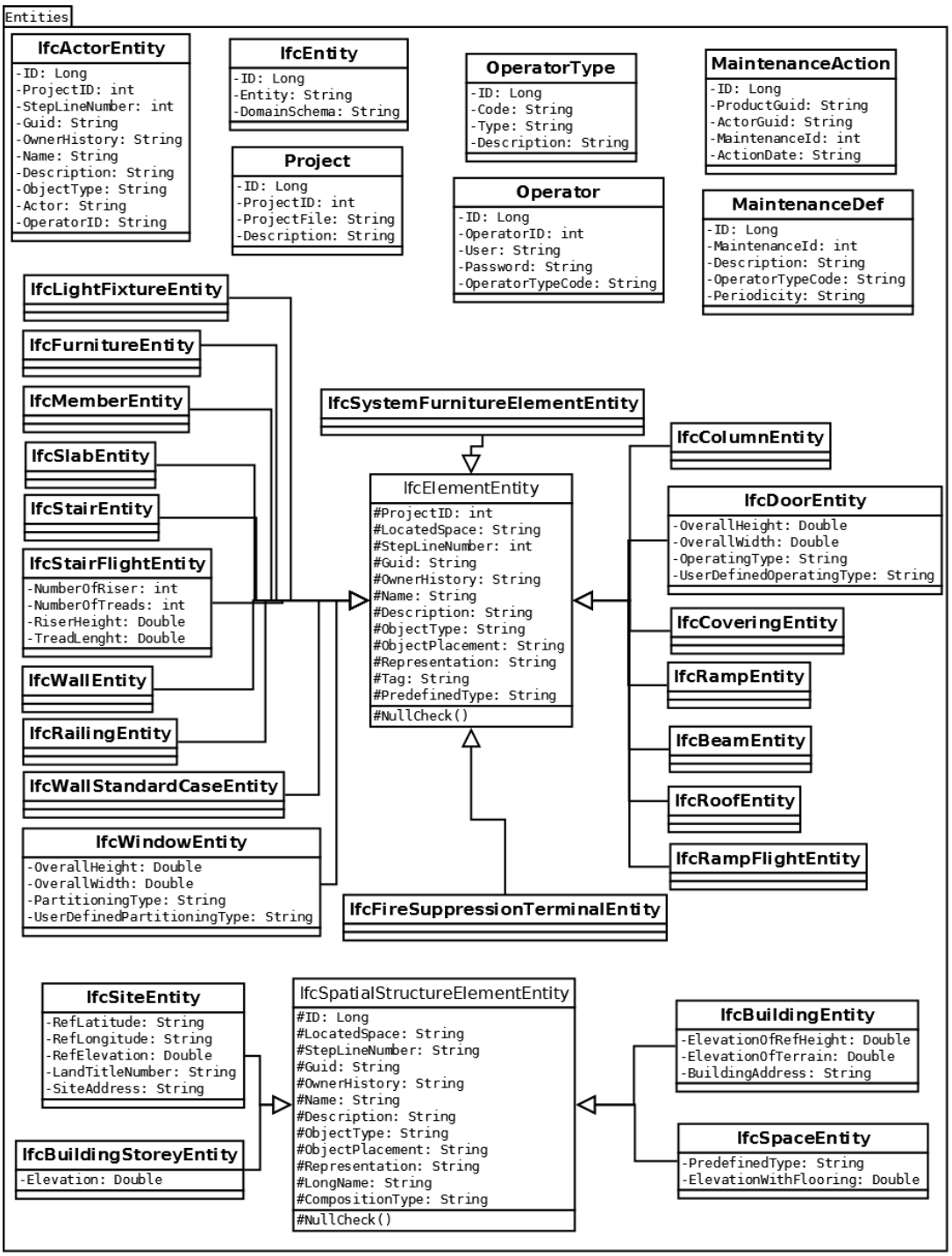

Figura 4.5. Diagrama de clases del paquete Entities.

4.4 Interfaz de la herramienta de testeo.

A pesar de su sencillez, se comentará brevemente la interfaz que se ha utilizado para trabajar con la herramienta de testeo de las bases de datos.

La interfaz está formada por un único frame, conteniendo un TabbedPane como elemento principal que permitirá movernos entre las 3 diferentes opciones de las que dispone la herramienta: inicializar las bases de datos, cargar ficheros de modelos y realizar los test masivos.

4.4.1 Interfaz de inicialización

Como un caso aislado, se ha decidido que los datos referentes a los tipos de operario y la información de los mantenimientos se carguen a partir de ficheros .txt, con un formato CSV concreto. Esta idea aparece debido a que esta parte del modelo no es fija, sino que depende del lugar y momento en el que se inicialice la aplicación. De esta forma, ejecutándolo aquí podemos cargar la información correspondiente a los mantenimientos en el entorno regional y en cualquier otra parte las suyas correspondientes.

Además, este tipo de información es demasiado extensa como para cargarla directamente en el código de la aplicación.

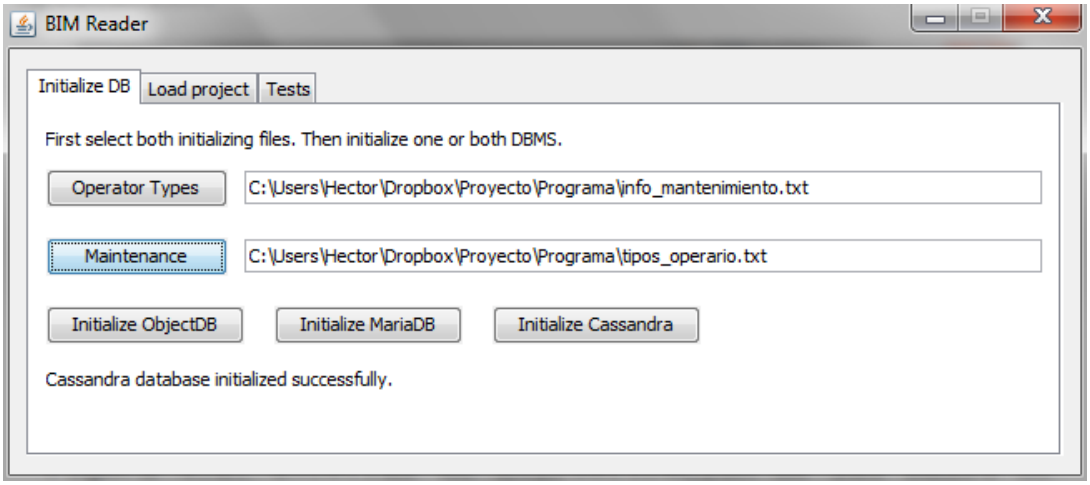

Figura 4.6. Interfaz de inicialización de las bases de datos.

En esta primera pestaña seleccionamos los ficheros de inicialización y seleccionamos la base de datos que queramos inicializar. Con esto se inicializan los tipos

de operario, las definiciones de los mantenimientos y las entidades. Estas últimas también se cargan desde un fichero, pero en este caso se encuentra dentro del paquete de la aplicación por ser un elemento común independiente de la situación. En caso de querer ampliar las entidades van a recibir mantenimientos, habrá que modificar el código y dicho fichero de entidades.

4.4.2 Interfaz de carga de proyectos

La segunda interfaz es la representación de la clase *Reader* que vimos anteriormente. Se encargar de buscar el fichero con el modelo IFC que debe cargarse, pasarlo a la clase mencionada y recuperar los resultados que esta le devuelve.

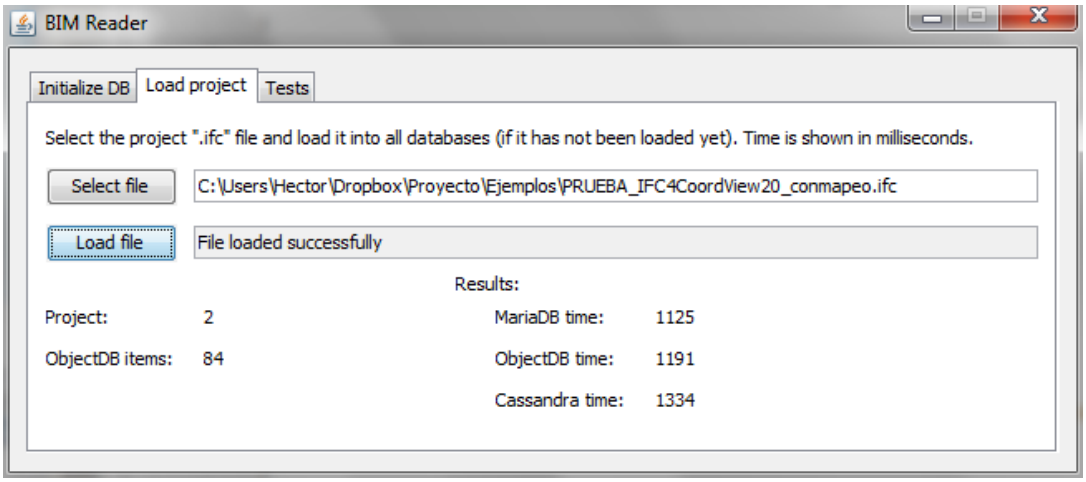

Figura 4.7. Interfaz de carga de proyectos en las bases de datos.

Cuando ordenamos la carga del fichero, se realiza la carga en ObjectDB en primer lugar, seguidamente en MariaDB y, por último, en Cassandra.

4.4.3 Interfaz de testeo

Esta última interfaz es la encargada de pasarle a la clase *Tester* la prueba que debe ejecutar junto con el número de veces que tiene que realizarla.

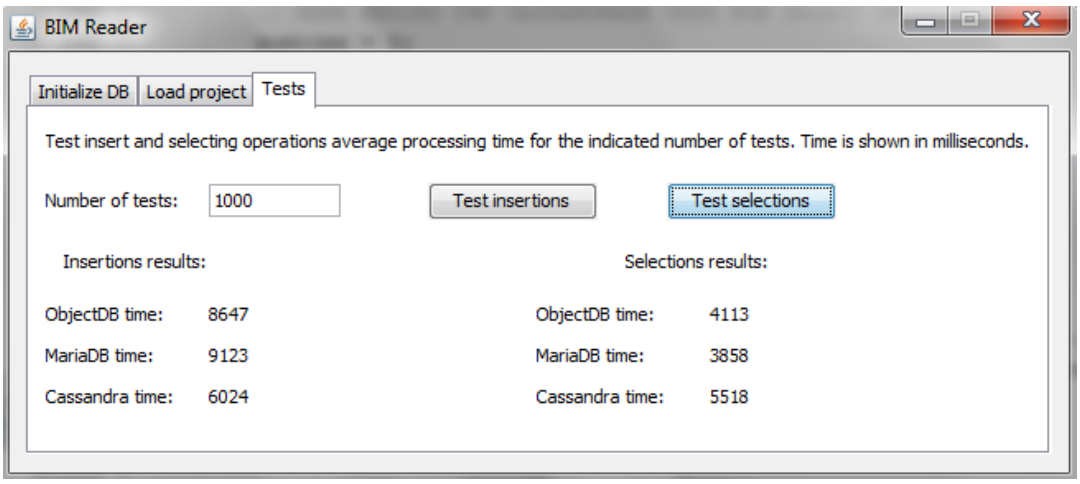

Figura 4.8. Interfaz de testeo de las bases de datos.

Estas pruebas se ejecutan individualmente, forzando a que se finalice la transacción de la inserción/recuperación y abriendo una nueva para el siguiente intento. Con esto evitamos datos irreales por transacciones optimizadas. En el caso real las consultas siempre se realizarán de 1 en 1, por transacciones de consultas individuales.

4.5 Diagramas de flujo de datos de la herramienta de testeo

4.5.1 Diagrama de contexto

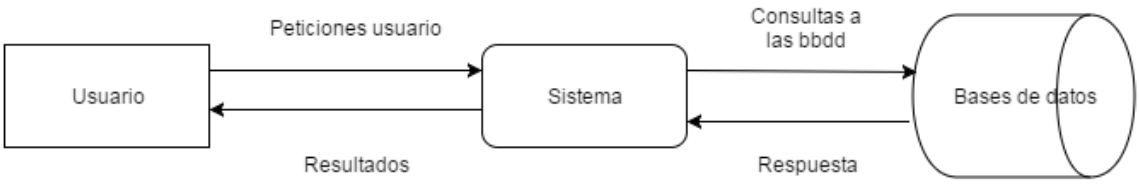

Figura 4.9. Diagrama de flujo de datos contextual.

