eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Facultad de Informática

Grado de Ingeniería Informática

▪ **T**rabajo **F**in de **G**rado ▪

Ingeniería de Software

Utilización de Neo4J para la detección de "bad smells" en requisitos de sistemas embebidos

Gorka Salaberria Flaño

Junio 2020

Director:

Oscar Díaz

Resumen

En este Trabajo de Fin de Grado, se ha desarrollado e investigado la utilización de bases de datos de grafos para detectar inconsistencias que puedan surgir a la hora de definir los requisitos para un componente. Este proyecto se ha desarrollado para la empresa Developair, quienes utilizan un lenguaje propio para definir los requisitos de unos componentes con los que trabajan para los sectores ferroviarios, aeroespaciales, de automoción, energía, salud y fabricación. El manejo de los requisitos se trabaja en una aplicación propia de la empresa. Es por eso que se ha pensado en integrar una nueva tecnología para detectar posibles inconsistencias que surjan.

Para realizar esta tarea, lo primero que se ha hecho es buscar una tecnología que sea capaz de soportar estos requisitos planteados. A continuación, se ha estudiado el lenguaje propio de la empresa, para después investigar acerca de las inconsistencias que pueden surgir. Una vez trabajadas estas partes, se ha desarrollado un grafo capaz de responder a la búsqueda de las inconsistencias.

Finalizada la investigación, se ha desarrollado el proyecto en la tecnología seleccionada, para lograr detectar las inconsistencias sobre casos reales.

Finalmente, se ha pensado en la viabilidad del proyecto en el futuro y sobre cómo mejorarlo.

Agradecimientos

Antes de comenzar, me gustaría agradecer este proyecto a mi familia, quienes han sido un apoyo durante el desarrollo de este proyecto, a mi cuadrilla y amigos, con quienes he despejado la mente en los tiempos libres, a mi tutor por orientarme y ayudarme cuando lo he necesitado, y a mis compañeros de empresa, Jokin, Mikel y Alex, quienes me han ayudado cuando ha hecho falta.

Índice

1. Introducción 9

1.1. Descripción del proyecto 9
1.2. Contenido de la memoria 9

2. Planificación del proyecto 11

2.1. Alcance 11
2.2. Objetivos 11
2.3. Exclusiones 12
2.4. EDT 12
2.5. Gestión del tiempo 13
2.6. Gestión de riesgos 16

3. Contexto 19

3.1. Teoría de grafos 19
3.2. Neo4j 20
3.2.1. Bases de datos orientadas a grafos 20
3.2.2. ¿Por qué se ha elegido Neo4j? 20
3.3. Análisis del mercado 20

4. Análisis del proyecto 23

4.1. Parte a desarrollar 23
4.1.1. Grafos 24
4.1.2. Archivo de requisitos y JSON 24
4.1.3. Bad Smells 24
4.2. Parte externa 24
4.3. Parte a desarrollar en el futuro 24

5. Gramática de los requisitos 25

5.1. Lenguaje de requisitos 25
5.1.1. ¿Cómo se definen las constantes y variables? 25
5.1.2. ¿Qué tipos de ámbitos existen? 26
5.1.3. ¿Qué tipos de patrones existen? 26
5.1.4. Uso del lenguaje 27
5.2. Gramática BNF 28

6. Bad Smells 33

6.1. Nivel de acoplamiento 33
6.2. Bad Smells entre predicados 36
6.2.1. Cadena de dependencias con una larga longitud 36
6.2.2. Demasiados requisitos dependen de uno 37

6.2.3. Dependencias en ciclo 39

7. Desarrollo del proyecto 41

7.1. Desarrollo del grafo 41
7.1.1. Desarrollando el primer grafo 41
7.1.2. Grafo con el propósito de detectar Bad Smells 44
7.2. Desarrollo del JSON 47
7.2.1. Desarrollando el JSON a través de un ejemplo 47
7.2.2. Esquema del JSON 49
7.2.3. Transformar un archivo de requisitos a JSON 50
7.3. Desarrollo del comando para importar 51
7.4. Desarrollo de detección de los Bad Smells 51
7.4.1. Comando para la detección del nivel de acoplamiento 51
7.4.2. Detección de Bad Smells entre predicados 52
7.5. Pruebas sobre el desarrollo 53
7.5.1. Archivo de requisitos pequeño 53
7.5.2. Archivo de requisitos grande 54
7.5.3. Conclusión de los resultados 56
7.6. Problemas en el desarrollo 56

8. Gestión del proyecto 57

8.1. Gestión del alcance 57
8.2. Dedicaciones 57

9. Conclusiones 59

9.1. Metodologías utilizadas 59
9.2. Futuro del proyecto 59

10. Referencias bibliográficas 61

11. Anexos 63

Lista de Figuras, Tablas y Códigos

FIGURAS

Figura 1: EDT 13
Figura 2: Grafo no orientado 19
Figura 3: Grafo orientado 19
Figura 4: Análisis del dominio 23
Figura 5: Esquema del componente de ejemplo 27
Figura 6: Ejemplo para el cálculo del nivel de acoplamiento 34
Figura 7.1: Proceso del cálculo del nivel de acoplamiento 34
Figura 7.2: Proceso del cálculo del nivel de acoplamiento 35
Figura 7.3: Proceso del cálculo del nivel de acoplamiento 35
Figura 8: Ejemplo demasiadas dependencias 37
Figura 9: Ejemplo real demasiadas dependencias 38
Figura 10: Solución a demasiadas dependencias 38
Figura 11: Ejemplo ciclos 39
Figura 12: Ejemplo real ciclos 40
Figura 13: Composición de un requisito 42
Figura 14: Composición del patrón 43
Figura 15: Composición del predicado 43
Figura 16: Fallo detección de Bad Smells 45
Figura 17: Solución modelo de detección de Bad Smells 45
Figura 18: Modelo de grafo mejorado 46
Figura 19: Modelo de grafo orientado a los Bad Smells 46
Figura 20: Esquema proporcionado por Neo4j 52
Figura 21: Requisito en Neo4j 54
Figura 22: Requisito en Neo4j 55

TABLAS

Tabla 1: Dedicaciones paquetes de trabajo 14
Tabla 2.1: Periodo de desarrollo de tareas 15
Tabla 2.2: Periodo de desarrollo de tareas 15
Tabla 3: Gramática BNF de las constantes y variables 29

Tabla 4.1: Gramática BNF de los requisitos 30
Tabla 4.2: Gramática BNF de los requisitos 31
Tabla 5: Comparativa de dedicaciones 57

CÓDIGOS

Código 1: Benchmark del componente 28
Código 2: Ejemplo de un requisito 47
Código 3: Ejemplo JSON 48
Código 4.1: Esquema JSON 49
Código 4.2: Esquema JSON 50
Código 5: Comando ejecución script 51
Código 6: Comando para importar un JSON 51
Código 7: Comando para el nivel de acoplamiento 52
Código 8: Detección de alguna dependencia simple 53

1. Introducción

En este capítulo se desarrolla una descripción del proyecto, así como la motivación para realizarlo, además de comentar brevemente los capítulos contenidos en esta memoria.

1.1. Descripción del proyecto

La empresa Developair trabaja elaborando requisitos para diferentes componentes. Dependiendo de la cantidad de requisitos que se elaboren para un componente, podrían surgir problemas e inconsistencias entre estos, tal y como podrían ser anti-patrones, "Bad Smells" o errores a la hora de definir los requisitos. Al utilizar un lenguaje propio, elaborar un sistema capaz de detectar estos problemas es bastante complejo.

El proyecto se centra en la detección de "Bad Smells" mediante la conversión de requisitos en grafos. Por esa razón, se almacenarán en una base de datos de grafos, como es el caso de Neo4j. Las razones de la elección de esta tecnología se concretan más adelante.

Esta memoria plasma todo el proceso seguido para lograr una serie de objetivos acordados con la empresa.

1.2. Contenido de la memoria

- **Capítulo 1:** Contiene una pequeña introducción al proyecto y un esquema que muestra el contenido de la memoria.
- **Capítulo 2:** Se muestra la planificación que se ha hecho de cara a la realización del proyecto.
- **Capítulo 3:** Se desarrolla el contexto del proyecto, definiendo algunos aspectos importantes para la comprensión de la memoria.
- **Capítulo 4:** Se analiza el dominio del proyecto, definiendo los diferentes bloques para el desarrollo del proyecto y cuáles serán desarrollados.
- **Capítulo 5:** Se analiza el lenguaje propio de la empresa y se desarrolla una gramática para una mejor comprensión.
- **Capítulo 6:** Se definen los Bad Smells que se han acordado con la empresa, mostrando algunos ejemplos y soluciones propuestas para tratarlos.
- **Capítulo 7:** Se plasma todo lo desarrollado para la búsqueda de la solución al problema planteado.
- **Capítulo 8:** Se describe toda la gestión para llevar a cabo este proyecto.

- **Capítulo 9:** Se desarrollan las conclusiones sobre este proyecto, además de ver la viabilidad que tendrá en la empresa.

- **Capítulo 10:** Se hace referencia a toda la bibliografía utilizada para este proyecto.

2. Planificación del proyecto

Este capítulo muestra la planificación realizada. Este capítulo se divide en varios apartados, todos ellos orientados a la planificación del proyecto.

2.1. Alcance

Antes de comenzar a realizar el proyecto es recomendable establecer qué tareas se van a realizar, así como el tiempo que se invertirá en realizarlo.

En este proyecto, se pretende obtener una solución para la empresa Developair, cuyo objetivo es poder formalizar una gramática de requisitos a modo de grafo, y así poder detectar posibles conflictos que surjan entre ellos. Como conflictos, se ha pactado con la empresa detectar los conocidos como Bad Smells, que en capítulos posteriores se detalla qué son y cuáles serán los que se trate en el proyecto.

Esta tarea requerirá el uso de un sistema de gestión de bases de datos orientada a los grafos. La herramienta elegida para almacenar los grafos ha sido Neo4j, en el siguiente capítulo se detalla porqué ha sido seleccionada esta tecnología.

2.2. Objetivos

Los objetivos se dividen en objetivos de carácter personal y en objetivos asociados a la elaboración del proyecto.

Como objetivos de carácter personal se encuentran el aprender a utilizar un nuevo sistema de gestión de bases de datos orientada a grafos, Neo4j. Dentro de este objetivo también se encuentran el aprender el lenguaje de consulta Cypher[1], para poder operar sobre la base de datos. Además de esto, profundizar más sobre los archivos JSON y cómo manejarse con ellos, ya que para poder importar los requisitos a la base de datos se usarán archivos con esta extensión. Teniendo en cuenta lo mencionado anteriormente, dentro de los propósitos personales se incluye aprender a utilizar correctamente el plugin de Neo4j APOC[2], ya que mediante esta herramienta es posible importar un archivo JSON a la base de datos.

En cuanto a los objetivos de elaboración del proyecto, el objetivo principal es transformar una serie de requisitos a grafos para poder analizarlos y detectar conflictos que surjan entre ellos, además se intentará que la búsqueda de la solución sea viable y capaz de integrarse en las tecnologías de la empresa.

[1] Cypher: Lenguaje para trabajar con los grafos en Neo4j.
[2] APOC: Librería que contiene alrededor de 450 funciones para operar sobre los grafos en la base de datos.

2.3. Exclusiones

Debido al alcance tan amplio que podría tener el proyecto, se excluye la implementación del sistema de gestión del almacenamiento de datos en el sistema web propio de la empresa.

El objetivo del proyecto es lograr una representación de requisitos por medio de los grafos para poder detectar los conflictos que surgen. En el caso de que no se pueda llegar al objetivo final, se excluirá la detección de dependencias en requisitos de casos reales y como solución se harán pequeñas pruebas.