4.5.2 DFD de nivel 1

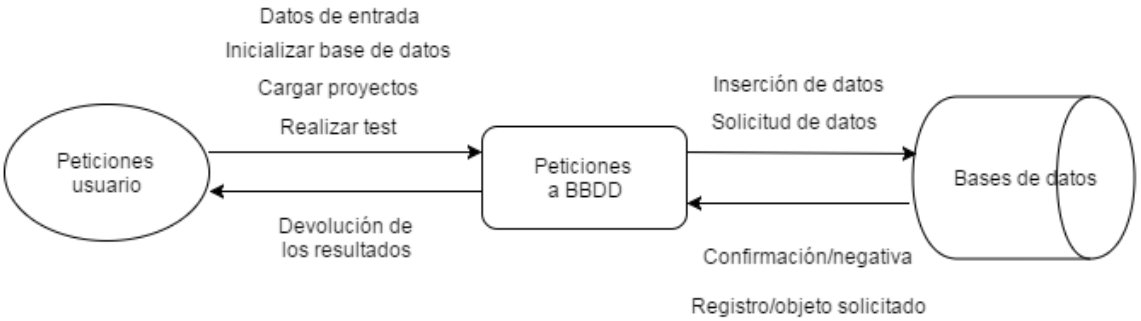

Figura 4.10. Diagrama de flujo de datos de nivel 1.

4.5.3 DFD de nivel 2: Peticiones usuario

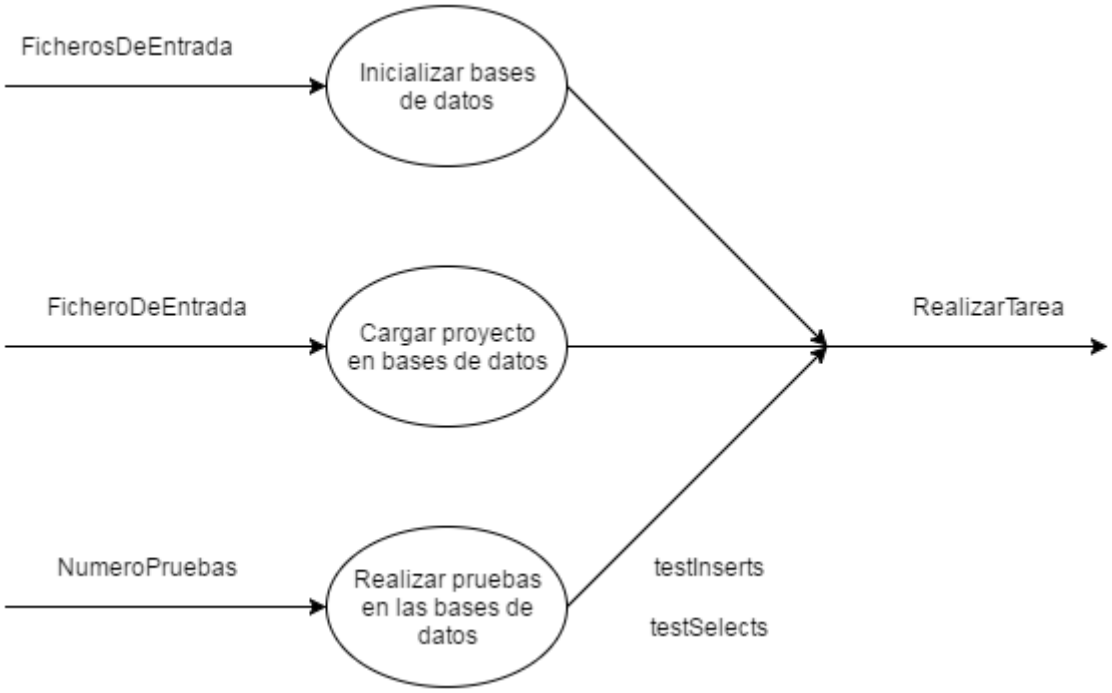

Figura 4.11. Diagrama de flujo de datos de nivel 2: Peticiones usuario.

4.5.4 DFD de nivel 2: Peticiones a BBDD

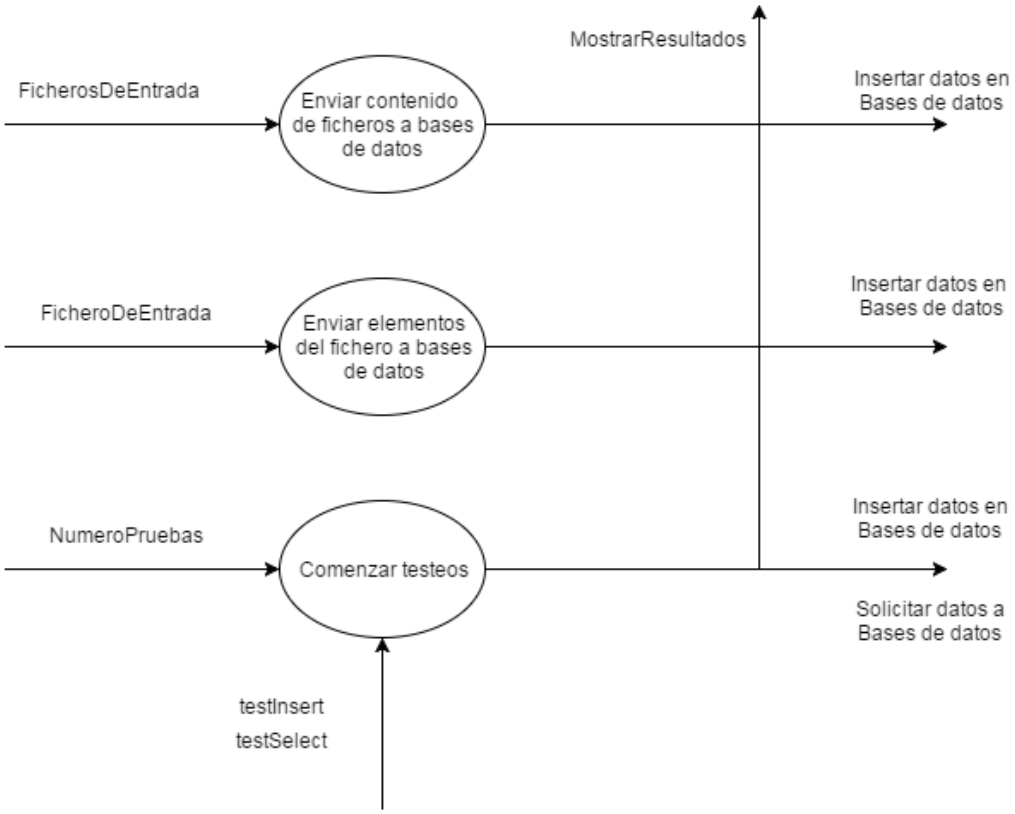

Figura 4.12. Diagrama de flujo de datos de nivel 2: Peticiones a BBDD.

4.6 Diagramas de flujo de datos de aplicación cliente

4.6.1 DFD contextual

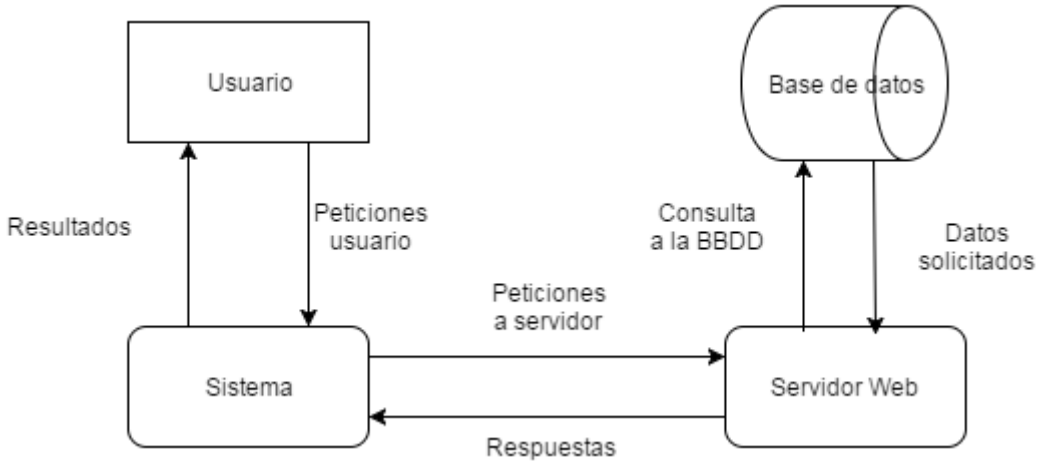

Figura 4.13. DFD contextual de la aplicación.

4.6.2 DFD de nivel 1

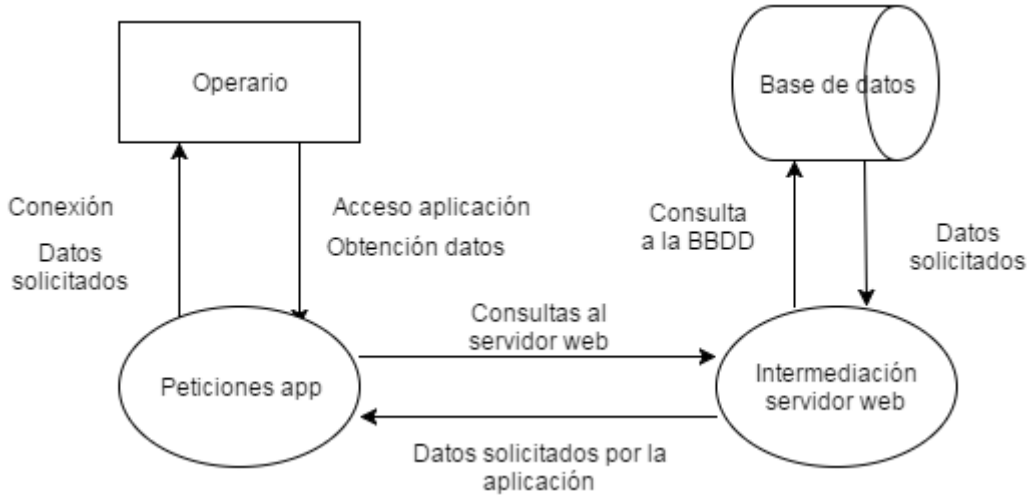

Figura 4.14. DFD de nivel 1 de la aplicación

4.6.3 DFD nivel 2: peticiones de la aplicación

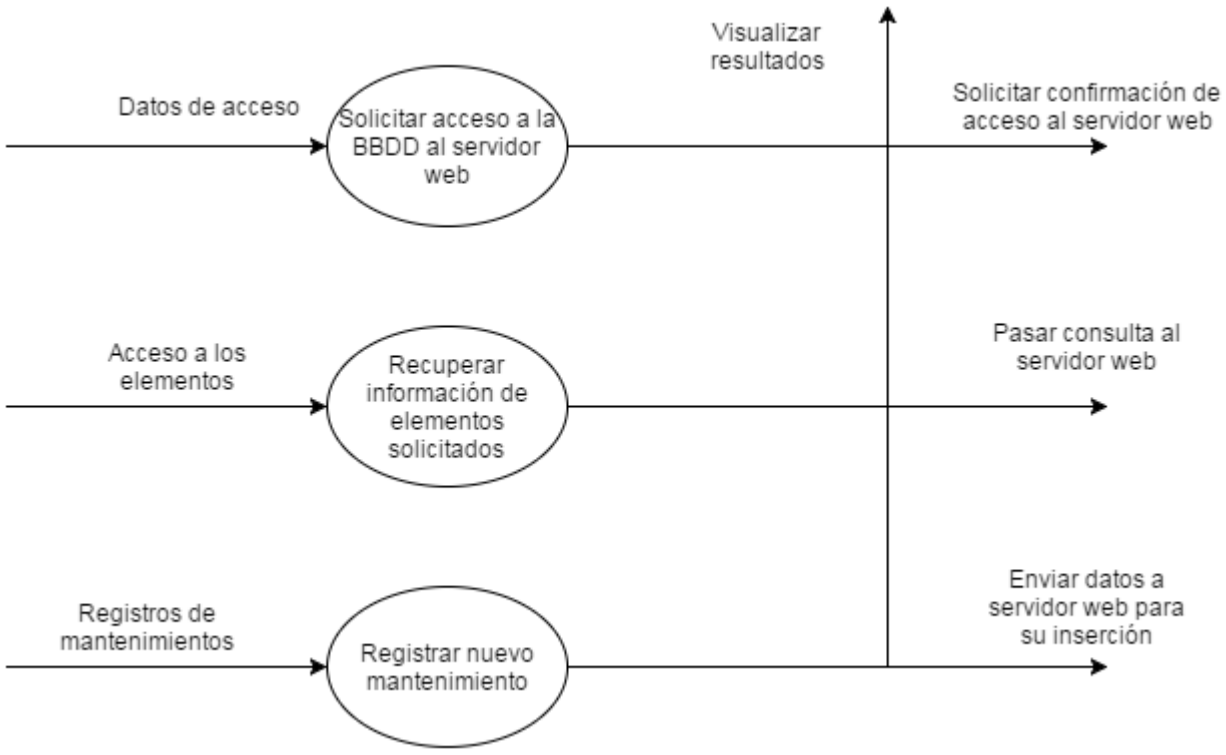

Figura 4.15. DFD de nivel 2: peticiones de la aplicación.

4.6.4 DFD de nivel 2: intermediación del servidor web

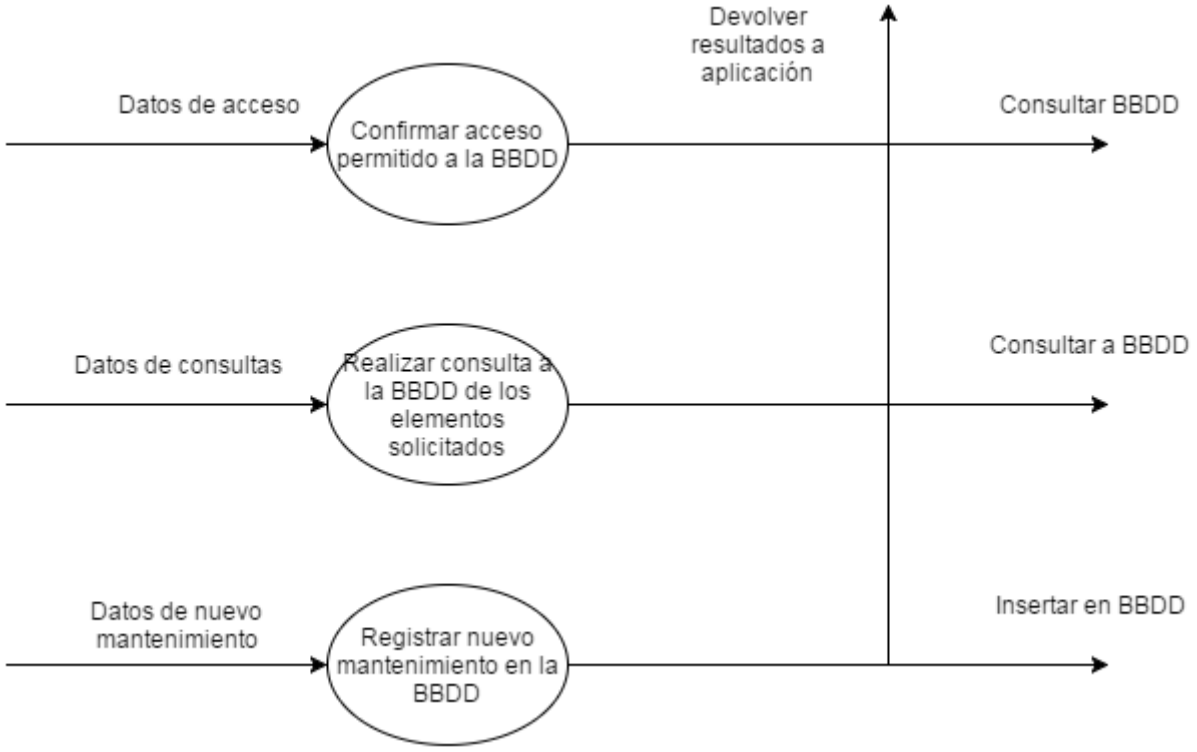

Figura 4.16. DFD de nivel 2: intermediación servidor web.

4.7 Diagrama de casos de uso de aplicación cliente

A continuación se especifica el diagrama de casos de uso de la aplicación. Cada uno de los casos se corresponde con las interfaces que se comentaron anteriormente, por lo que no se volverá a entrar en detalles de cada una.

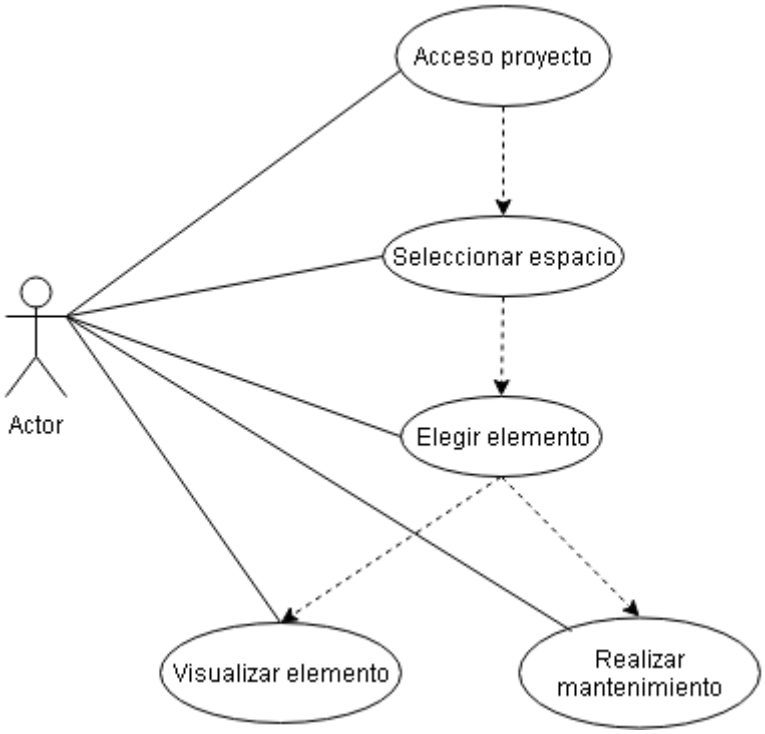

Figura 4.17. Diagrama de casos de uso de la aplicación.

4.8 Interfaz de la aplicación cliente

A continuación se muestran capturas correspondientes a la aplicación Android. Todo el desarrollo se realizó en una tablet de 10.1 pulgadas, dispositivo con el que se corresponden todas las imágenes.

Con esta implementación se pretende reflejar más claramente la forma de trabajar de IFC y del modelo de datos que se ha desarrollado.

En cada una de las interfaces se listan todas las opciones disponibles para ir avanzando nivel por nivel hasta llegar el elemento del cual se quiere realizar el mantenimiento, siempre siguiendo la estructura jerárquica de IFC de la que ya se ha hablado.

4.8.1 Acceso a la aplicación

En esta primera interfaz el operario hará login en la aplicación. En el momento de acceder se mantiene el tipo de operario que es, por lo que las opciones que le aparezcan estarán condicionadas a este valor.

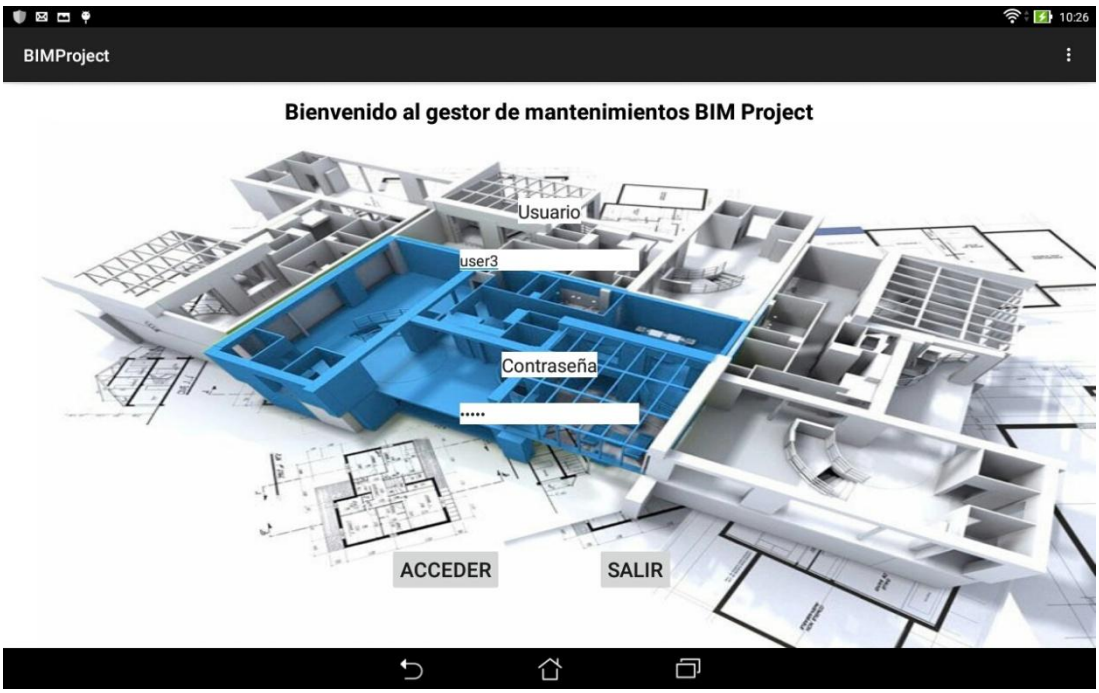

Figura 4.18. Interfaz de login.

4.8.2 Selección de proyecto

En esta segunda interfaz, el operario debe seleccionar el proyecto con el que va a trabajar. Para facilitar la selección, se muestra una vista previa (guardada en la base de datos) de la estructura representada y un resumen de la cantidad de elementos sujetos a mantenimiento que contiene el proyecto.

Como se refleja en el modelo, los operarios están representados en los modelos por los IfcActor, por lo que el operario que haya accedido a la aplicación solo visualizará aquellos proyectos en los que tenga un IfcActor asociado.

Figura 4.19. Interfaz de elección de proyecto.

4.8.3 Selección del habitáculo

Una vez seleccionado el proyecto, el operario deberá seleccionar el habitáculo en el que va a realizar el mantenimiento. Este es el último paso antes de llegar a los elementos, y depende por completo de que en el modelo de diseño original se hayan ubicado correctamente todos los elementos del edificio. De no ser así, no puede haber constancia alguna de dónde está ubicado y, por tanto, de los mantenimientos que se le realizan.

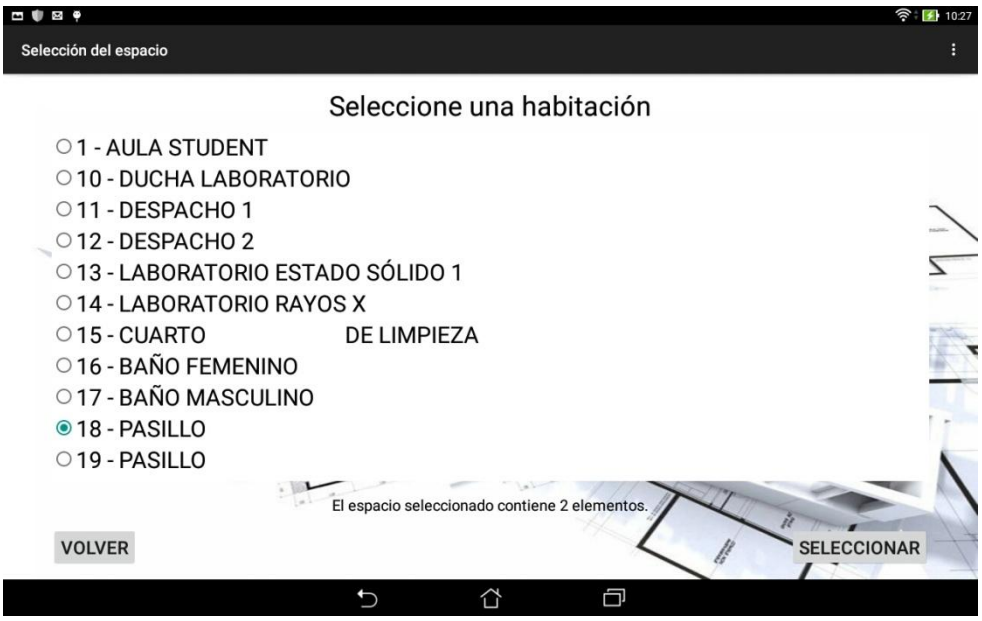

Figura 4.20. Interfaz de selección de espacio.

4.8.4 Selección del elemento

En esta interfaz ya se listan todos los elementos que contiene el espacio seleccionado. Los elementos se listan agrupados según su entidad, para facilitar su localización en caso de que haya múltiples opciones.