2.4. EDT

Una estructura de descomposición de trabajo (EDT) [3] sirve para descomponer por jerarquías un proyecto, y así poder identificar correctamente y fácilmente los paquetes de trabajo que se han de abordar. Este proyecto estaría compuesto por tres paquetes de trabajo, descritos a continuación:

1. **Desarrollo de tecnologías:** En este apartado se define todo lo relacionado con el desarrollo de las tecnologías que se han usado, tal y como son la adquisición de conocimientos, los primeros pasos realizados para desarrollar el grafo, importación de grafos, desarrollo de script para obtener JSON y detección de Bad Smells.

2. **Gestión del proyecto:** En este apartado se agrupan todos los activos desarrollados para realizar el proyecto de manera ordenada y poder hacer un seguimiento de este. Este bloque se descompone en otros dos bloques; la planificación del proyecto y el seguimiento y control.

3. **Defensa del proyecto:** En este apartado se abordarán los apartados de desarrollo de la memoria y la presentación del trabajo realizado.

[3] Estructura de descomposición del trabajo. (2020, 24 de mayo). *Wikipedia*, *La enciclopedia libre*. Fecha de consulta: 13:56, mayo 25, 2020 desde https://es.wikipedia.org/w/index.php?title=Estructura_de_descomposici%C3%B3n_del_trabajo&oldid=126333693.

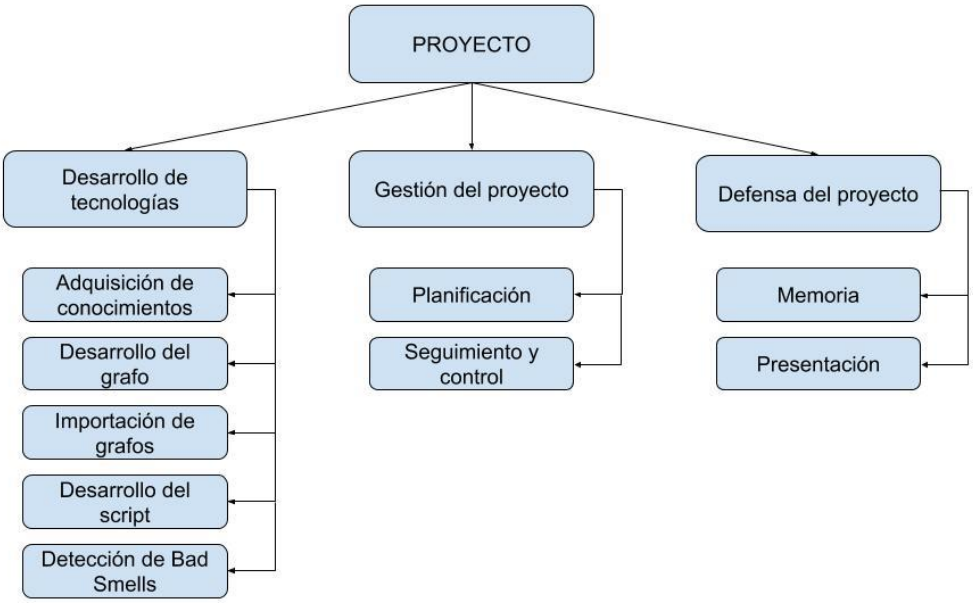

Figura 1: EDT

2.5. Gestión del tiempo

Conociendo las horas de dedicación y el plazo de entrega del proyecto, así como la identificación de los paquetes de trabajo, se podrá estimar la dedicación de tiempo a cada uno.

El tiempo de realización del proyecto está marcado en 300 horas, y el plazo de finalización propuesto el 01/06/2020. La fecha de inicio fue el 27/01/2020.

PAQUETES DE TRABAJO	DEDICACIÓN (horas)
Desarrollo de tecnologías	
Adquisición de conocimientos	50
Desarrollo del grafo	20
Importación de grafos	50
Desarrollo del script	30
Detección de Bad Smells	50
Total	200
Gestión del proyecto	
Planificación	15
Seguimiento y Control	20
Total	35
Defensa del proyecto	
Memoria	30
Presentación	20
Total	50
TOTAL PROYECTO	285

Tabla 1: Dedicaciones paquetes de trabajo

Además, para gestionar bien la dedicación del tiempo a lo largo de las semanas del proyecto, se ha desarrollado un diagrama de Gantt, el cual permite representar la dedicación a cada tarea.

El número de semanas entre el 27/01/2020 y el 01/06/2020 es de 18. El objetivo es trabajar entre 4 y 5 horas diarias de lunes a viernes.

	SEMANAS									
	Enero	Febrero				Marzo				
TAREAS	1	2	3	4	5	6	7	8	9	10
Adquisición de conocimientos										
Desarrollo del grafo										
Importación de grafos										
Desarrollo del script										
Detección de Bad Smells										
Planificación										
Seguimiento y Control										
Memoria										
Presentación										
Proyecto										

Tabla 2.1: Periodo de desarrollo de tareas

	SEMANAS							
	Abril				Mayo			
TAREAS	11	12	13	14	15	16	17	18
Adquisición de conocimientos								
Desarrollo del grafo								
Importación de grafos								
Desarrollo del script								
Detección de Bad Smells								
Planificación								
Seguimiento y Control								
Memoria								
Presentación								
Proyecto								

Tabla 2.2: Periodo de desarrollo de tareas

2.6. Gestión de riesgos

A la hora de desarrollar un proyecto se establece un número de horas para realizarlo en un plazo de tiempo, pero hay veces que surgen problemas en medio de la realización de este. Es por ello que es recomendable prevenir qué riesgos pueden surgir y darles una solución en caso de que ocurran. A continuación una lista de posibles riesgos y su solución:

1. Problemas con las tecnologías utilizadas: Al utilizar tecnologías nuevas, la probabilidad de que puedan surgir problemas es alta, tal y como pueden ser errores que no se entienden.

   Solución: La búsqueda a través de Internet por páginas y foros de confianza, o la consulta a expertos en estas tecnologías pueden ser la solución en caso de que surgiese un problema de este estilo.

2. Falta de conocimientos en algún tema: Cuando se abordan temas nuevos, es bastante probable que la falta de conocimiento sea un problema a afrontar.

   Solución: Al igual que el riesgo anterior, la consulta a expertos es la solución ante estos problemas.

3. Problemas personales: Durante el desarrollo del proyecto, uno podría sufrir problemas personales que puedan hacer que no sea posible la realización durante un tiempo indefinido.

   Solución: Cuando sea posible la recuperación de la situación, lo recomendable será recuperar el tiempo perdido con horas extras, o pidiendo un plazo para realizar todo como es debido.

4. Pérdida de todo lo realizado: Podría darse la casualidad que por un fallo, error o problema toda la documentación del proyecto se viese borrada.

   Solución: Para que esto no ocurra este proyecto cuenta con el soporte de tres tecnologías que respaldan copias de seguridad: la primera se trata de guardar el proyecto en la nube, por medio de DropBox, la segunda es tener el proyecto descargado en la computadora personal, y la tercera realizar cada semana una copia en un disco duro externo.

5. Alteraciones en el proyecto: Durante la realización del proyecto, por motivos de la empresa, motivos del director o asuntos personales, el proyecto podría cambiar su rumbo y producir así una desviación grande o pequeña.

   Solución: Sea un cambio grande o pequeño, la solución a este riesgo será dedicarle un tiempo extra para poder continuar todo al mismo ritmo.

6. Problemas académicos: Al estar cursando una asignatura no es un riesgo del que preocuparse, pero podría surgir que se deba dedicar tiempo a este, si surge un imprevisto.

Solución: En caso de que se pierda algo de tiempo para dedicárselo a la asignatura, se intentará recuperar este tiempo con horas extras.

3. Contexto

Este capítulo muestra el contexto del proyecto: los grafos, Neo4j, Bad Smells y el análisis del mercado para comprobar la originalidad del proyecto. Por medio de este capítulo se busca ayudar al lector a comprender los temas que se abordan en la memoria.

3.1. Teoría de grafos

Un grafo es un conjunto, no vacío, de objetos llamados vértices o nodos, y una selección de pares de vértices, llamados aristas que pueden ser orientados o no. La representación más común se hace mediante una serie de puntos (nodos) conectados por líneas (aristas).[4]

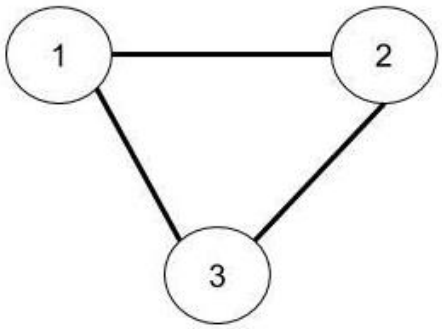

Figura 2: Grafo no orientado

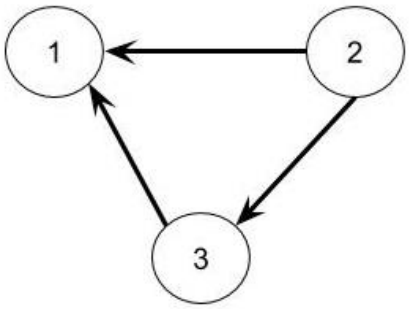

Figura 3: Grafo orientado

Matemáticamente un grafo se define como se define como **G=(V,E)**, donde **V** es el conjunto de vértices (nodos) , y **E** es el conjunto de aristas.

[4] Teoría de grafos. (2012,11 de julio). *Unipamplona* Fecha de consulta: 17:57, abril 14, 2020 desde http://www.unipamplona.edu.co/unipamplona/portalIG/home_23/recursos/general/11072012/grafo3.pdf

En el ámbito de la informática, los grafos son de gran utilidad para la resolución de problemas reales, ya que a diferencia de un esquema jerárquico, como un esquema tipo árbol, el esquema de los grafos es mucho más flexible, y gracias a eso es posible representar varios tipos de escenarios, tal y como podría ser un sistema de carreteras o el historial médico de toda una población.

3.2. Neo4j

Neo4j es un Sistema de Gestión de Bases de Datos (SGBDS) orientado a grafos. Su uso es muy común para la detección de fraudes, gestión de centros de datos, gestión de sistemas de datos maestros y recomendaciones en tiempo real.

3.2.1. Bases de datos orientadas a grafos

El sistema de gestión de este tipo de bases de datos (SGBD) tiene operaciones de crear, leer, actualizar y eliminar (CRUD es el acrónimo en inglés de estas operaciones: 'Create, Read, Update and Delete').[5]

3.2.2. ¿Por qué se ha elegido Neo4j?

Este proyecto requiere almacenar mucha información, y a su vez procesar esa información a gran velocidad. Es por eso que se pensó que utilizar una base de datos orientada a los grafos era la solución más apropiada, ya que tal y como se ha comentado anteriormente, estas bases de datos ofrecen un rendimiento excepcional, además de mucha escalabilidad[6].

Teniendo en cuenta los aspectos anteriormente mencionados, se buscó una tecnología capaz de cumplir con estos requisitos, y había una que los cumplía, Neo4j, que además tiene otros aspectos positivos, como es el soporte, ya que está constantemente actualizado ofreciendo mejoras y soluciones a posibles errores, además de una comunidad de usuarios muy implicada, que ante cualquier duda o problema, casi siempre tienen las soluciones.

La versión de Neo4j utilizada para este proyecto ha sido la 3.5.14, aunque también se han realizado pruebas en la 4.0.3, y el funcionamiento ha sido igual.

3.3. Análisis del mercado

Este proyecto investiga el desarrollo de una herramienta que se capaz de gestionar requisitos y detectar los Bad Smells que puedan surgir entre ellos. Dentro del mercado actual existen herramientas capaces de gestionar requisitos, pero no se ha encontrado ninguna capaz de detectar Bad Smells.