Figura 4.21. Interfaz de selección de elemento.

4.8.5 Ficha del elemento seleccionado

Desde la ventana de selección de elemento, podremos consultar la ficha completa de cualquier elemento, incluyendo una imagen del mismo. Esto ayudará a detectar correctamente el elemento que buscamos para hacer el mantenimiento en caso de que haya varios elementos similares en el mismo espacio.

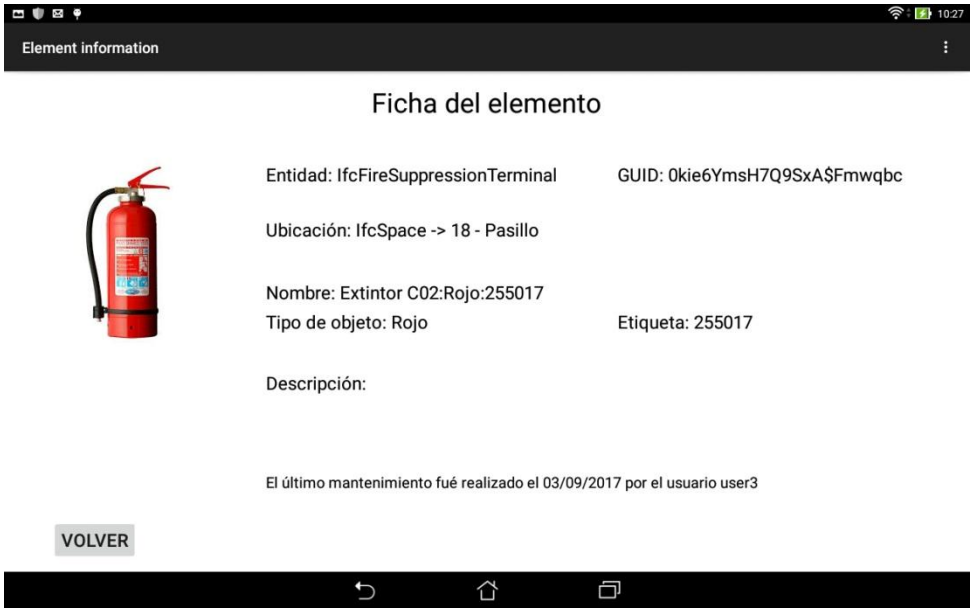

Figura 4.22. Ficha del elemento seleccionado.

4.8.6 Registro del mantenimiento

Con esta interfaz llegamos al final de la cadena. Aquí se registran los mantenimientos de los diferentes elementos. Por supuesto, solo se permiten realizar aquellos mantenimientos orientados al elemento que se ha seleccionado, pero además sólo se muestran como realizables aquellos mantenimientos que puede realizar el operario que ha accedido a la aplicación (recordemos que cada operación de mantenimiento solo puede llevarla a cabo un tipo de operario concreto).

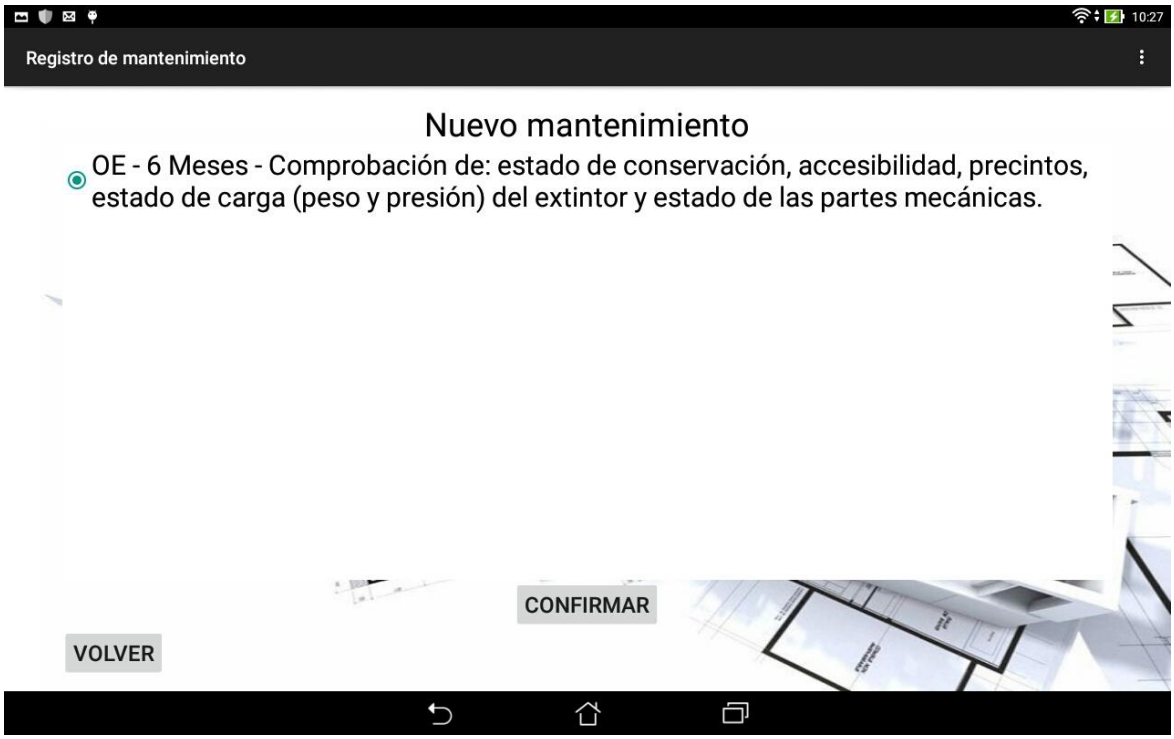

Figura 4.23. Interfaz de registro de mantenimiento.

4.9 Aspectos a comentar

Como aspecto a comentar, la aplicación en Android ha tenido que realizarse con una implementación diferente de la realizada en la herramienta de testeo. Esto se debe a que los drivers JDBC de conexión a base de datos no funcionan en Android, debido a los problemas de seguridad derivados de los mismos. Por tanto, ha habido que recurrir a alternativas de implementación.

En este caso se ha optado por montar un servidor web Apache con soporte para PHP y montar scripts en este lenguaje que funcionen como intermediarios para todas las comunicaciones con la base de datos. Esto se ve reflejado en los diagramas de flujo de datos representados anteriormente.

Capítulo 5. Resultados obtenidos en la investigación

A continuación se exponen los resultados obtenidos en la investigación realizada. El análisis se ha realizado tomando en cuenta varios parámetros, no todos ellos relacionados con los números obtenidos por parte de la aplicación de testeo. Se abordará cada uno de los puntos de vista de la forma más detallada posible según las observaciones realizadas durante el desarrollo del proyecto, de forma que al finalizar este apartado se pueda concluir una respuesta clara de cuál es el modelo de datos que se debe utilizar para el tipo de sistema que buscamos.

Pero antes de conocer los resultados, debe aclararse el entorno de pruebas que se ha preparado para la realización del estudio.

5.1 Entorno de las pruebas

5.1.1 Propuesta inicial

Para el desarrollo de las pruebas se han utilizado 2 máquinas. La primera de ellas es la máquina que actúa como servidor. Se trata de un sistema Windows 7 64 bits con 3 GB de memoria, conectado por Wifi a la red local. La velocidad media de descarga de esta máquina es de 70 Mb/s.

La segunda máquina, que actuará como cliente, se corresponde con un Windows XP de 3 GB de memoria, conectado por cable a la red local a velocidad de 1 Gbps.

Como se puede comprobar ambas máquinas son de rendimiento medio-bajo. Además, durante todos los test se ha forzado el funcionamiento de los equipos a un nivel medio de su capacidad, buscando la simulación de un entorno de trabajo lo más real posible tanto para el servidor como para el cliente. Ninguno de los equipos

dispone de discos de estado sólido ni dispositivos de alta disponibilidad.

Se ha descartado del estudio la realización de pruebas consistentes desde equipos en redes remotas debido a que los pocos intentos realizados daban resultados demasiado dispares e irreales como para tenerlos en cuenta, debido a la dependencia del estado de la red en el momento justo de las consultas. Con esto buscamos eliminar el mayor número de variables posibles que puedan poner en duda la fiabilidad de los resultados.

Además, todas las mediciones se han realizado bajo los mismos datos. En las cargas de ficheros IFC se ha utilizado el mismo fichero para testear todas las bases de datos, y las pruebas de inserciones y recuperación se han realizado sobre los mismos objetos.

Como última aclaración, indicar que el número de pruebas realizadas supera en gran número las reflejadas en el documento. Para cada uno de los distintos tipos de mediciones se lanzó la prueba varias veces, observando y analizando la desviación obtenida en los resultados. Después de varios intentos se toma captura, para reflejo en el presente documento, de una de las ejecuciones que represente fielmente la media de los valores obtenidos en todos los intentos.

5.1.2 Situación final. Problemas y soluciones adoptadas

Tras tomar la decisión de agregar la base de datos Cassandra a las pruebas, se intenta poner en funcionamiento el sistema en la máquina que actúa como servidor (Windows 7). En los entornos Windows, la base de datos Cassandra se utiliza como un paquete comprimido en el que vienen todas las herramientas que necesita el sistema. Dentro de este paquete vienen incluidos los ficheros batch, powershell y bash que nos permiten poner en marcha el sistema o alguna de sus herramientas según el entorno en el que estemos trabajando.

Como en nuestro caso el entorno es Windows, se utilizan la herramienta descargada en paquete. Al no ser instalada, la herramienta no reconoce la versión de Java que tenemos instalada ni fija una ruta de ejecución para la base de datos. Por tanto, tenemos que crear manualmente las variables del sistema que necesita Cassandra y agregar su definición en los ficheros .bat de

la aplicación. Una vez generado, se ejecutan los ficheros correspondientes. Los ejecutables llaman a otro fichero powershell que se encarga de comprobar que el entorno sea válido y empieza a dar errores con la versión de Java. Si echamos un vistazo a dicho fichero encontramos lo que se refleja en la Figura 5.1.

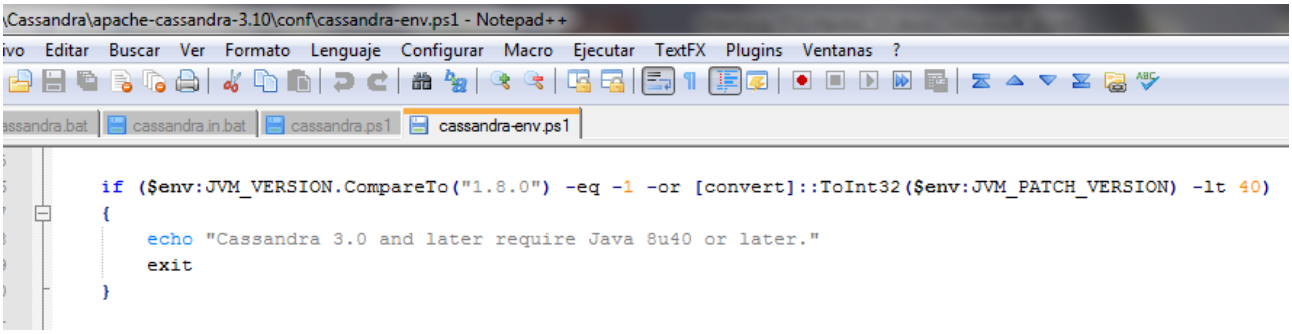

Figura 5.1. Fichero de configuración de Cassandra.

Como se puede ver en la Figura 5.1, el programa compara si la versión de Java es exactamente la 8.40 (versión 1.8 + parche 40) para dar un mensaje de que se requiere la versión 8.40 o posterior y detener la ejecución. Al ejecutar, se obtiene el mensaje de error a pesar de que se tiene instalada la versión 8.112 de Java (dado que se compara exactamente con la 8.40).

A pesar de que el script tiene algunas sentencias de este tipo, si se utiliza la versión 8.40, se fuerza la comparación a la versión que queremos o directamente se quita la comparación del fichero de configuración, al volver a ejecutar el programa comienza a cargar la configuración de la base de datos y termina indicando que la herramienta está operativa, pero al intentar establecer una conexión, se rechaza.