---

[5] Neo4j Graph Database Platform. 2020. *Why Graph Databases? - Neo4j Graph Database Platform*. https://neo4j.com/why-graph-databases/

[6] BBVAOpen4U. 2020. *Neo4j: Qué Es Y Para Qué Sirve Una Base De Datos Orientada A Grafos*. https://bbvaopen4u.com/es/actualidad/neo4j-que-es-y-para-que-sirve-una-base-de-datos-orientada-grafos

La herramienta más conocida para gestionar requisitos es Rational DOORS[7]. Esta herramienta ha sido desarrollada por IBM, y está pensada para la mejora de la gestión de un proyecto en equipo. IBM oferta su herramienta como una solución para que un proyecto se realice con eficacia reduciendo el tiempo y los costes[8].

Aunque esta herramienta sea una solución aceptable, Developair buscaba una herramienta capaz de detectar los Bad Smells que puedan surgir entre requisitos, además de poder hacer consultas para obtener información sobre ellos. Para ello se les propuso elaborar una herramienta nueva y original capaz de afrontar lo que ellos requerían.

[7] Página web sobre la herramienta Rational Doors: https://www.ibm.com/support/knowledgecenter/SSYQBZ_9.5.0/com.ibm.doors.requirements.doc/topics/c_welcome.html

[8] Vídeo presentación acerca de Rational Doors: https://www.youtube.com/watch?v=P2KKdCrejFc

4. Análisis del proyecto

Este capítulo muestra cómo será la estructura del proyecto, y qué aspectos serán los que habrá que desarrollar para lograr el alcance planteado.

El proyecto tiene como alcance lograr detectar los Bad Smells en un conjunto de requisitos, aunque, por otro lado su proyección hacia el futuro es amplia, y por ello se ha hecho un planteamiento de cómo sería una herramienta capaz de detectar los Bad Smells.

A continuación se muestra un análisis del dominio realizado con el equipo de Developair, definiendo la herramienta al completo

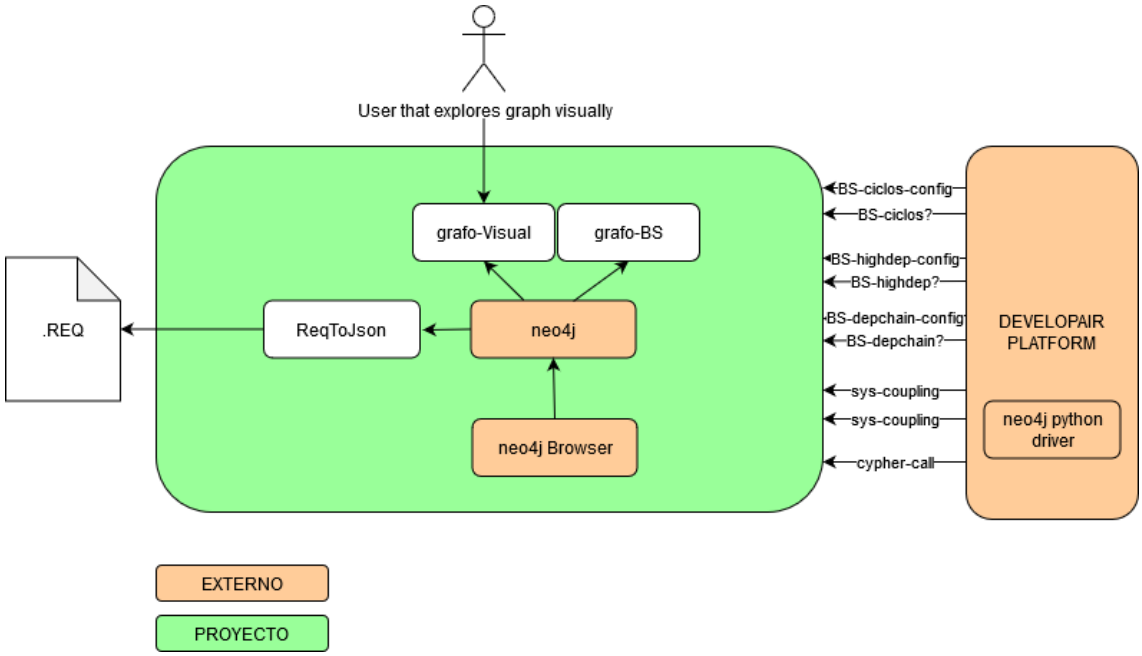

Figura 4: Análisis de dominio

4.1. Parte a desarrollar

Este apartado contiene todo lo que ha de desarrollarse para cumplir el alcance del proyecto, que se detalla en el capítulo 7. Este apartado se marca en la figura sobre el análisis del dominio dentro de un rectángulo verde.

4.1.1. Grafos

Esta parte del proyecto serían los grafos con los que se operaría desde el sistema. Uno de ellos es el grafo visual, que se puede ver en la figura 4 como grafo-Visual, el cual permitirá al usuario navegar a través de él para ver una representación gráfica del requisito.

El otro grafo sería orientado exclusivamente a la detección de los Bad Smells, en la figura 4 es el grafo-BS, que no estaría accesible para el usuario, ya que su propósito es detectar los Bad Smells eficientemente, por eso este grafo contendrá el mínimo de información posible.

La razón sobre porqué hay dos grafos se detalla en el capítulo 7, dónde se muestran los modelos de los grafos.

4.1.2. Archivo de requisitos y JSON

En esta parte del proyecto se obtiene un archivo de requisitos, y con un script de Python se transforma a un archivo JSON para poder importarlo a la base de datos de Neo4j.

Como puede apreciarse en la figura 4, este proceso se hace al principio con un archivo ".REQ", se le aplica la transformación en el apartado "ReqToJSON" para después meterlo en la base de datos, en Neo4j.

4.1.3. Bad Smells

Desde este apartado se desarrollarán los comandos de Cypher capaces de detectar los Bad Smells que se han acordado. Los comandos se probarán desde el navegador de Neo4j.

4.2. Parte externa

La parte externa hace referencia a todas las tecnologías que se usan para obtener el alcance del proyecto. Esto agrupa la tecnología de bases de datos que ofrece Neo4j, y el navegador de Neo4j para realizar las consultas sobre Bad Smells.

4.3. Parte a desarrollar en el futuro

Este apartado no se desarrollará en el proyecto, pero en caso de que se le dé viabilidad al proyecto, se llevará a cabo este apartado.

Esto trataría de implementar todo lo desarrollado en la plataforma de la empresa, para poder detectar los Bad Smells desde ahí, además de hacer consultas al grafo visual.

5. Gramática de los requisitos

En este capítulo se analiza el lenguaje propio de la empresa para desarrollar los requisitos, y a su vez escribirlo como una gramática BNF.

5.1. Lenguaje de requisitos

A la hora de definir los requisitos de un componente, la empresa trabaja con un lenguaje específico.

La agrupación de requisitos de un componente se denomina “benchmark”, y se agrupan en un archivo de extensión “.req”. Estos tipos de archivos son utilizados en la empresa dentro de una aplicación propia, y todos siguen una misma estructura.

Al comienzo del archivo hay una sección para definir la arquitectura del sistema. En este apartado se podrán definir constantes, variables de entrada, variables de salida y variables internas.

Después de haber definido la arquitectura del sistema vienen los requisitos. Los requisitos se identifican por un “id”, y posteriormente viene la definición del requisito, compuesto por un patrón y precedido por un ámbito. Tanto el ámbito como el patrón están definidos en el lenguaje, y serán los que puedan usarse para especificar los requisitos.

5.1.1. ¿Cómo se definen las constantes y variables?

A la hora de definir en el archivo una constante o una variable, hay que utilizar un modo específico que se muestra a continuación.

Definir una constante:
CONST *nombre_constante* IS *valor_constante*
Ejemplo: CONST MAIN_CONTACTOR_STATE_Open IS 4

Definir una variable de entrada:
Input *nombre_variable* IS *tipo_variable*
Ejemplo: Input MAIN_CONTACTOR_ACK IS int

Definir una variable de salida:
Output *nombre_variable* IS *tipo_variable*
Ejemplo: Output MAIN_CONTACTOR_CMD IS int

Definir una variable interna:
Internal *nombre_variable* IS *tipo_variable*
Ejemplo: Output MAIN_CONTACTOR_CMD IS int

5.1.2. ¿Qué tipos de ámbitos existen?

Como bien se ha dicho anteriormente, el ámbito se sitúa al principio, y como dice su nombre, especificará el ámbito donde operará el patrón.

- None
- Globally
- Before “{P}”
- After “{P}”
- Between “{P}” and “{Q}”
- After “{P}” until “{Q}”

Aunque existan seis ámbitos, normalmente se usa el de “Globally”.

5.1.3. ¿Qué tipos de patrones existen?

Los patrones se diferencian en tres grupos diferentes, pueden ser de ocurrencia, de orden o de tiempo real.

Ocurrencia:
- It is always the case that if "{R}" holds, then "{S}" holds as well
- It is never the case that "{R}" holds
- It is always the case that "{R}" holds
- Transitions to states in which "{R}" holds occur at most twice

Orden:
- It is always the case that if "{R}" holds then "{S}" previously held
- It is always the case that if "{R}" holds and is succeeded by "{S}", then "{T}" previously held
- It is always the case that if "{R}" holds then "{S}" previously held and was preceded by "{T}"

Tiempo real:
- It is always the case that once "{R}" becomes satisfied, it holds for at least "{S}" time units
- It is always the case that once "{R}" becomes satisfied, it holds for less than "{S}" time units
- It is always the case that "{R}" holds at least every "{S}" time units
- It is always the case that if "{R}" holds, then "{S}" holds after at most "{T}" time units
- It is always the case that if "{R}" holds, , then "{S}" holds for at least "{T}" time units
- It is always the case that if "{R}" holds for at least "{S}" time units, then "{T}" holds afterwards for at least "{U}" time units
- It is always the case that if "{R}" holds for at least "{S}" time units, then "{T}" holds afterwards
- It is always the case that if "{R}" holds, then "{S}" holds after at most "{T}" time units for at least "{U}" time units
- It is always the case that if "{R}" holds then "{S}" toggles "{T}"
- It is always the case that if "{R}" holds then "{S}" toggles "{T}" at most “{U}” time units later

5.1.4. Uso del lenguaje

Aunque hay diecisiete tipos de patrones, a la hora de definirlos se utilizan mayormente cuatro de ellos:

- “It is always the case that if "{R}" holds, then "{S}" holds after at most "{T}" time units” es utilizado en el 40% de los requisitos.
- “It is always the case that if "{R}" holds, then "{S}" holds as well” es utilizado en el 13% de los requisitos.
- “It is always the case that if "{R}" holds for at least "{T}" time units, then "{S}" holds afterwards” es utilizado en el 14% de los requisitos.
- “It is always the case that "{R}" holds” es utilizado en el 25% de los requisitos.

Dentro del ámbito y los patrones hay unos apartados que se encuentran entre comillas y corchetes, estos son los predicados, dónde se podrán insertar operaciones lógicas. Dentro de las operaciones lógicas se usarán operadores Booleanos básicos. A continuación se muestran cuáles pueden ser:

- “Or”: “||”
- “And”: “&&”
- “Not”: “!”
- Operadores de comparación: “==” | “!=” | “<” | “>” | “<=” | “>=”

Utilizando los operadores Booleanos, y las variables o constantes, se compone un predicado para el patrón o el ámbito, teniendo como resultado:
“Globally, It is always the case that if "(INPUT_1 && INPUT_2)" holds, then "OUTPUT_1" holds as well”.

A continuación se muestra como ejemplo el esquema de un componente, y posteriormente como sería su archivo “.req”.

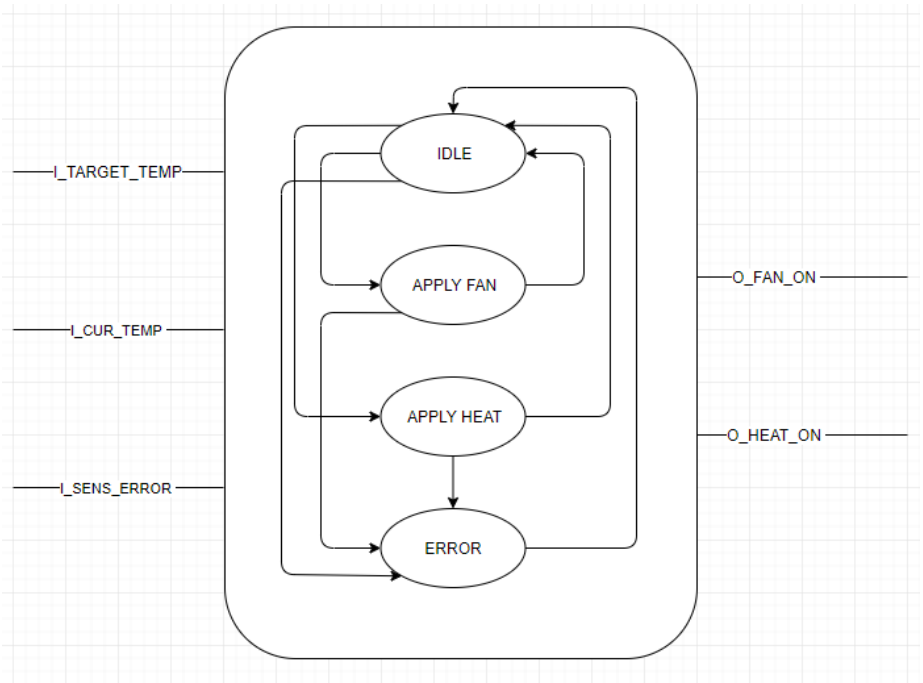

Figura 5: Esquema del componente de ejemplo

```
Input I_TARGET_TEMP IS int
Input I_CUR_TEMP IS int
Input I_SENS_ERROR IS bool
Output O_FAN_ON IS bool
Output O_HEAT_ON IS bool
Internal STATE IS int
CONST STATE_ERROR IS 0
CONST STATE_IDLE IS 0
CONST STATE_HEAT IS 0
CONST STATE_FAN IS 0

R8_0: Globally, it is always the case that if "I_SENS_ERROR == 1" holds,
then "STATE == STATE_ERROR" holds as well

R1_0: Globally, it is always the case that if "STATE == STATE_IDLE || STATE
== STATE_ERROR" holds, then "O_FAN_ON == 0 && O_HEAT_ON == 0" holds as well

R2_0: Globally, it is always the case that if "STATE == STATE_FAN" holds,
then "O_FAN_ON == 1" holds as well

R3_0: Globally, it is always the case that if "STATE == STATE_HEAT" holds,
then "O_HEAT_ON == 1" holds as well

R4_0: Globally, it is always the case that if "I_CUR_TEMP > I_TARGET_TEMP +
1" holds, then "STATE == STATE_FAN" holds after at most "1" time units

R5_0: Globally, it is always the case that if "I_CUR_TEMP < I_TARGET_TEMP
+ 1" holds, then "STATE == STATE_IDLE" holds after at most "1" time units

R6_0: Globally, it is always the case that if "I_CUR_TEMP  <= I_TARGET_TEMP
- 1" holds, then "STATE == STATE_HEAT" holds after at most "1" time units

R7_0: Globally, it is always the case that if "I_CUR_TEMP  >= I_TARGET_TEMP
- 1" holds, then "STATE == STATE_IDLE" holds after at most "1" time units

R9_0: Globally, it is always the case that if "STATE == STATE_HEAT &&
I_SENS_ERROR == 1" holds, then "STATE == STATE_ERROR" holds after at most
"1" time units
```

Código 1: Benchmark del componente

5.2. Gramática BNF

Una gramática BNF, conocida por la notación de Backus-Naur, es un metalenguaje utilizado para describir lenguajes formales.

Se ha desarrollado este tipo de gramática para entender mejor la definición del lenguaje y así poder comprender cómo están compuestos los archivos de requisitos. Esta gramática ha sido desarrollada a partir de la documentación acerca del lenguaje propio proporcionado por la empresa, además de varios ejemplos, también proporcionados por la empresa.

Por un lado se encuentra la gramática BNF de las constantes y las variables, la que se sitúa al principio del archivo, definiendo así la arquitectura del sistema.

Constante	::=	CONST **Nombre** IS **Numero**
Input	::=	Input **Nombre** IS **Tipo**
Output	::=	Output **Nombre** IS **Tipo**
Interna	::=	Internal **Nombre** IS **Tipo**
Nombre	::=	[a-zA-Z_0-9]*
Tipo	::=	int | bool
Numero	::=	[0-9]+

Tabla 3: Gramática BNF de las constantes y variables

Como se puede apreciar en la gramática primero estaría la definición de las constantes y las variables, y después la definición del nombre y su valor.

Por otro lado está la gramática de los requisitos, que aunque aparezcan en diferentes tablas, irían conjuntamente para definir todo el lenguaje.

Requisito	::=	**Id** : **Ambito**, **Patron**
Id	::=	[a-zA-Z_] [a-zA-Z_0-9]*
Ambito	::=	None | Globally | Before **Predicado** | After **Predicado** | Between **Predicado** and **Predicado**| After **Predicado** until **Predicado**
Patron	::=	It is always the case that if **Predicado** holds, then **Predicado** holds as well | It is never the case that **Predicado** holds | It is always the case that **Predicado** holds | Transitions to states in which **Predicado** holds occur at most twice | It is always the case that if **Predicado** holds then **Predicado** previously held | It is always the case that if **Predicado** holds and is succeeded by **Predicado**, the **Predicado** previously held | It is always the case that if **Predicado** holds then **Predicado** previously held and was preceded by **Predicado** | It is always the case that once **Predicado** becomes satisfied, it holds for at least **Predicado** time units | It is always the case that once **Predicado** becomes satisfied, it holds for less than **Predicado** time units | It is always the case that **Predicado** holds at least every **Predicado** time units | It is always the case that if **Predicado** holds, then **Predicado** holds after at most **Predicado** time units | It is always the case that if **Predicado** holds, then **Predicado** holds for at least **Predicado** time units | It is always the case that if **Predicado** holds for at least **Predicado** time units, then **Predicado** holds afterwards for at least **Predicado** time units | It is always the case that if **Predicado** holds for at least **Predicado** time units, then **Predicado** holds afterwards | It is always the case that if **Predicado** holds, then **Predicado** holds after at most **Predicado** time units for at least **Predicado** time units | It is always the case that if **Predicado** holds then **Predicado** toggles **Predicado** | It is always the case that if **Predicado** holds then **Predicado** toggles **Predicado** at most **Predicado** time units later |
Predicado	::=	**LPAREN Predicado RPAREN** | **NOT Predicado** | left=**Predicado** op=**Comparador** right=**Predicado** | left=**Predicado** op=**Binario** right=**Predicado** | **Bool** | **Identificador** | **Decimal**
Comparador	::=	**GT** | **GE** | **LT** | **LE** | **EQ**
Binario	::=	**AND** | **OR**
Bool	::=	**TRUE** | **FALSE**

Tabla 4.1: Gramática BNF de los requisitos

AND	::=	"&&"
OR	::=	"||"
NOT	::=	"!"
TRUE	::=	"TRUE"
FALSE	::=	"FALSE"
GT	::=	">"
GE	::=	">="
LT	::=	"<"
LE	::=	"<="
EQ	::=	"=="
LPAREN	::=	"("
RPAREN	::=	")"
Decimal	::=	'-'? [0-9]+ ('.' [0-9]+)?
Identificador	::=	[a-zA-Z_] [a-zA-Z_0-9]*

Tabla 4.2: Gramática BNF de los requisitos

En esta gramática BNF lo primero se define el requisito, para después ir descomponiéndolo poco a poco y obtener mayor granularidad, hasta poder definir correctamente como está estructurado un predicado.

6. Bad Smells

Este capítulo muestra los Bad Smells que se acordaron con la empresa.

6.1. Nivel de acoplamiento

Este Bad Smell busca obtener una media sobre cuántas veces comparten los requisitos sus variables en un grafo.

El cálculo de esta media se realiza de la siguiente forma:

1. Se selecciona una variable que todavía no haya sido seleccionada.
2. Se cuenta por cada variable del requisito las veces que está siendo compartida por otros requisitos, contando duplicados.
3. Se suma el recuento anterior al recuento general de todos los requisitos.
4. En caso de que queden requisitos que no hayan sido seleccionados se vuelve al paso 1, si no se continúa.
5. Teniendo el recuento general se divide por el número de requisitos que haya en el grafo para obtener el nivel de acoplamiento.

A continuación un ejemplo sobre cómo se calcularía:

Partiendo de un ejemplo y su grafo se haría el proceso de cálculo.

- R1_0: Globally, it is never the case the “STATE == STATE_ERROR”
- R2_0: Globally, it is always the case that “STATE_ERROR > 1”
- R3_0: Globally, it is never the case that “STATE_ERROR == 1 && STATE== STATE_ERROR”

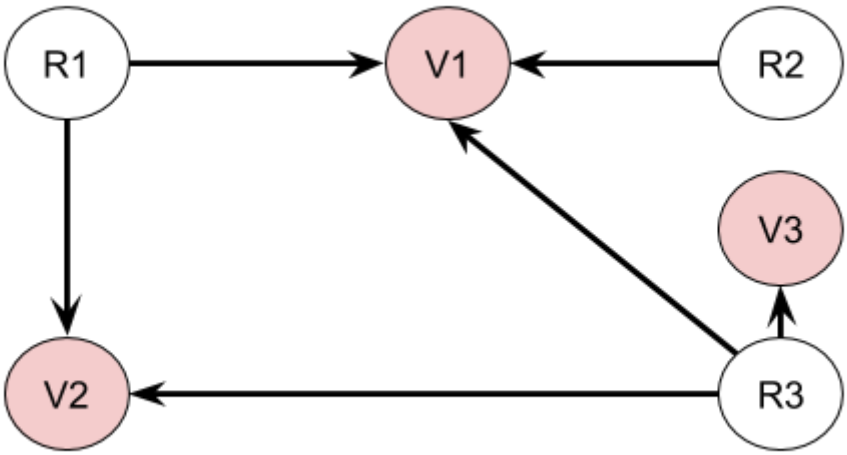

Figura 6: Ejemplo para el cálculo del nivel de acoplamiento

Se comienza por el requisito número uno (R1) y se miran las variables que están siendo compartidas por otros requisitos (V1 y V2). Por cada variable compartida, se analiza las veces que se comparte por otros requisitos; en V1 se comparte con R2 y R3, y en V2 con R3, y como no se descartan los duplicados, el R1 comparte sus variables tres veces. Esto se suma al recuento general, que hasta ahora era 0, por lo tanto ahora su valor es 3.

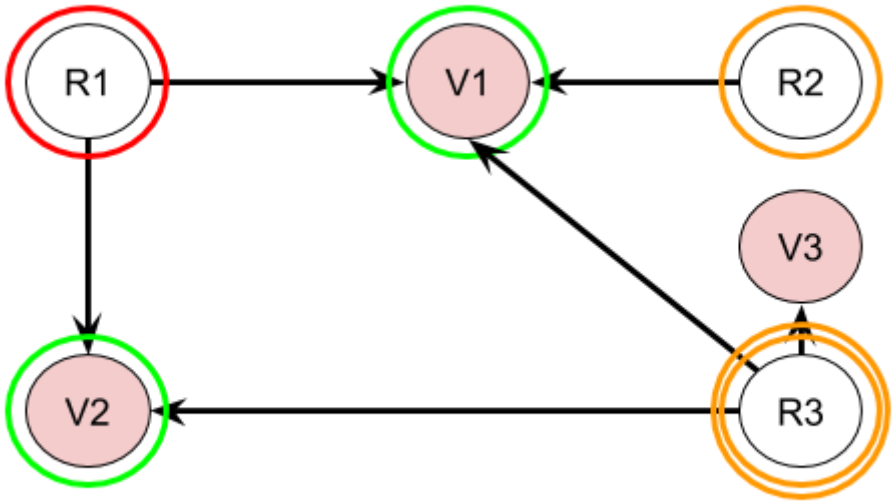

Figura 7.1: Proceso del cálculo del nivel de acoplamiento

Al haber requisitos que todavía no han sido seleccionados, se seleccionará el requisito número dos (R2), y analizando sus variables compartidas con otros requisitos, se ve que únicamente hay

una (V1), y esta es compartida con dos requisitos, por lo tanto sumándose al recuento general, ahora la suma estaría en 5.