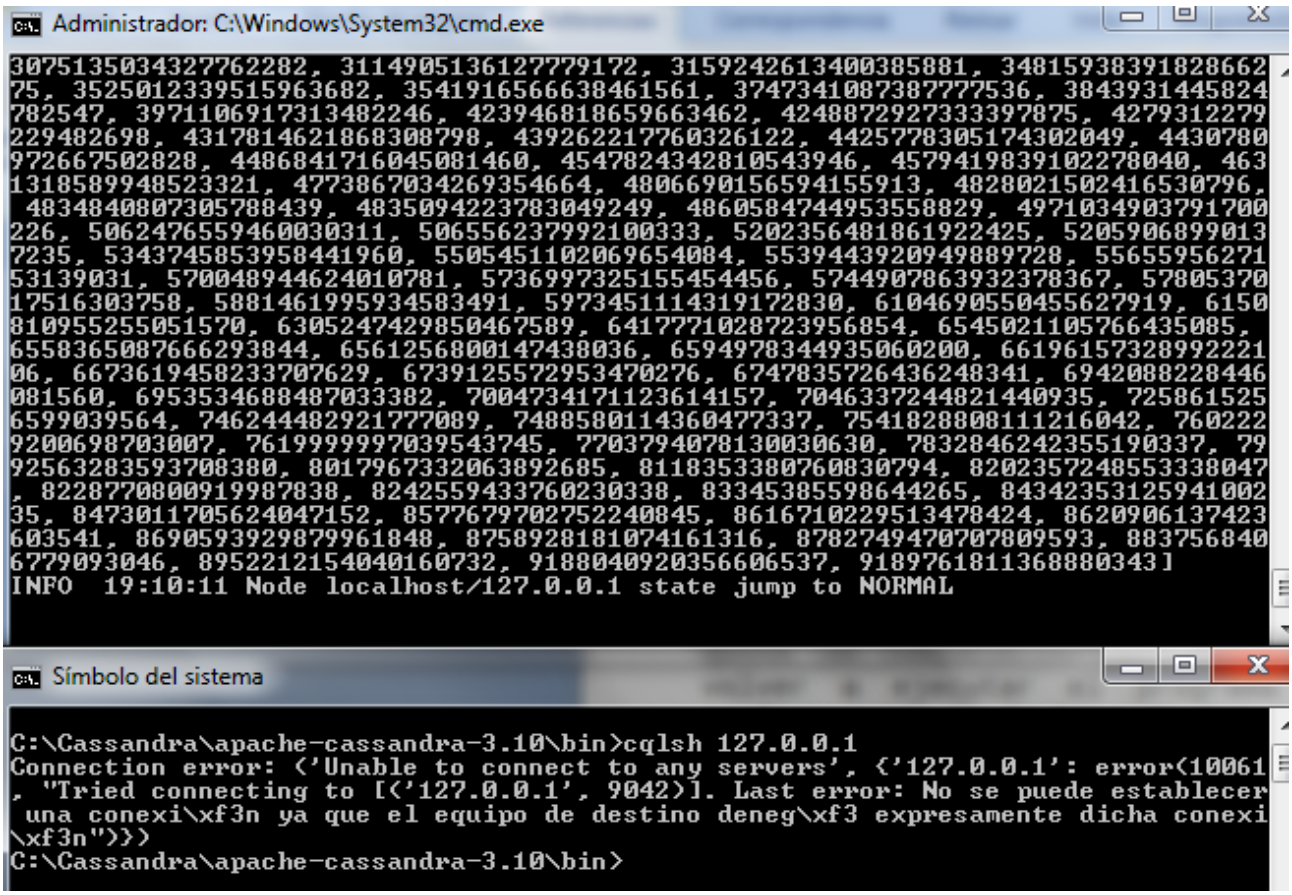

Figura 5.2. Prueba de conexión a Cassandra en Windows.

Los sucesivos intentos con múltiples versiones tanto de Java como de Cassandra, dan exactamente los mismos resultados que se pueden ver en la Figura 5.2, desconociendo por completo la causa y verificando en diversos foros que otros usuarios han sufrido de la misma situación.

Finalmente, para resolver este bache, se opta por realizar una instalación Ubuntu en el mismo equipo de sobremesa en el que se hicieron las pruebas iniciales e instalar Cassandra en él. En este caso no sólo es más sencillo de instalar puesto que se encuentra disponible en los repositorios, sino que además funciona perfectamente con sólo indicarle la IP del propio servidor en el archivo de configuración principal.

Por tanto, los resultados finales que se exponen a continuación están recabados con un servidor Ubuntu de 3 GB de RAM conectado en red local a 1 Gbps y el equipo portátil Windows 7 de 3 GB de RAM conectado por Wifi, que finalmente actuará como cliente.

5.2 Resultados obtenidos

Para las mediciones se ha trabajado con 3 proyectos en formato IFC:

- El proyecto 1 es el proyecto más ligero, de apenas 16 elementos para almacenar en la base de datos.
- El proyecto 2 es el proyecto mediano, de una única habitación pero con varios elementos en su interior. Su tamaño físico es mayor, pero se debe a que tiene mayor contenido geométrico. El número de elementos a almacenar es de 84.
- El proyecto 3 es el proyecto más grande. Representa un edificio completo, aunque sin ningún tipo de mobiliario, solo elementos estructurales, por lo que el fichero es más pequeño que el del proyecto 2. Aun así, el número de elementos a almacenar es de 523.

Se abordarán en primer lugar los aspectos medidos en los test, dejando para el final los aspectos más abstractos de medir.

También hay que tener en cuenta que el funcionamiento de ObjectDB implica un tratamiento de la información diferente del de los otros candidatos. La estructura de tablas de MariaDB que refleja el problema es exactamente la misma que la registrada en Cassandra, dado que la información queda representada en tablas pese a su estructura clave-valor. Por tanto, el nº y tipo de las operaciones con la base de datos en las pruebas son exactamente las mismas, siendo ObjectDB el único que varía en este aspecto.

Por último, hay que mencionar que el entorno de estudio se ha intentado asemejar lo más posible a la realidad, pero aun así hay que interpretar los resultados asumiendo que no es exacta, lo que podría afectar a los resultados. Por ejemplo, Cassandra es un sistema gestor pensado para trabajar sobre múltiples clusters con múltiples nodos y con datos masivos en cada nodo, mientras que aquí se está testeando sobre un único cluster y nodo. Del mismo modo, las arquitecturas sobre las que se están realizando las pruebas no son las propias de un proveedor de bases de datos de uso estándar.

5.2.1 Velocidad de respuesta

Se reflejará este parámetro en primer lugar por ser el gran protagonista de los test y, por tanto, el más extenso. Para abordarlo mejor lo dividiremos entre las pruebas realizadas en local y las pruebas en remoto.

I.5.2.1.1 Velocidad en local

La velocidad de respuesta de las bases de datos a nivel local se ha medido desde dos herramientas. En primer lugar se ha medido mediante la herramienta de carga de ficheros, en la cual se mide lo que tarda la aplicación en guardar el proyecto cargado en cada base de datos. Los resultados obtenidos son los reflejados en la Figura 5.3.

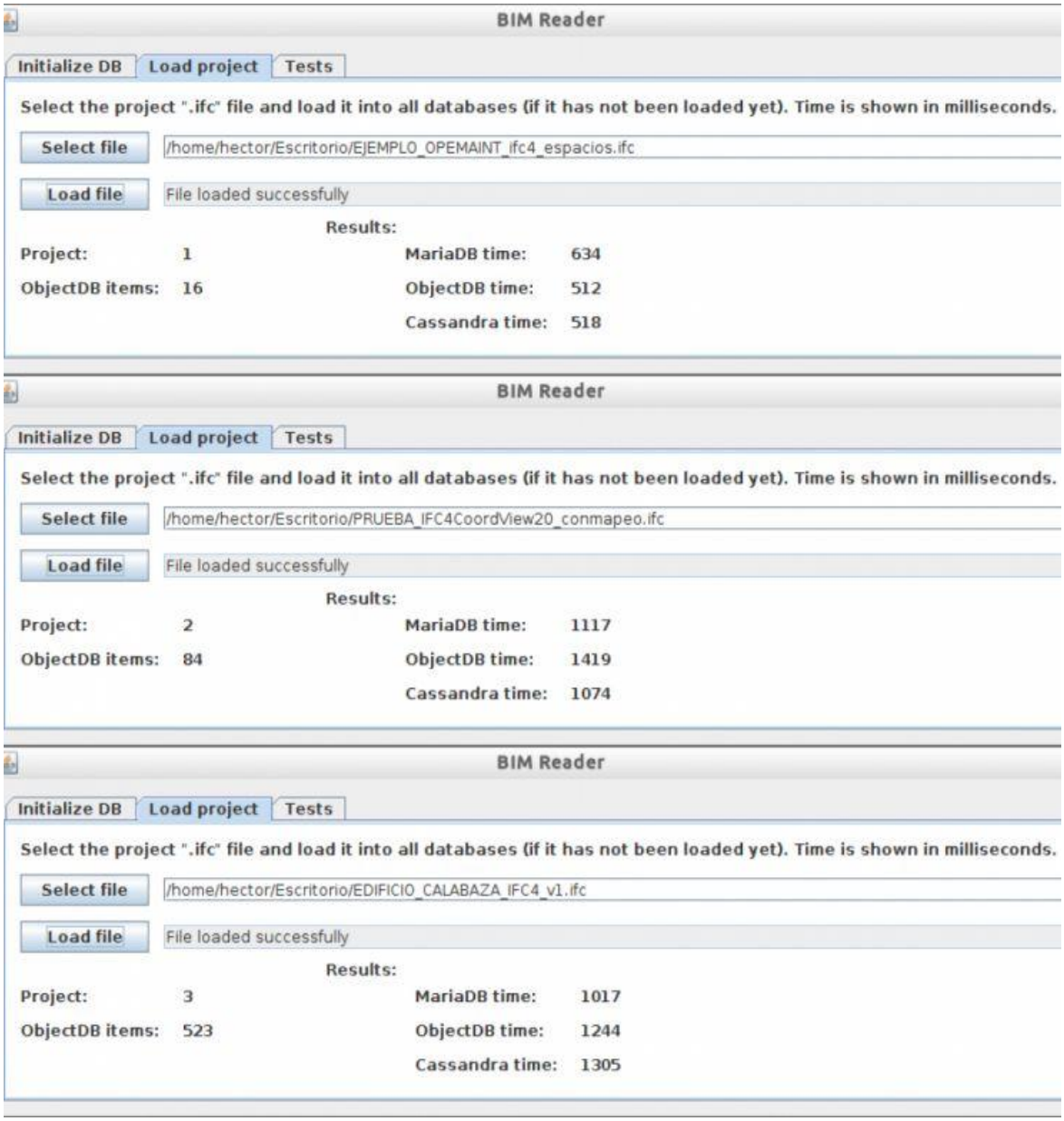

Figura 5.3. Tiempos de carga de proyectos en local.

Como se puede ver, los modelos no relacionales parecen ser los más ligeros al cargar el primer proyecto, con tal solo unos pocos datos que almacenar. Sin embargo, al continuar con los proyectos de mayor envergadura el modelo relacional comienza a predominar, aunque muy ligeramente. En este punto, la diferencia entre las bases de datos es despreciable.

A pesar de ello, se observa que **conforme incrementa el nº de objetos a almacenar el modelo relacional tiende a aumentar el tiempo de trabajo de manera más lineal que los modelos no relacionales**.

También hay que resaltar que, pese a que los resultados favorecen al modelo relacional, **almacenar la estructura ifc en el modelo relacional requiere de más consultas que almacenar objetos**, a pesar de que todos los cambios se realizan en una sola transacción. Hay que recordar que algunos elementos del modelo IFC requieren realizar inserciones en 2 tablas del modelo relacional mientras que en ObjectDB se almacena un único objeto. En Cassandra se produce la misma situación que en el modelo relacional dada la misma estructura de tablas.

La segunda prueba de velocidad realizada consiste en la realización de un número n de inserciones y recuperaciones a las bases de datos. Estas consultas se realizan sobre las acciones de mantenimiento, por ser la información a la que se accederá más frecuentemente desde la aplicación cliente.