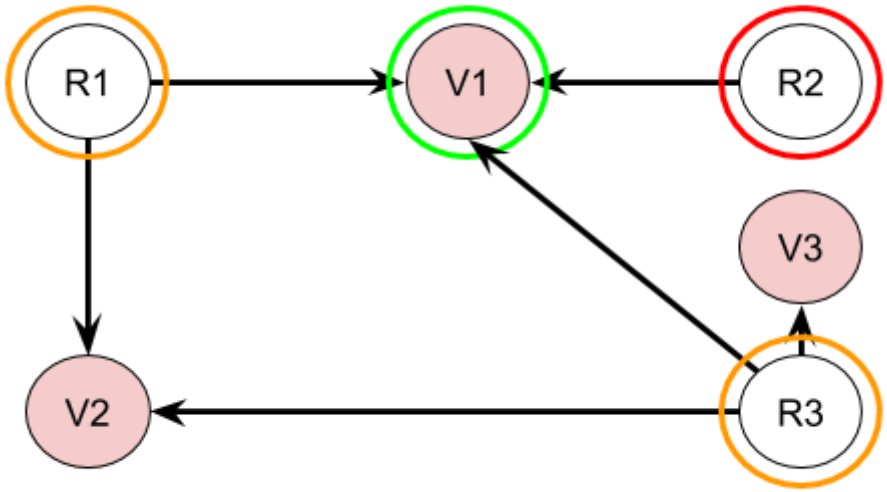

Figura 7.2: Proceso del cálculo del nivel de acoplamiento

Como todavía sigue habiendo requisitos sin seleccionar, se seleccionará el último (R3), y este requisito utiliza tres variables, pero únicamente dos de ellas son utilizadas también por otros requisitos (V1, por R1 y R2, y V2 por R1), por lo tanto le sumamos las veces que se comparte al recuento general, que en este caso es tres veces, teniendo así como valor del recuento 8.

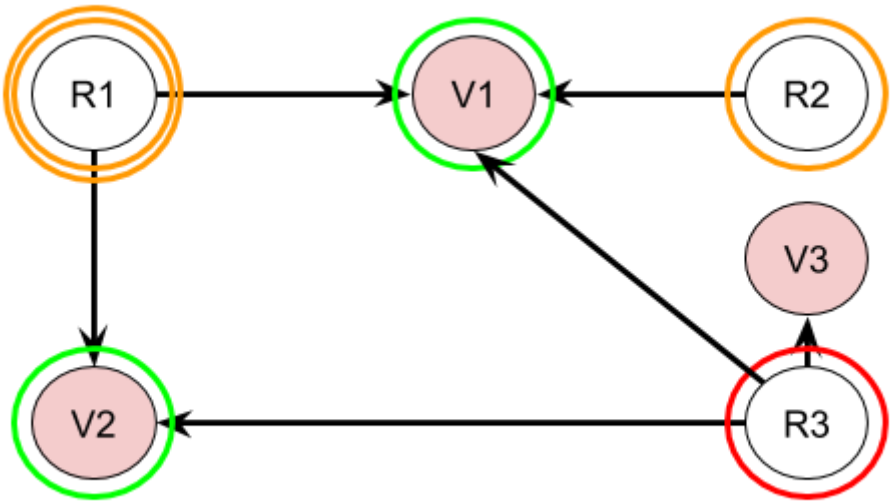

Figura 7.3: Proceso del cálculo del nivel de acoplamiento

Al no haber más requisitos que seleccionar, se hace un recuento de los requisitos que existen en el grafo, en este caso son tres, y se divide el recuento general por el número de requisitos para obtener el nivel de acoplamiento:
8/3 = 2,67 es el nivel de acoplamiento del grafo, que indica que cada requisito comparte sus variables 2,67 veces.

6.2. Bad Smells entre predicados

Este tipo de Bad Smells son los que surgen entre los predicados, concretamente en la parte del patrón de un requisito[9].

A la hora de detectar Bad Smells entre predicados, se acordaron definir tres tipos de Bad Smells en este contexto.

6.2.1. Cadena de dependencias con una larga longitud

Este es el caso donde un requisito hace que otro dependa de él, y este último hace que otro dependa de él, y así sucesivamente, provocando que se genere una larga cadena de dependencias.

R→R1→R2→R3→…
Donde R es un requisito

Para poder comprender mejor este tipo de dependencia se presenta un ejemplo a continuación.

- R1_0: Globally, it is always the case that if "I_SENS_ERROR == 1" holds, then "STATE == STATE_ERROR" holds as well
- R2_0: Globally, it is always the case that if “STATE== STATE_ERROR" holds, then "STATE == STATE_FAN" holds as well
- R3_0: Globally, it is always the case that if “STATE== STATE_FAN" holds, then "O_FAN_ON == 1" holds as well
- R4_0: Globally, it is always the case that if “O_FAN_ON == 1" holds, then "STATE == STATE_IDLE" holds as well
- R5_0: Globally, it is always the case that if “STATE == STATE_IDLE " holds, then "STATE == STATE_HEAT" holds as well

Estos tipos de requisitos tienen la misma estructura para poder hacer un ejemplo claro sobre esta dependencia; el primer predicado es quien “activa” al requisito, y el segundo predicado es “activado” por el requisito.

[9] Para llevar a cabo estas dependencias hay que tener en cuenta que los predicados desempeñan una función diferente dentro del patrón, algo que se contempla en el capítulo posterior. En este capítulo no es necesario definir porqué hay diferentes predicados, sólo tener en cuenta que los predicados pueden hacer surgir dependencias entre requisitos.

Por lo tanto viendo los requisitos, se aprecia que surge una cadena de dependencias entre requisitos de la siguiente manera:

R1_0→R2_0→R3_0→R4_0→R5_0

Solución para el Bad Smell:

Para solucionar este Bad Smell, se ha planteado que cuando se quieran detectar posibles dependencias de este tipo, se establezca un valor 'n', que cuando se supere ese valor salte una alerta. A continuación se muestra un ejemplo.

N=5
R1→R2→R3→R4→R5→R6

En el ejemplo se puede observar como la quinta dependencia se marca con un color diferente, indicando que se ha sobrepasado el número permitido de dependencias en cadena.

6.2.2. Demasiados requisitos dependen de uno

Este es el caso donde un requisito hace que otros dependan de él, provocando que puedan surgir problemas más adelante, como podrían ser las inconsistencias entre requisitos, o tener que revisar todos los requisitos que dependen de uno al editar este último, suponiendo un coste de tiempo muy grande en algunos casos.

Dónde R es un requisito

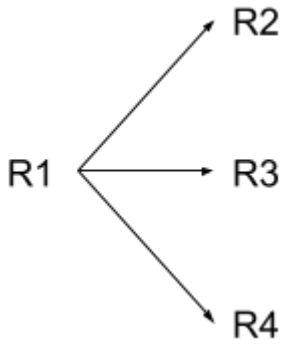

Figura 8: Ejemplo demasiadas dependencias

Para poder comprender mejor este tipo de dependencia se presenta un ejemplo a continuación.

- R1_0: Globally, it is always the case that if "I_SENS_ERROR == 1" holds, then "STATE == STATE_ERROR" holds as well

- R2_0: Globally, it is always the case that if “STATE== STATE_ERROR || O_FAN_ON == 2" holds, then "STATE == STATE_FAN" holds as well
- R3_0: Globally, it is always the case that if “STATE== STATE_ERROR || O_FAN_OFF == 3" holds, then "O_FAN_ON == 1" holds as well
- R4_0: Globally, it is always the case that if “STATE== STATE_ERROR || O_FAN_ON == 3" holds, then "STATE == STATE_IDLE" holds as well
- R5_0: Globally, it is always the case that if “STATE== STATE_ERROR || O_FAN_OFF == 5" holds, then "STATE == STATE_HEAT" holds as well

Estos tipos de requisitos tienen la misma estructura para poder hacer un ejemplo claro sobre esta dependencia; el primer predicado es quien “activa” al requisito, y el segundo predicado es “activado” por el requisito.

Por lo tanto, viendo los requisitos, se aprecia que el requisito R1_0 hace que todos los demás requisitos dependan de él:

Dónde R es un requisito

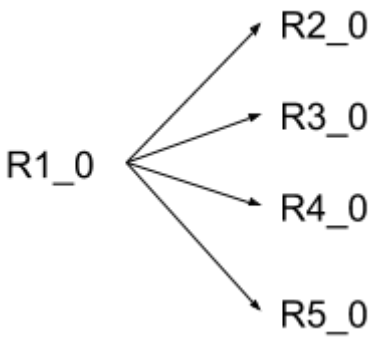

Figura 9: Ejemplo real demasiadas dependencias

Solución para el Bad Smell:

Para detectar este Bad Smell, se ha planteado una solución igual a la del Bad Smell del apartado 6.2.1, donde se establece un valor ‘n’, que cuando se supere el número de dependencias permitidas, salte una alerta.

Dónde R es un requisito

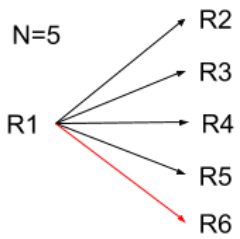

Figura 10: Solución a demasiadas dependencias

En el ejemplo se puede observar como la quinta dependencia que provoca el requisito R1 se marca con un color diferente, indicando que se ha sobrepasado el número permitido de dependencias desde un mismo requisito.

6.2.3. Dependencias en ciclo

Este es el caso donde las dependencias forman un ciclo. Este tipo de dependencias son las que más problemas podrían dar, ya que la reacción en cadena podría provocar que una serie de requisitos estuviesen constantemente "activos".

Dónde R es un requisito

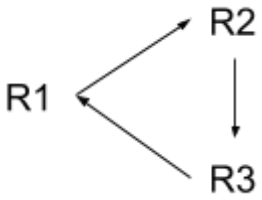

Figura 11: Ejemplo ciclos

Para poder comprender mejor este tipo de dependencia se presenta un ejemplo a continuación.

- R1_0: Globally, it is always the case that if "I_SENS_ERROR == 1" holds, then "STATE == STATE_ERROR" holds as well
- R2_0: Globally, it is always the case that if "STATE== STATE_ERROR" holds, then "STATE == STATE_FAN" holds as well
- R3_0: Globally, it is always the case that if "STATE== STATE_FAN " holds, then "O_FAN_ON == 1 && I_SENS_ERROR == 1 " holds as well
- R4_0: Globally, it is always the case that if "O_FAN_ON == 1" holds, then "STATE == STATE_IDLE" holds as well

Estos tipos de requisitos tienen la misma estructura para poder hacer un ejemplo claro sobre esta dependencia; el primer predicado es quien "activa" al requisito, y el segundo predicado es "activado" por el requisito.

Viendo el conjunto de requisitos se puede ver como surgen ciclos entre ellos, haciendo que un subconjunto esté siempre "activo":

Dónde R es un requisito

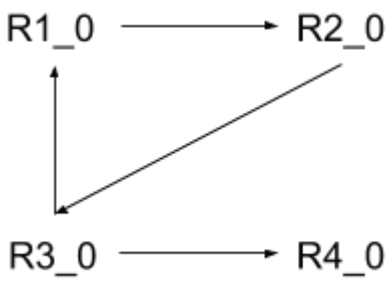

Figura 12: Ejemplo real ciclos

Solución para el Bad Smell:

Para solucionar este Bad Smell se ha planteado la solución más común para la detección de ciclos, el algoritmo 'Depth First Search' (DFS), o en castellano, búsqueda en profundidad.

Este algoritmo recorre todos los nodos del grafo, y es por eso que se podría detectar algún ciclo mediante su uso. El algoritmo empezaría en el primer nodo, y avanzaría nodo a nodo hasta completar el grafo; pero en el supuesto caso de que la arista le lleve a un vértice ya recorrido, nos indicaría que existe un ciclo en el grafo.

Siguiendo el ejemplo que se ha mostrado anteriormente, se realizará un DFS para ver si existen ciclos.

1. Lo primero que hay que hacer es tener un vector que almacene los nodos ya se hayan recorrido, para así poder saber si existen ciclos.
2. A continuación se empieza el algoritmo desde un nodo, en este caso empezaremos desde "R1_0", almacenamos el nodo en el vector, y el vector estaría compuesto de la siguiente forma: **['R1_0']**. Después vemos que su arista nos envía al siguiente nodo, "R2_0".
3. Ahora toca procesar el siguiente nodo, lo primero se almacena en el vector, quedándose así: **['R1_0', 'R2_0']**. Y repetimos lo mismo que hemos hecho antes, ver a dónde nos llevaría la arista.
4. La arista nos lleva al nodo "R3_0", lo almacenamos, **['R1_0', 'R2_0', 'R3_0']**, y pasamos al siguiente nodo al que nos lleve la arista.
5. En este caso hay dos aristas, así que vamos a procesar las dos. Por un lado tenemos el nodo "R4_0", que al incluirlo en el vector este sería su contenido: **['R1_0', 'R2_0', 'R3_0', 'R4_0']**. Y por el otro lado tenemos el nodo "R1_0", que ya ha sido procesado y se encuentra en el vector. Por lo tanto no almacenaríamos el nodo en el vector y pararíamos el algoritmo, ya que esto nos indica que se ha detectado un ciclo.

Y mediante este algoritmo, ya habríamos solucionado el problema de los ciclos en el grafo.

7. Desarrollo del proyecto

Este capítulo muestra el proceso de desarrollo con el propósito de lograr el alcance planificado. Para ello primero se ha desarrollado el modelo del grafo que se utilizará, después el proceso para convertir un archivo de requisitos en uno tipo JSON, y tras obtenerlo, importarlo a la base de datos. Una vez esté todo importado, se buscará si surgen Bad Smells.

7.1. Desarrollo del grafo

Es la parte más importante del proyecto y para que todo pueda realizarse correctamente habrá que desarrollar un grafo bien estructurado y con mucha escalabilidad. Para ello se desarrollará un primer modelo de grafo, para después mejorarlo según las especificaciones de la herramienta.

7.1.1. Desarrollando el primer grafo

El primer modelo del grafo se elaborará siguiendo una serie de pasos que permitirán sacarle el mayor rendimiento a esta primera versión a desarrollar.

Definir los requisitos del dominio

Este apartado ya ha sido desarrollado en el capítulo 4.

Ejemplo de datos para modelar

El ejemplo sobre los datos que hay que modelar se encuentra en el capítulo 5.

Definir preguntas para el dominio

Se harán unas preguntas mirando los ejemplos para poder desarrollar el primer modelo y mirar posteriormente si este es capaz de responderlas.

- ¿Qué es lo que compone un requisito y cómo se diferencian entre ellos?
- ¿Cómo están compuestos los predicados? ¿Y las variables?
- ¿Se pueden detectar los Bad Smells planteados?

Identificar las entidades a través de las preguntas

- Requisito:
    - Se identifican por un ID.
    - Están compuestos por un ámbito y un patrón.

- Ámbito:
    - Se identifica por el texto del ámbito que es.
    - Puede contener de uno a dos predicados.
- Patrón:
    - Se identifica por el texto del patrón que es.
    - Contiene de uno a cuatro predicados.
- Predicado:
    - Lo más habitual es que contenga variables
    - Puede contener operaciones lógicas.
    - Los predicados se identifican por el texto que lo componen
    - Contiene un indicador para saber qué predicado es dentro del ámbito o del patrón.
- Variables:
    - Se identifican por su nombre, el tipo que son y su valor

Identificar las conexiones entre las entidades

- Un requisito lo componen un ámbito y un patrón

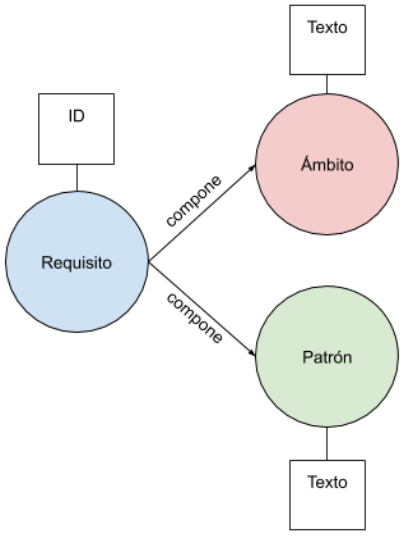

Figura 13: Composición de un requisito

- El ámbito puede estar compuesto por predicados, dependiendo del tipo que sea, pero el patrón siempre tiene uno, y máximo cuatro, dependiendo del tipo.

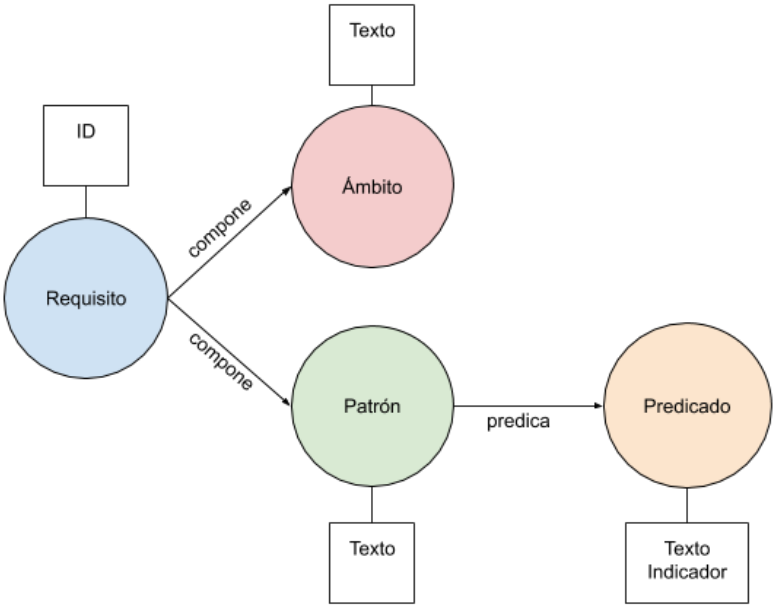

Figura 14: Composición del patrón

- Un predicado la mayoría de veces lo componen variables, así que un predicado usa variables.

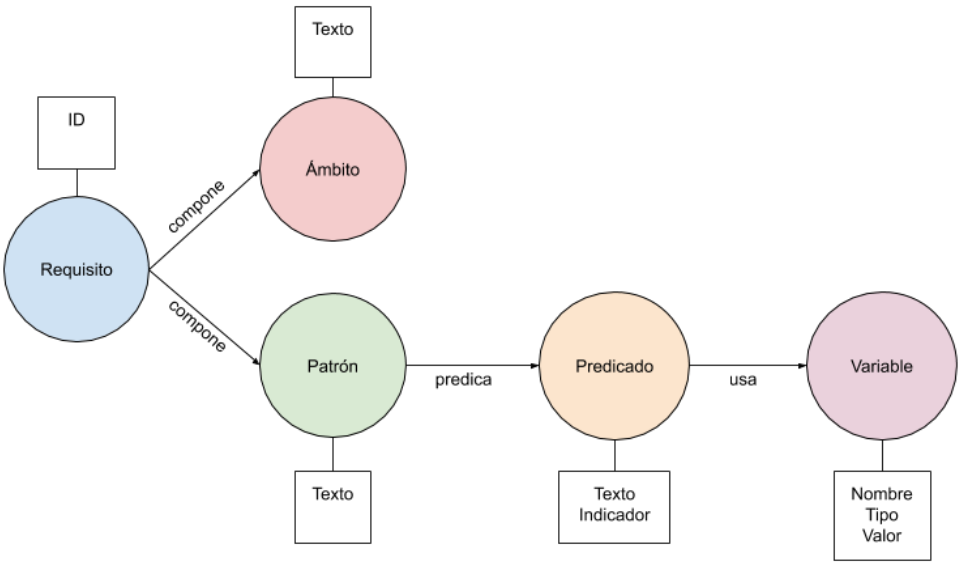

Figura 15: Composición del predicado

Verificar el grafo a través de las preguntas

- ¿Qué es lo que compone un requisito y cómo se diferencian entre ellos?

Un requisito está compuesto por dos nodos, el ámbito y el patrón, y la forma de diferenciar un requisito es el id, que es una propiedad del nodo del requisito.

Para acceder al patrón y el predicado de un requisito, bastaría con hacer una consulta preguntando por las relaciones que componen el requisito. Y para obtener el identificador del requisito se pregunta por la propiedad.

- ¿Cómo están compuestos los predicados? ¿Y las variables?

  Un predicado puede estar compuesto por otros nodos de variable, en el supuesto caso donde un predicado los usa. Además de eso, tiene como propiedades el texto del predicado y el indicador, que este último nos indica dónde se encuentra dentro del texto del ámbito o el patrón.

  Para acceder a los predicados desde el requisito, bastaría con consultar por los nodos que componen el requisito y los nodos que predican.

  Las variables están compuestas por tres propiedades, que son el nombre, el tipo y el valor.

  Para acceder a las variables desde los requisitos, habría que preguntar por los nodos que componen el requisito, por los predicados que lo predican y los nodos que usan los predicados.

- ¿Se pueden detectar los Bad Smells planteados?

  Este modelo de grafo desarrollado a partir de unos determinados pasos permitiría detectar únicamente el nivel de acoplamiento.

  En el siguiente punto se explica porqué no se pueden detectar todos los Bad Smells y una solución para poder obtenerlos.

7.1.2. Grafo con el propósito de detectar Bad Smells

Los Bad Smells que se quieren detectar son dependencias que surgen entre los predicados de los patrones, pero en este modelo de grafo, con más de un requisito y que haya dependencias entre ellos, no podría ser detectada la dependencia, ya que se necesita saber quién depende de quién. Véase con un ejemplo:

En un grafo con dos requisitos, dónde el requisito R1 tiene dos predicados P1 y P2, y el otro requisito R2 tiene dos predicados P1 y P3, hay una dependencia donde R1 hace que R2 dependa de él por medio del predicado P1. Esta descripción del grafo se representaría de la siguiente manera, pero de forma reducida, omitiendo el patrón:

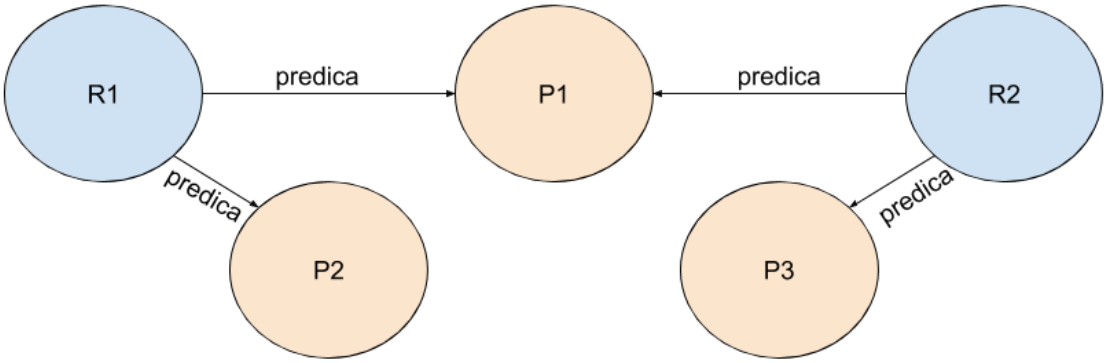

Figura 16: Fallo de detección de Bad Smells

El hecho de que sea cual sea la función que desempeñe un predicado dentro un requisito sea representado de igual manera en el grafo, hace que no puedan detectarse dependencias.