Las pruebas se realizaron para 1000, 5000 y 10000 inserciones y selecciones respectivamente, midiendo el tiempo que tardó en realizarse el número completo de consultas, que no la media individual. Estos son los resultados:

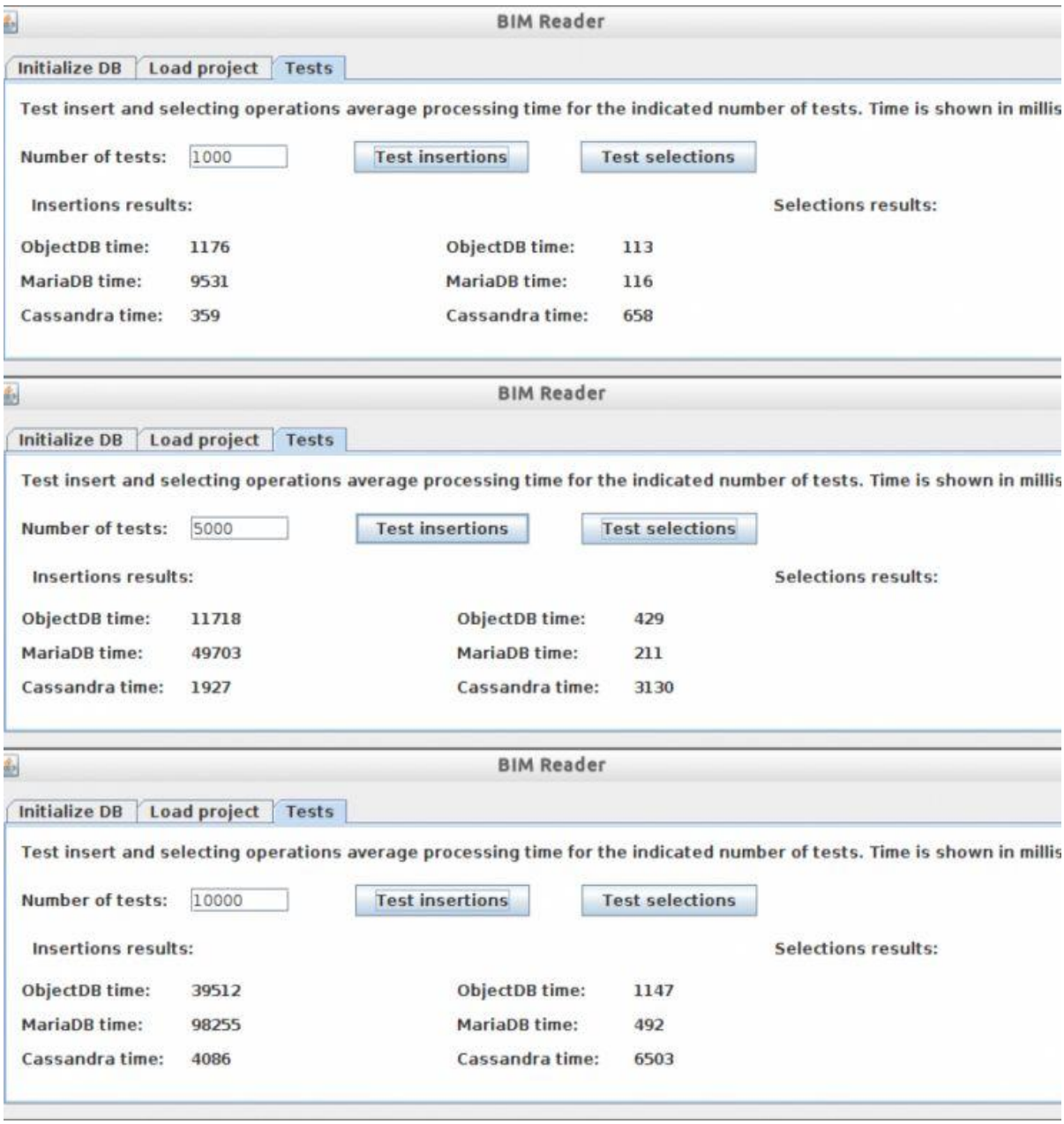

Figura 5.4. Tiempos de respuesta local para un número n de inserciones y recuperaciones.

En este caso podemos ver resultados más destacables. En primer lugar se aprecia que hay que tomar las operaciones de inserción y de selección por separado, puesto que los resultados que muestran son totalmente opuestos.

Comenzando **con las operaciones de inserción**, se puede ver que **el modelo de datos clave-valor es el de mejor rendimiento**, tardando apenas 4 segundos en realizar 10000 inserciones mientras que el modelo orientado a objetos multiplica por 10 ese tiempo y el relacional por 25.

Aquí hay que matizar las situaciones. El sistema gestor de bases de datos relacional conlleva una serie de operaciones intrínsecas en el momento de realizar las

inserciones en la base de datos que se encargan comprobar el cumplimiento de todas las condiciones que garantizan la integridad de una base de datos relacional. La tabla sobre la que se hacen las inserciones tiene definidas 4 claves foráneas, que el sistema relacional comprueba con cada inserción. Si se quisiera garantizar esta misma integridad en los modelos no relacionales debe hacerse desde el software que trabaje con la base de datos, incrementando el nº de operaciones a realizar para cada inserción en gran medida y con ello el tiempo de trabajo. Más concretamente, en el caso de Cassandra no se está reflejando el lastre que tiene esta base de datos de la replicación de la información entre todos los nodos del cluster que deben almacenarla.

Aun así, podemos concluir que **a la hora de almacenar información los modelos no relacionales tienen mejor rendimiento que el modelo relacional, sacrificando la integridad de la información a cambio**.

Por otro lado, observando los resultados para cada uno de los valores de n, podemos ver que los tiempos no aumentan de igual manera en todas las bases de datos. Si multiplicamos por 10 el número de pruebas (de 1000 a 10000), vemos que el tiempo de la base de datos relacional también se ve incrementado en la misma proporción; mientras que el modelo orientado a objetos aumentó su tiempo en casi 40 veces más. La base de datos clave-valor también mantiene un escalado constante.

Por tanto, asumiendo que el modelo relacional aporta los resultados más fiables se puede asegurar que **si aumentamos drásticamente el número de elementos contenidos en la base de datos el rendimiento del modelo de objetos baja de forma igualmente drástica, mientras que el aumento de tiempo en las bases de datos relacional y clave-valor se mantiene proporcional al número de inserciones que se realizan**.

De esta forma, **las inserciones se vuelven mucho más eficientes en modelos relacionales sobrecargados que en modelos orientados a objetos o clave-valor sobrecargados**.

Pasando a analizar las operaciones de recuperación, se observa que **el modelo relacional es bastante más rápido en las consultas que los otros candidatos, con menos de 0,5 segundos de respuesta para 10000 pruebas, sin verse prácticamente afectado por el número de pruebas**. Le sigue el modelo orientado a objetos, cuyo tiempo es bueno y se

mantiene proporcional al número de pruebas a realizar. Por último, el modelo clave-valor es el más lento debido a las operaciones de cluster que debe realizar antes de devolver la información, aunque su tiempo de trabajo también es proporcional al número de pruebas.

Como dato adicional de esta prueba, hay que aportar que en un momento determinado de las pruebas se testeó realizar las consultas buscando en la tabla de mantenimientos sobre un elemento no indexado. Los resultados de esta prueba marcaron que los tiempos generales de respuesta se multiplicaban por 10 en el caso de ObjectDB y hasta por 30 en el caso de MariaDB. Con esto se demuestra que **el modelo relacional es mucho menos tolerante a errores de diseño en la base de datos que el modelo orientado a objetos**. Sobre Cassandra no se pudo realizar esta prueba, puesto que por defecto rechaza todas las operaciones de consulta que se hagan filtrando sobre campos no indexados, como medida de precaución para evitar grandes tiempos de ejecución cuando la base de datos está sobrecargada.

I.5.2.1.2 Velocidad en red local

En este caso se han realizado las mismas pruebas que en local pero esta vez desde el ordenador cliente. En primer lugar se han realizado las inserciones de los proyectos en cada base de datos.

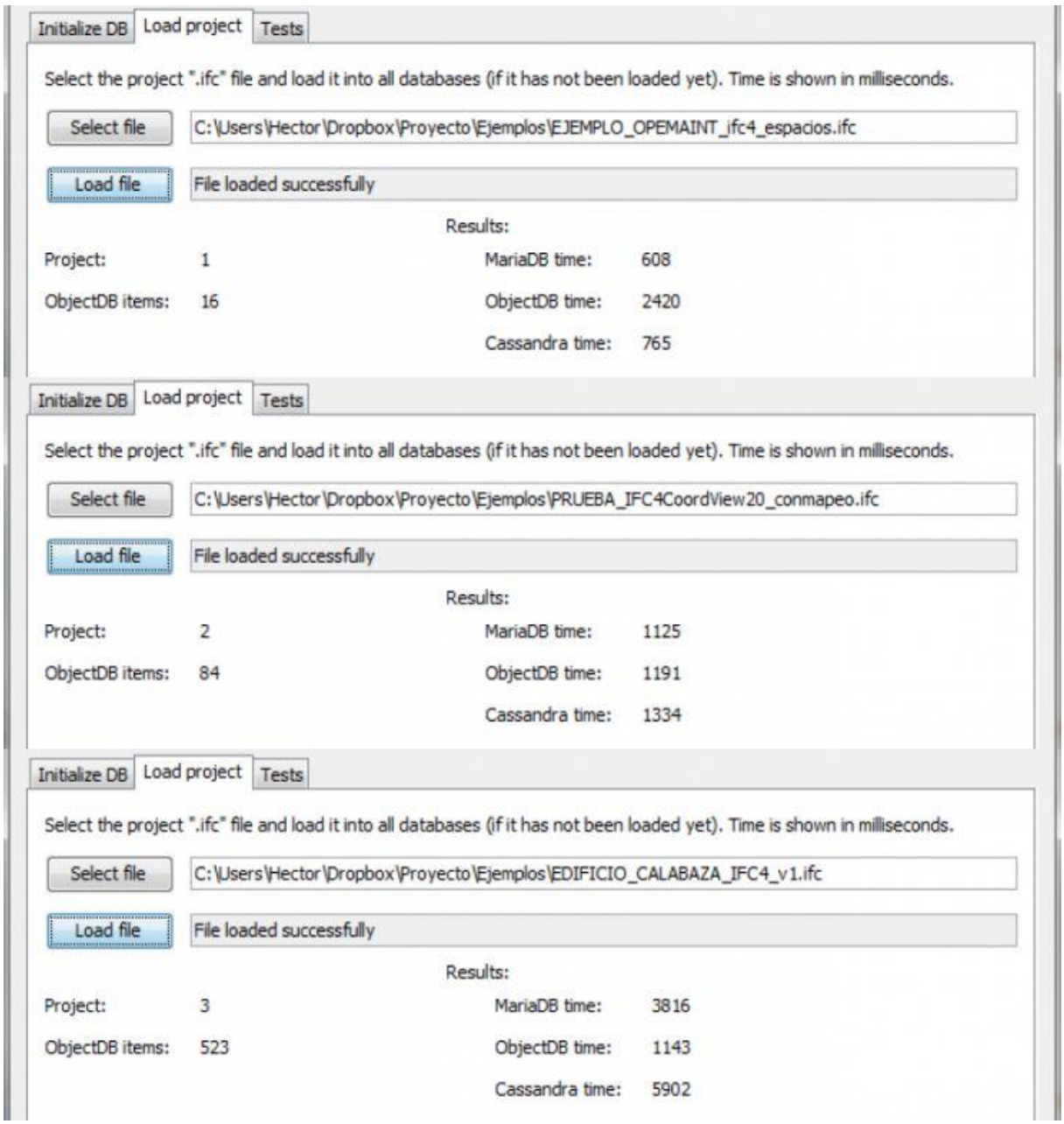

Figura 5.5. Tiempos de carga de proyectos en red.

Como se puede ver, los resultados varían con respecto a los resultados de las pruebas en local. En este caso **el modelo que da mejor respuesta es el orientado a objetos (aunque con fluctuaciones importantes en los valores)**. **Le sigue el modelo relacional**, con un incremento lineal según la cantidad de información a evaluar. **El modelo clave-valor** también crece de manera lineal, pero **sigue siendo el de peor rendimiento en las inserciones**, al igual que en las pruebas locales.

Le siguen las pruebas de inserciones y recuperaciones masivas.