Un predicado debe diferenciarse según su función en el patrón. Por lo tanto, un predicado puede ser quien "active" un requisito o sea "activado" por el requisito.

Tras varios intentos con el primer modelo para la búsqueda de Bad Smells, se llegó a esta conclusión, que los predicados han de diferenciarse por "accionador", el que hace que el requisito se "active", y el "accionante" que es el que se "activa" cuando el requisito se cumple.

"*It is always the case that if "{R}" holds, then "{S}" holds as well*". En este ejemplo, el predicado "{R}" sería el "accionador", y el predicado "{S}" el "accionante", ya que en caso de que "{R}" se cumpla, posteriormente "{S}" será "activado".

Otro tipo de predicado, se trata del de tiempo. Este tiene la misma función que uno del tipo "accionante". "*It is always the case that if "{R}" holds for at least "{S}" time units, then "{T}" holds afterwards*", en este ejemplo el predicado "{T}" sería de tiempo, pero al igual que uno de tipo "accionante" es "activado" por el requisito.

Por lo tanto, para diferenciar la función que desempeñan los predicados, la solución planteada es modificar la dirección de la relación que hay entre el patrón y el predicado, dependiendo del tipo de predicado que sea. En caso de que sea "accionador", el predicado apuntará al patrón, y si es de tiempo o "accionante" será al revés.

Tomando el ejemplo anterior, el predicado P1 será "accionador" en R2 y "accionante" en R1.

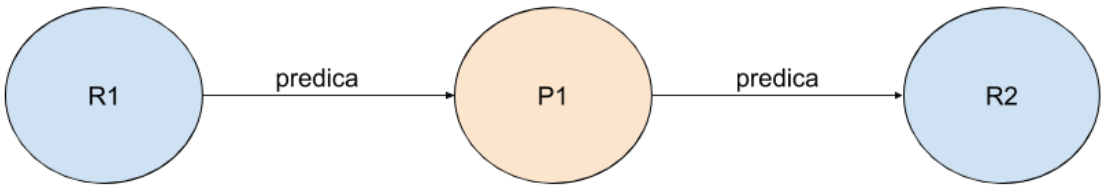

Figura 17: Solución modelo de detección de Bad Smells

Por lo tanto, el modelo del grafo quedaría de la siguiente manera, donde se ha resaltado la relación entre el predicado y el patrón, indicando que puede ser de cualquiera de las direcciones.

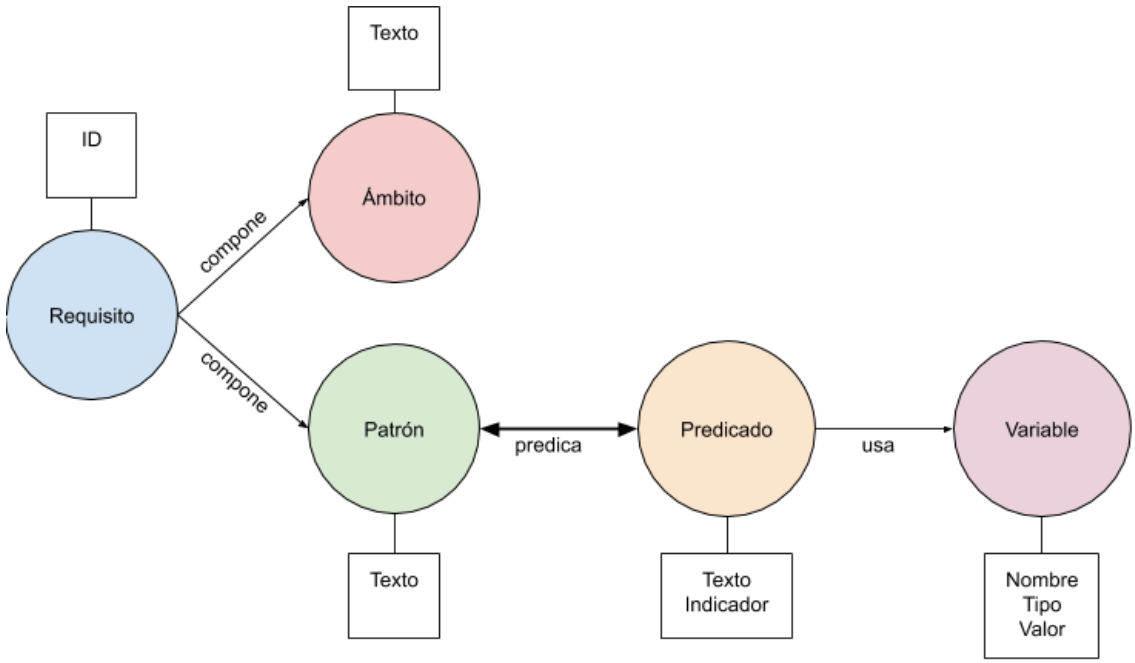

Figura 18: Modelo de grafo mejorado

Este modelo de grafo ya sería capaz de detectar todos los Bad Smells planteados. Pero el objetivo principal de este grafo es que únicamente sea capaz de poder soportar los comandos para la detección de los Bad Smells, por ello se podría reducir el modelo para que sea más eficiente.

Para detectar los Bad Smells únicamente se requiere de los requisitos, los predicados y las variables, y de las propiedades que los identifica, el id, el texto y el nombre respectivamente. Por lo tanto partiendo del anterior modelo y simplificando quedaría de la siguiente manera:

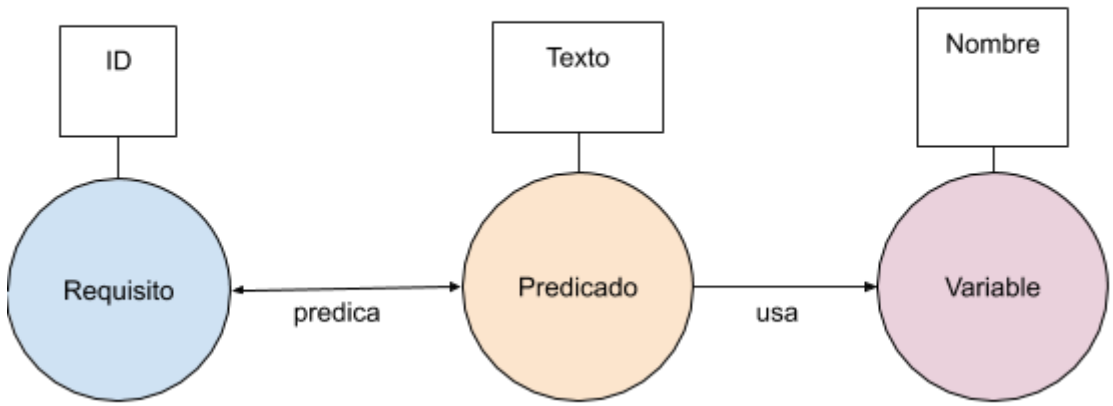

Figura 19: Modelo de grafo orientado a los Bad Smells

Así será el modelo de grafo final, el de la figura 19, con la suficiente granularidad para detectar los Bad Smells en los grafos.

7.2. Desarrollo del JSON

Para poder importar un benchmark entero de un componente a Neo4j, se hará por medio de un archivo JSON y el plugin para Neo4j APOC.

7.2.1. Desarrollando el JSON a través de un ejemplo

Como requisito de ejemplo, se ha cogido R8_0 del capítulo 5.

```
   Input I_SENS_ERROR IS bool
   Internal STATE IS int
   CONST STATE_ERROR IS 0
R8_0: Globally, it is always the case that if "I_SENS_ERROR == 1"
holds, then "STATE == STATE_ERROR" holds as well
```

Código 2: Ejemplo de un requisito

El requisito se identifica por su Id, que en este caso es “R8_0”.

Por otro lado estarían el ámbito y el patrón, que siguen una misma estructura, están compuestos por el texto, indicando el tipo que son según el lenguaje propio, y los predicados que tiene, contenidos en un “array” de predicados.

Contenido dentro del “array”, un predicado se diferencia por el texto, por su indicador, el cual nos dice dónde pertenece en el texto del ámbito o el patrón, el tipo de predicado que es, si es “accionador, “accionante” o de tiempo, y las variables que lo componen agrupados en un “array”.

Dentro del “array” de variables, se agrupan los objetos de variable, para saber qué variables utiliza cada predicado. El objeto de variable contiene la información acerca de la variable, que son, el nombre, el tipo de variable y su valor.

```
{
  "id_requisito" : "R8_0",
  "ambito": {
      "texto": "Globally",
      "predicados": []
  },
  "patron": {
      "texto": "it is always the case that if {R}, then {S} holds
      as well",
      "predicados": [
      {
      "predicado": {
            "texto": "I_SENS_ERROR == 1",
            "indicador": "R",
            "tipo_predicado": "accionador",
            "variables": [
                              {
                              "variable": {
                                    "nombre": "I_SENS_ERROR",
                                    "tipo": "input",
                                    "valor": "bool"
                                          }
                                          }
                              ]
                  }
                  },
                  {
                  "predicado": {
                  "texto": "STATE == STATE_ERROR",
                  "indicador": "S",
                  "tipo_predicado": "accionante",
                  "variables": [
                        {
                        "variable": {
                                          "nombre": "STATE",
                                          "tipo": "internal",
                                          "valor": "int"
                                          }
                                          },
                                          {
                                          "variable": {
                                          "nombre": "STATE_ERROR",
                                          "tipo": "constant",
                                          "valor": "0"
                                          }
                                          }
                              ]
                  }
                  }
      ]
  }
}
```

Código 3: Ejemplo de JSON

Este sería el resultado del objeto de requisito en el JSON, que iría contenido en un "array" de requisitos para representar todo el "benchmark". Se ha logrado obtener una granularidad alta,

para luego importar todo sin ningún problema. Aunque en el grafo de detección de Bad Smells no se requiere tanta información, para un desarrollo futuro tal vez sí se requiera toda la información.

7.2.2. Esquema del JSON

Viendo el ejemplo sobre cómo ha de ser un "benchmark" en un archivo JSON, se ha desarrollado un esquema JSON con el objetivo de comprender mejor su estructura.

```
{
  "requisitos":{
    "type": "array",
    "required": ["id_requisito", "ambito", "patron", "variables"],
    "items":{
      "type": "object",
      "properties":{
        "id_requisito":{
          "type": "string",
          "pattern" : "^R[0-9]{1,2}_0$"
        },
        "ambito":{
          "type": "object",
          "required": ["texto", "predicados"],
          "properties":{
            "texto": {"type": "string"},
            "predicados":{
              "type": "array",
              "items": {"$ref": "#pred"}
            }
          }
        },
        "patron":{
          "type": "object",
          "required": ["texto"],
          "properties":{
            "texto": {"type": "string"},
            "predicados":{
              "type": "array",
              "items": {"$ref": "#pred"}
            }
          }
        },
```

Código 4.1: Esquema JSON

```
          "definitions":{
            "pred":{
              "$id": "#pred",
              "type": "object",
              "required": ["predicado"],
              "properties":{
                "predicado":{
                  "type": "object",
                    "required": ["texto", "indicador",
"tipo_predicado", "variables"],
                    "properties":{
                      "texto": { "type": "string" },
                      "indicador": { "type": "string" },
                      "tipo_predicado": {"type": "string" },
                      "variables": {
                        "type": "array",
                        "items":{"$ref": "#var"}
                      }
                    }
                  }
                }
              },
              "var":{
                "$id": "#var",
                "type" : "object",
                "required": ["variable"],
                "properties":{
                  "variable":{
                    "type": "object",
                    "required": ["nombre", "tipo", "valor"],
                    "properties":{
                      "nombre": {"type" : "string"},
                      "tipo": {"type" : "string"},
                      "valor": {"type" : "string"}
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
}
```

Código 4.2: Esquema JSON

7.2.3. Transformar un archivo de requisitos a JSON

Un “benchmark” se almacena en un archivo con extensión “.req”, y para poder importarlo en la plataforma de Neo4j hace falta transformar en JSON según el esquema introducido anteriormente. Para ello, se ha desarrollado un script en Python, que tiene como parámetro de entrada el archivo “.req” y como salida el JSON.