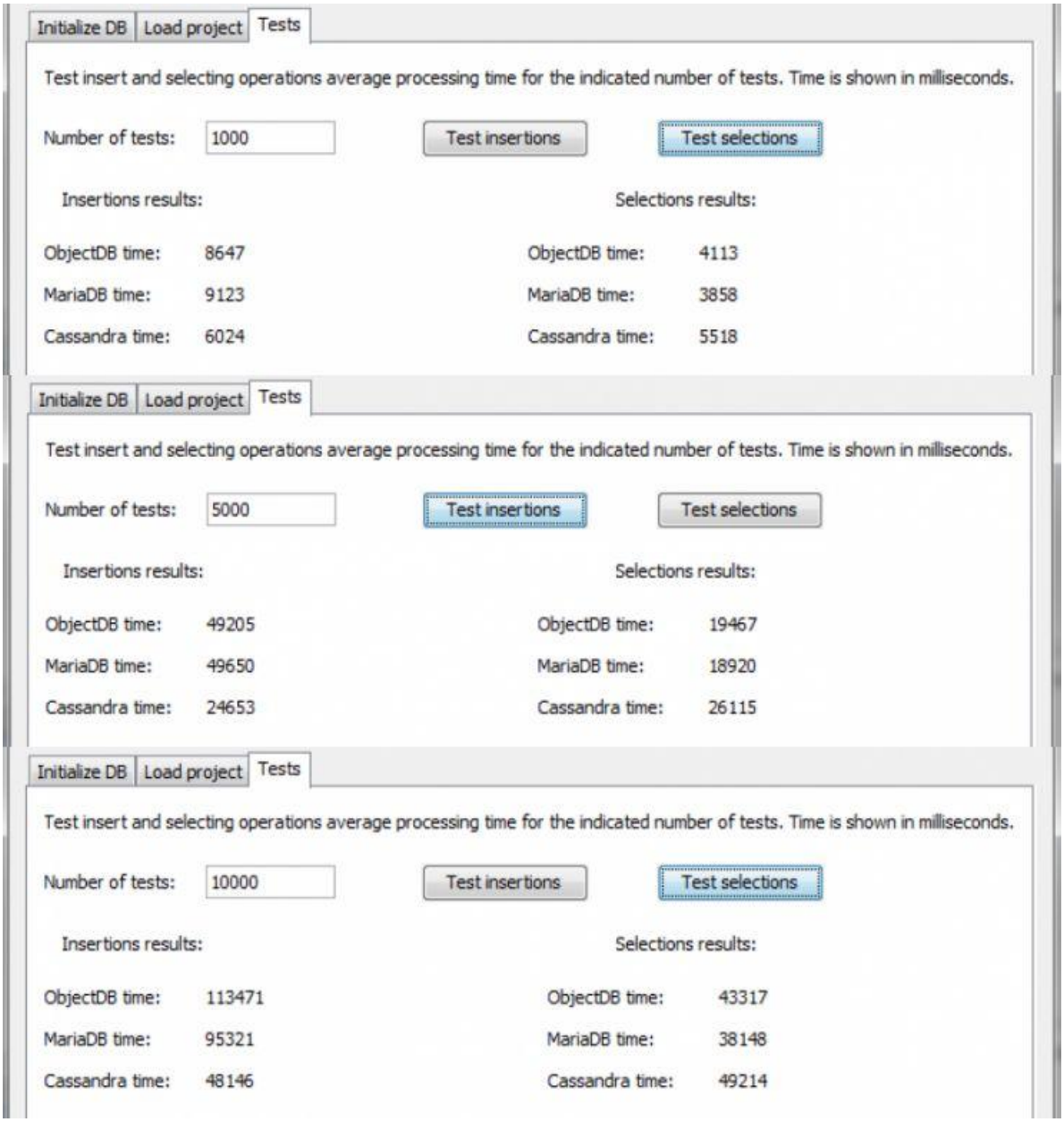

Figura 5.6. Tiempos de respuesta en red para un número n de inserciones y recuperaciones.

En esta última prueba se producen algunas variaciones con respecto a las realizadas en local. **Empezamos evaluando las inserciones. El modelo clave-valor se mantiene en cabeza en las tareas de inserción,** sin verse afectado por el nº de pruebas realizadas; **el modelo relacional** se corresponde con los valores presentados en las pruebas locales **con los peores resultados pero el mejor escalado lineal ante el aumento de pruebas**. Por último, **el modelo orientado a objetos**, en este caso, **comienza rindiendo mejor que el relacional pero finalmente se muestra con el peor rendimiento de todos,** con un mal escalado con respecto al nº de pruebas realizadas.

En cuanto a los tiempos de **recuperación de información,** los resultados se corresponden exactamente con los mostrados en las pruebas locales. Aunque en este caso no es tan singular, **el modelo relacional sigue ofreciendo los mejores resultados en las operaciones de recuperación y el modelo clave-valor los peores. El modelo orientado a objetos se mantiene entre ambos**, nuevamente con mayores problemas a medida que se incrementa el número de operaciones.

Además de estas pruebas en red, se probó a reducir el funcionamiento de los equipos al mínimo para comprobar si la carga de los equipos afectaba significativamente a la respuesta de las bases de datos, pero no fue así. Los resultados mejoraban entre un 4% y un 6% en todos los casos, lo cual no resulta un impacto significativo teniendo en cuenta el número de pruebas que se realizaron.

Como anécdota final a este apartado, cabe comentar que el rango de valores entre los que se movía MariaDB en todas las pruebas era mayor que el de ObjectDB y Cassandra, es decir, **la conexión con los servidores ObjectDB y Cassandra dan la impresión de ser más estables que la conexión con el servidor MariaDB**. Aun así, los casos en los que se dejó ver este hecho no fueron numerosos ni determinantes para el estudio.

5.2.2 Entidades de la base de datos.

Observando ambos modelos diseñados, se puede ver que el modelo relacional y el de clave-valor resultan mucho más simplificados que el modelo orientado a objetos. En la parte ajena a IFC (proyecto, definiciones de mantenimientos y operarios…) ambos modelos requieren de los mismos elementos, mientras que a la hora de almacenar los elementos IFC hay una gran diferencia. En ObjectDB, cada uno de los elementos tiene su propia entidad, tal como lo marca el esquema IFC, por lo que dar solución al problema ha implicado definir mayor número de entidades que tablas en el caso del modelo relacional y el de clave-valor. En estos últimos, todos los elementos se encuentran en la misma tabla diferenciados por un campo, ya que el almacenamiento en tablas independientes no es viable en este caso.

Esto afecta también a la escalabilidad del sistema. En ObjectDB se ha de definir cada nueva entidad en la aplicación que conecte con la base de datos antes de poder dar persistencia a ese objeto. En MariaDB o Cassandra sólo hay que comprobar si esa nueva entidad que se quiere representar dispone de datos únicos. En caso de no ser así, no es necesario realizar ninguna modificación en la base de datos, únicamente dar entrada al nuevo registro en la tabla de elementos con la información correcta.

En este aspecto tenemos por un lado al **modelo orientado a objetos que respeta el formato de entidades independientes del estándar IFC pero su definición es más costosa**; mientras que **en el modelo relacional o en el de clave-valor no es viable respetar dicho estándar pero resulta en que la definición de entidades es mucho más sencilla y no implica realizar modificaciones en la estructura de la base de datos para las nuevas incorporaciones** (salvo en algunos pocos casos) además de agilizar las consultas sobre estos datos.

Del mismo modo, al realizar las inserciones, en el modelo orientado a objetos cada entidad a almacenar es un objeto nuevo en la base de datos, mientras que en el modelo relacional se trata de una única entrada en una tabla en la mayoría de los casos.

5.2.3 Consumo de recursos del sistema.

En este caso no se ha podido realizar una comparación de consumo de recursos eficiente. Hay que tener en consideración que el uso de ObjectDB conlleva al uso de Java como lenguaje. El consumo de recursos de los procesos que hemos utilizado dependen de la asignación de recursos del sistema a la JVM, y la cantidad de recursos asignados a la aplicación dentro de la máquina depende de la gestión que realiza la propia máquina en el momento de la ejecución. Según el monitor del sistema, durante la aplicación la memoria destinada a Java se mantiene constante durante toda la ejecución, y sus valores no son muy altos (entorno a los 300 MB).

Aun así, es muy posible que una aplicación que se comunique con MariaDB o Cassandra desde un lenguaje que no sea Java consuma menos recursos que cualquier trabajo equivalente con ObjectDB.

En cuanto a las bases de datos, hay que destacar que **ObjectDB no dispone de instalación en el sistema, sino que se ejecuta bajo aplicación portable**. Esto implica que mientras MariaDB o Cassandra actúan como servicio del sistema, con todo lo que eso significa frente al sistema operativo, **ObjectDB se mantiene en memoria como un programa de escritorio en segundo plano pero en ejecución continua, por lo que podemos afirmar que su consumo de memoria será siempre superior al de MariaDB y Cassandra**, además de las desventajas de no actuar como un servicio del sistema operativo.

5.2.4 Espacio en disco

A pesar de que hoy en día este aspecto cada vez es menos relevante, con él podemos ver reflejada la diferencia entre almacenar un sistema con relaciones y dependencias, frente a un sistema de almacenamiento sencillo y directo (siempre que se comparen modelos que den respuesta al mismo problema).

En el momento de crear la base de datos, en el caso de ObjectDB esta no ocupa espacio en memoria, ya que ni siquiera existe. La base de datos se creará en el momento que se establezca la primera conexión con la base de datos y ocupará un espacio en disco mínimo (unos 256 KB). En MariaDB la propia creación de las tablas, relaciones, índices, triggers y demás elementos que formen parte de la base de datos ocupan espacio de disco, en mayor o menor cantidad dependiendo de la complejidad del modelo.

Tras cargar los proyectos en la base de datos se aprecia que el aumento de tamaño en disco no es significativo en ninguno de los dos modelos: ambos se mantienen inferiores a 1 MB de capacidad. Sin embargo, en el momento de realizar los test, tras agregar 10000 objetos a ambos modelos, el tamaño de MariaDB ha subido a los 9,8 MB, mientras que en ObjectDB se ha mantenido en menos de 2 MB.

Por norma general, **las bases de datos ObjectDB consumen menos espacio de disco que las bases de datos de MariaDB, aumentando la diferencia a medida que crece el número de elementos almacenados.**

Esta característica se ha ignorado en Cassandra puesto que es un sistema de bases de datos distribuido, cuyo espacio ocupado dependerá de la red de nodos que se cree.

5.2.5 Conectividad

El primer gran hándicap que nos encontramos en términos de conectividad corresponde a la versatilidad de los sistemas. Mientras que **MariaDB y Cassandra son sistemas válidos para comunicar desde prácticamente cualquier tipo de aplicación y ante la mayoría de problemas que se plantee** (aunque en algunos casos funcione de forma menos eficiente), **con ObjectDB estamos totalmente limitados al uso de un único lenguaje de programación factible**. El hecho de que este lenguaje sea el más utilizado en el mundo actualmente suaviza ligeramente el problema, pero no quita que sea una gran barrera a la hora de plantearnos qué modelo de datos utilizar para nuestra aplicación. Además, **la casuística de problemas que resuelve con una eficiencia destacable es mucho más reducida que la del modelo relacional,** aunque equiparable al modelo clave-valor en este aspecto.

Por contrapartida, la curva de aprendizaje de ObjectDB y Cassandra es mucho más rápida que en MariaDB. Es cierto que, quien más o quien menos, ha tenido contacto alguna vez con alguna base de datos relacional y puede manejarse en él cómodamente, pero **los requisitos de aprendizaje necesarios (partiendo de un nivel bajo) para aprender a utilizar correctamente un modelo relacional es más elevado que el requerido por el modelo orientado a objetos o el de clave-valor**.

Para poder crear una base de datos relacional que resulte eficiente hay que tener en cuenta una serie de restricciones y comportamientos del modelo, así como unas nociones básicas de diseño de bases de datos, que son más complejos de asimilar que el hecho de crear un objeto en un lenguaje más o menos conocido y simplemente almacenarlo. Además, tanto en el modelo de objetos como en el de clave-valor se pueden cometer errores de diseño sin que lleguen a afectar al funcionamiento del sistema, mientras que el modelo relacional es mucho menos tolerante a fallos y más exigente frente a los mismos.

Para una persona conocedora del modelo relacional que quiera aventurarse en las bases de datos orientadas a objetos el tiempo de adaptación también es bajo, dada la sencillez de su uso, mientras que en el caso contrario es posible que no sea así.

5.3 Análisis de los resultados

Antes de evaluar los resultados obtenidos, se puede aventurar que los modelos relacional y orientado a objetos son los que mejor encajan con la estructura y naturaleza de los datos que se trabajan en este proyecto. Sin embargo, una vez recopilados los resultados vemos que el modelo clave-valor también ofrece buenos números para el problema presentado. Aun así, los resultados de rendimiento obtenidos no descartan a ningún modelo como inviable, puesto que todos han respondido a las pruebas, en mayor o menor medida. Sí parece que el modelo relacional responde mejor no sólo por los tiempos de rendimiento, sino porque nos garantiza la integridad de unos datos con una fuerte jerarquía de dependencias entre ellos. Un atributo que no ofrecen los otros modelos.

Si bien es cierto que, dados los números que ha ofrecido en las pruebas, el modelo clave-valor en el que se basa Cassandra puede resultar muy adecuado si se da la casuística para el que está pensado: disponibilidad continua, escalabilidad, almacenamiento distribuido… Sería una gran opción ante volúmenes de datos masivos.

En cuanto al modelo orientado a objetos, a pesar de ser perfectamente viable, parece el menos adecuado. No sólo por los peores números frente a sus competidores que ofrece sino también por la limitación derivada del uso de un lenguaje orientado a objetos para poder desarrollarla. Además, mientras que el modelo clave-valor sacrifica la integridad para obtener disponibilidad y rendimiento, en el caso del modelo orientado a objetos tenemos números normales aun sabiendo que tendremos que controlar la integridad mediante el software que lo controle si necesitamos de esa característica.

En la Tabla 5.1 Resumen comparativo de las bases de datos utilizadas. se presenta un resumen de los pros y contras encontrados en el análisis de cada una de las BBDD barajadas.

ObjectDB	- Rápido, con cargas de datos pequeñas, pero el tiempo aumenta exponencialmente al incrementarse el número de objetos, y su rendimiento baja drásticamente en bases de datos sobrecargadas.

	- Rendimiento equilibrado entre las recuperaciones y las inserciones. - Requiere de menos instrucciones para trabajar con los datos. - La exclusividad de su diseño obliga a trabajar con un único lenguaje posible. - Mayor flexibilidad en el diseño de la base de datos. Curva de aprendizaje suave. - Dificulta la integración con otras aplicaciones. - No garantiza la integridad de la información ni su disponibilidad. - No es válido para todos los casos.
MariaDB	- Requiere de más instrucciones para trabajar con la información. - Muy lenta en modelos con pocos elementos, pero muy eficiente en bases de datos sobrecargadas, con incremento lineal de los tiempos respecto al crecimiento de los datos. - Velocidad de inserción elevada frente a una velocidad de recuperación mínima. - No garantiza la disponibilidad de los datos, pero sí su integridad. - Su diseño es más complejo, restrictivo y sensible a errores que el de un modelo no relacional. Requiere de mayores conocimientos que los modelos no relacionales para su correcto desarrollo y uso. - Se adapta a la mayoría de casuísticas, facilitando en gran medida la interoperabilidad con otras herramientas.
Cassandra	- Buen rendimiento en cargas ligeras de datos, sobre todo en inserciones. Velocidad media en las recuperaciones, aumentando en gran medida conforme a la sobrecarga de la base de datos. - Diseño sencillo y adaptable a casi

	cualquier casuística, aunque sólo se obtiene el máximo partido en casos muy específicos. - Se integra bien con otras herramientas. - Redundancia máxima, aunque con escalabilidad horizontal sencilla. - No asegura la integridad, pero sí la disponibilidad de la información. - Su curva de aprendizaje es sencilla.

Tabla 5.1 Resumen comparativo de las bases de datos utilizadas.

Capítulo 6. Conclusiones y líneas futuras

Tras haber entrado en profundidad en la temática BIM e IFC, la impresión que me ha dado es que se encuentra en una fase de desarrollo demasiado prematura como para aventurarse a desarrollar proyectos de gran complejidad basándonos en estos sistemas. Considero tan importante avanzar y desarrollarse como asegurarse del correcto funcionamiento y uso de lo que ya está hecho, y esto último no se está logrando de momento. Poniendo un ejemplo concreto, no sirve de mucho poder almacenar tanto volumen de información si aquellos encargados de almacenarla no lo hacen correctamente o ni siquiera saben cómo hacerlo.

Creo que antes de aventurarse en grandes proyectos de desarrollo deben aventurarse más proyectos de investigación y de formación paralelos al crecimiento de la tecnología que se encarguen de ir puliendo poco a poco las capacidades que estos sistemas pueden aportar a los usuarios. Por supuesto nos encontramos ante el gran hándicap de la evolución que es la resistencia natural de los humanos frente a los cambios. Pero, al fin y al cabo, son los usuarios de estos sistemas los que más ventajas pueden alcanzar de aprender a utilizarlos eficientemente, por lo que debe mantenerse una línea de trabajo constante y disciplinada hacia estos objetivos por parte de los miembros de este sector profesional.

Otro aspecto que llama la atención, es que estamos incorporando una carga técnica que reside, a priori, en áreas de informática a un proyecto de edificaciones. Es posible que de ahora en adelante las alianzas entre los profesionales de ambos sectores sean no sólo más frecuentes sino también necesarias. Esto sin duda resulta interesante por la cantidad de posibilidades que genera.

En cuanto a la parte técnica, la principal conclusión que se puede sacar es que no porque algo se encuentre en auge o de moda, como son las bases de datos NoSQL, ello significa que sea mejor que lo que había antes. En este

caso se ha visto reflejado perfectamente: nos hemos encontrado con un problema que, por la estructura del estándar que lo soporta, parecía un claro candidato a un modelo orientado a objetos. Tras analizar los resultados vemos que no ha sido así, ya que las variables que marcan qué modelo de datos utilizar y cuál no son múltiples, y se deben contemplar todas en su conjunto para poder tomar una decisión correcta.

Como trabajos futuros, parece claro que no hay ningún modelo de base de datos no relacional que concuerde mejor con el problema desarrollado en este proyecto, salvo que se den unas condiciones adicionales que favorezcan muy concretamente a un modelo no relacional, por lo que me atrevería a descartar que se pueda avanzar en esa línea, al menos de momento. Seguramente el punto más fuerte en el que se debe trabajar es en desarrollar un modelo relacional que se adapte lo más ajustadamente posible al esquema IFC y perfeccione el funcionamiento conjunto de estas dos tecnologías. Esto podría servir como base en un gran número de aplicaciones futuras que pretendan seguir la metodología BIM.

También, por supuesto, se puede empezar a desarrollar aplicaciones destinadas al uso empresarial mucho más elaboradas y específicas que el ejemplo aquí presentado. De esta forma se empezaría a dar salida a esta tecnología también en el mercado.

Capítulo 7.
Summary and Conclusions

After this depth entering into BIM and IFC issue, it has given the impression that the technology is in a too earlier development phase to begin developing complex projects based on these systems. I consider advance and develop as important as ensure proper operation and use of what is already done, and it hasn’t being achieved at the moment. For example, it makes no sense to be able to store that amount of information if the responsible don’t store it correctly or don’t even know how to do it.

I think before venturing into major development projects, there should be more research projects parallel to the growth of the technology in order to polish the capabilities that these systems can provide to users. Of course, we face the great handicap of evolution as the natural human resistance against changes. But, after all, those systems users are the ones that can reach the most advantages for learning how to use them efficiently, so there should be a constant and disciplined way towards these goals kept by members of this professional sector.

Another aspect that draws attention is that we are mixing technical duties included, a priori, in a computing context, with building projects. It is possible that from now on partnerships between professionals from both sectors would be not only more frequent but also necessary. This certainly is interesting for the many possibilities it generates.

As for the technical part, the main conclusion that can be drawn is not because something is bang on trend, such as NoSQL databases, it means that is better than what went before. In this case it has been reflected perfectly: we have faced a problem that the structure of the standard that supports it, seemed to be a clear candidate for an object-oriented model. After analyzing the results we saw that has not happened because there is a couple of variables that marks which data model should be used and which not, and they should be all consider as a whole to make a correct decision.

As future work, it seems clear that there is no nonrelational data model that matches better with the problems analyzed in this project, so I would say that way of investigation is not available, at least for now. Surely the stronger point to work on is to develop a relational model that fits the IFC scheme as closely as possible and improves the co working of these two technologies. This could serve as a starting point for many future applications seeking to follow BIM methodology.

Also, of course, you can start developing any application intended for business use, much more elaborated and specific than the example given in this project. That would be the way to introduce these apps in the market.

Capítulo 8. Presupuesto

8.1 Tiempos de ejecución por fase

Fase	Tiempo aproximado
Investigación inicial	80 Horas
Implementación de la herramienta de testeo	30 Horas
Pruebas y análisis de los resultados	8 Horas
Implementación de la app cliente	20 Horas

Tabla 8.1. Tiempos de ejecución por fase.

8.2 Presupuesto

Concepto	Coste aproximado
PC Portátil (Intel Core 2 Duo 2,20 GHz; 3 GB RAM DDR3; 320 GB SATA HDD)	250,00 €
Hora de trabajo	6,00 €
PC Sobremesa (Intel Core 2 Duo 3 GHz; 3 GB RAM DDR2; 500 GB SATA HDD)	220,00 €
Licencias de software	Sin coste

Tabla 8.2. Presupuesto del proyecto.

Referencias

1. Building Information Modeling (s.f). En Wikipedia. Recuperado el 19 de Julio de 2017 de https://en.wikipedia.org/wiki/Building_information_modeling

2. Industry Foundation Classes (s.f). En Wikipedia. Recuperado el 19 de Julio de 2017 de https://en.wikipedia.org/wiki/Industry_Foundation_Classes

3. BuildingSMART International Ltd. (2008-2017) BuildingSmart. International home of openBIM. Recuperado el 19 de Julio de 2017 de http://www.buildingsmart-tech.org/

4. Tecnoteca srl (s.f) OpenMaint. Recuperado el 19 de Julio de 2017 de http://www.openmaint.org/en

5. Colegio Oficial de Arquitectos de Canarias, demarcación de Tenerife, La Gomera y el Hierro (s.f) Manual de uso y mantenimiento del edificio. Recuperado de http://www.coacyle.com/descargas/cat_coacyle_1218537402.pdf

6. Data Model (s.f). En Wikipedia. Recuperado el 19 de Julio de 2017 de https://en.wikipedia.org/wiki/Data_model

7. Relational Model (s.f). En Wikipedia. Recuperado el 19 de Julio de 2017 de https://en.wikipedia.org/wiki/Relational_model

8. Moreno, A (2000) Diseño e implementación de un lexicón computacional para lexicografía y traducción automática [Figura] Recuperado de http://elies.rediris.es/elies9/4-2-3.htm

9. Object database (s.f). En Wikipedia. Recuperado el 19 de Julio de 2017 de https://en.wikipedia.org/wiki/Object_database

10. Object database (s.f). En Wikipedia [Figura] Recuperado de https://en.wikipedia.org/wiki/Object_database

11. Key-value database (s.f). En Wikipedia. Recuperado el 19 de Julio de 2017 de https://en.wikipedia.org/wiki/Key-value_database

12. Alberto, J. (7 de marzo de 2009) Bases de datos clave-valor [Mensaje en un blog] Recuperado de http://softinspain.com/desarrollo/bases-de-datos-clave-valor/

13. Liebich, T. (2013) BuildingSMART_IFC4_WhatIsNew [Figura]. Recuperado de http://www.buildingsmart-tech.org/specifications/ifc-releases/ifc4-release/buildingSMART_IFC4_Whatisnew.pdf/view

14. BuildingSMART International Ltd (1996-2013) Industry Foundation Classes Release 4 (IFC4) [Figura] Recuperado de http://www.buildingsmart-tech.org/ifc/IFC4/final/html/schema/ifcplumbingfireprotectiondomain/lexical/ifcfiresuppressionterminal.htm

15. BuildingSMART International Ltd (1996-2013) Industry Foundation Classes Release 4 (IFC4) Recuperado de http://www.buildingsmart-tech.org/ifc/IFC4/final/html/

16. MariaDB (2017) Enterprise Open Source Database and Data Warehouse. Recuperado de https://mariadb.com/

17. ObjectDB Software (2010-2017) ObjectDB – Fast Object Database for Java – with JPA/JDO support. Recuperado de http://www.objectdb.com/

18. Apache Cassandra (s.f.) En Wikipedia. Recuperado el 19 de Julio de 2017 de

https://en.wikipedia.org/wiki/Apache_Cassandra
19. Apstex (2013) IFC TOOLS Project. Recuperado de http://www.ifctoolsproject.com/

20. Oracle Corporation (2017) Netbeans IDE. Recuperado de https://netbeans.org/

21. Google Inc. (2017) Android Studio. Recuperado de https://developer.android.com/develop/index.html

22. Solibri Inc. (2017) Solibri Model Viewer. Recuperado de https://www.solibri.com/products/solibri-model-viewer/

23. Autodesk Inc. (2017) Autodesk Revit. Recuperado de http://www.autodesk.es/products/revit-family/overview

24. BimObject (2013) BimObject Cloud. Recuperado de https://bimobject.com/es

25. Becker, A. (s.f.) HeidiSQL. http://www.heidisql.com/

DataStax Academy. 2017. DataStax Academy for Apache Cassandra https://academy.datastax.com/