El script desarrollado se encuentra en el anexo 1. Para llamar a la función, se tiene que tener instalado Python y desde un intérprete de comandos, por ejemplo la consola de Windows, se ejecuta el siguiente comando:

```
py nombre_script.py benchmark.req –o fichero_de_salida.json
```

Código 5: Comando ejecución script

7.3. Desarrollo del comando para importar

Una vez obtenido el JSON de un "benchmark", es hora de importarlo a la base de datos, y para ello hay que desarrollar una instrucción Cypher. Esta instrucción importará el "benchmark" a un grafo. Mediante el uso de la herramienta APOC se abre el archivo JSON, para después poder recorrerlo e ir importando y enlazando los nodos.

Nunca se utiliza la función "CREATE" de Neo4j, que lo que hace es crear. Siempre se utiliza "MERGE", que a diferencia del anterior, primero comprueba si existe el nodo o relación que se quiere crear, y en caso de que no exista ya en el grafo, lo crea.

El comando para importar se introducirá en la consola del navegador de Neo4j, y tras unos segundos todo se importará.

A continuación se muestra todo el comando para importar el JSON.

```
CALL apoc.load.json("archive_a_importar.json") YIELD value AS json
UNWIND json.requisitos AS requisito

MERGE (r:Requisito{nombre: requisito.id_requisito})
WITH *
UNWIND requisito.ambito AS ambito
UNWIND requisito.patron AS patron
UNWIND patron.predicados as patron_predicados
UNWIND patron predicados.predicado AS patron predicado
```

Código 6: Comando para importar un JSON

7.4. Desarrollo de la detección de los Bad Smells

Para la detección de los Bad Smells, se han desarrollado distintas instrucciones en Cypher.

7.4.1. Instrucción para la detección del nivel de acoplamiento

El comando para la detección del Bad Smells sobre el nivel de acoplamiento hace un recuento y una división de tipo 'float'. A continuación se muestra en la figura 20 el esquema del grafo que proporciona Neo4j y en el código 7 cómo es el comando desarrollado en Cypher.

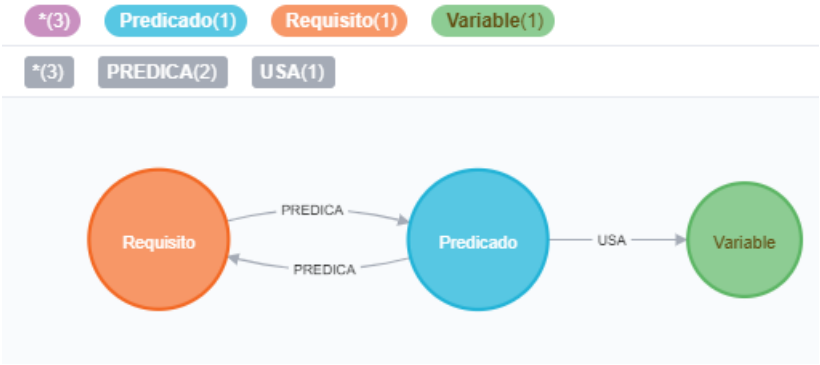

Figura 20: Esquema proporcionado por Neo4j

```
MATCH (n:Requisito)-[:PREDICA]-(p1:Predicado)-[:USA]->(v:Variable)<-
[:USA]-(p2:Predicado)-[:PREDICA]-(r:Requisito)
WITH count(r) AS re
MATCH (n:Requisito)
WITH re, count(n) as requisitos
RETURN re*1.0/requisitos
```

Código 7: Comando para el nivel de acoplamiento

El comando busca variables que están siendo utilizadas por dos requisitos diferentes. Después hace el recuento de todas las relaciones que ha encontrado de este estilo, para después tras encontrar el número requisitos, hacer la división de las relaciones por el número de requisitos, este último será un tipo 'float'.

7.4.2. Detección de Bad Smells entre predicados

Este apartado no ha podido ser finalizado, se han obtenido resultados y conclusiones pero no son totalmente verídicos. En este apartado se explicará lo que se ha logrado y cómo se seguirá investigando.

Cuando un predicado está compuesto por un 'AND' (&&), tienen que cumplirse ambos 'lados' para que sea cierto, es decir teniendo 'A && B', tanto 'A' como 'B' tienen que ser ciertos para que todo se cumpla, entonces, un predicado con esa estructura hará que otro predicado dependa de él, o al revés, solo si son iguales.

Por otro lado, si un predicado está compuesto por un 'OR' (||), basta con cumplirse uno de los 'lados' para que sea cierto, teniendo 'A || B', con tal de que 'A' sea cierto, o 'B' lo sea, ya se cumple todo. Por lo tanto, un predicado con esa estructura puede depender de uno que solo contenga 'A'.

En este último tipo de predicados es dónde surgen los problemas. Para poder detectar este tipo de dependencias habría que obtener mayor granularidad en los nodos de tipo predicado, y aquí es donde han surgido problemas.

Para obtener mayor granularidad se pensó en descomponer aún más el JSON y que quedase con un esquema que habría que importarlo recursivamente, pero tras investigar acerca de hacerlo de esta manera, no había forma de hacer un comando que pudiera importar recursivamente. Así que, como segunda opción, se planteó crear una función que se llamase en el comando de importar cuando hubiera que tratar el predicado, y que descompusiera el predicado en caso de que encontrase un 'OR'.

Esta última opción se ha empezado a investigar, aunque al llegar al límite de la fecha de finalización del proyecto no se ha podido concluir.

Para poder detectar si existe alguna dependencia de un predicado a otro se ha implementado un pequeño comando de Cypher que muestra si algún requisito depende de otro por medio de un predicado. El comando desarrollado es el siguiente:

```
match (n:Requisito)<-[:PREDICA]-(p:Predicado)<-[:PREDICA]-
(n2:Requisito) return *
```

Código 8: Detección de alguna dependencia simple

Por lo tanto, este apartado únicamente detecta dependencias de un requisito a otro, y queda pendiente de desarrollarlo en un futuro.

7.5. Pruebas sobre el desarrollo

Durante el desarrollo del proyecto se han hecho múltiples pruebas, pero para plasmar las pruebas en esta memoria se han elegido dos 'benchmarks, ambos de uso real proporcionados por la empresa.

7.5.1. Archivo de requisitos pequeño

Este 'benchmark' tiene en total 39 variables y constantes, además de 31 requisitos.

Conversión a JSON:

La conversión del archivo de requisitos a JSON se ha realizado en **0,038 segundos**, obteniendo un archivo de **2693 líneas**.

Importar a Neo4j:

La importación a Neo4j se ha realizado en **0,531 segundos** creando **121 nodos** y **261 relaciones**.

Muestra visual de un requisito:

Para ver que está bien importado, mediante un comando pedimos que se muestre el requisito con id "R3", cuya estructura es "*R3: Globally, it is always the case that if "STATE == STATE_Set_Cooking_Temp && PRESSED_LCD_RIGHT" holds, then "(DesiredTemp == DesiredTempPrev + 1) && STATE == STATE_Set_Cooking_Temp" holds for at least CONST_SLEEP time units".*

Y como resultado de Neo4j aparece lo siguiente:

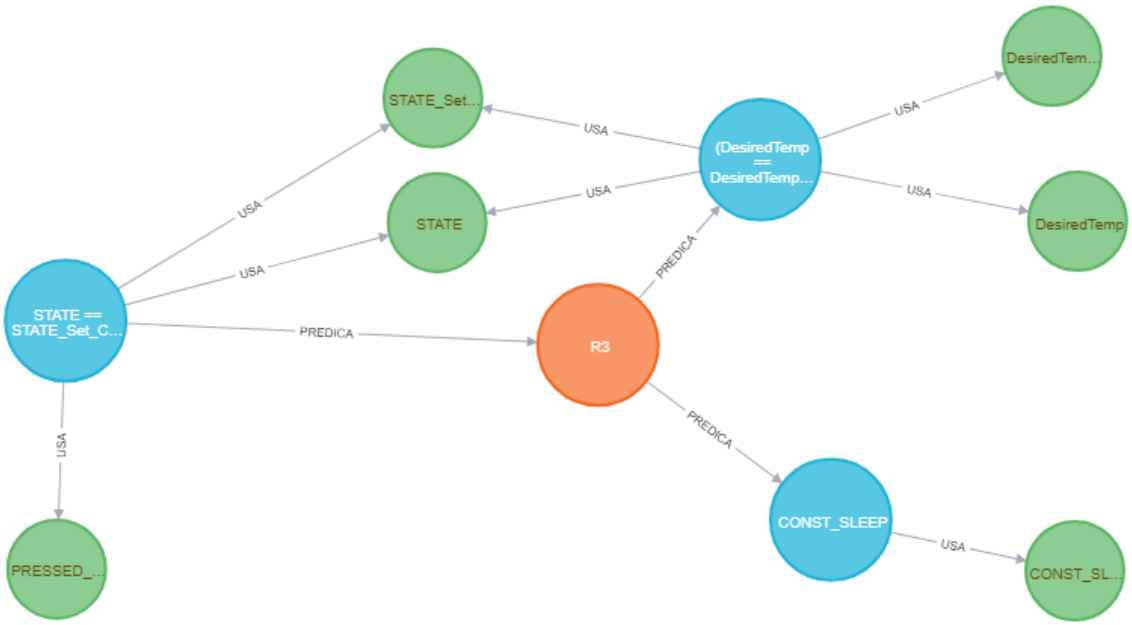

Figura 21: Requisito en Neo4j

Por lo tanto viendo el resultado y comparándolo con el texto se puede afirmar que está bien importado.

Detección del nivel de acoplamiento:

Se ha puesto en marcha el cálculo del nivel de acoplamiento en este 'benchmark' obteniendo como resultado **116,552** en un tiempo de **0,005 segundos**.

Detección de Bad Smells entre predicados:

Se ha ejecutado el comando que permite visualizar si existe alguna dependencia de requisitos en el grafo, y únicamente ha aparecido una. Después se ha buscado en el documento si existe algún predicado compuesto por algún 'OR', y no había ninguna. Entonces, visto esto, en este grafo no habría ningún Bad Smell entre predicados.

7.5.2. Archivo de requisitos grande

Este 'benchmark' tiene en total 140 variables y constantes, además de 107 requisitos.

Conversión a JSON:

La conversión del archivo de requisitos a JSON se ha realizado en **0,235 segundos**, obteniendo un archivo de **7270 líneas**.

Importar a Neo4j:

La importación se ha realizado en **0,371 segundos** creando **437 nodos** y **715 relaciones**.

Muestra visual de un requisito:

Para ver que está bien importado, mediante un comando pedimos que se muestre el requisito con id "AD_58_CpuApp_InputStage_99_0", cuya estructura es *"AD_58_CpuApp_InputStage_99_0: Globally, it is always the case that if "STATE != STATE_TEST " holds, then "OPERATIONAL_SUBSTATE ==OPERATIONAL_SUBSTATE_DISCONNECTED " holds after at most "CONST_TIME_CYCLE " time units".*

Y como resultado de Neo4j aparece lo siguiente:

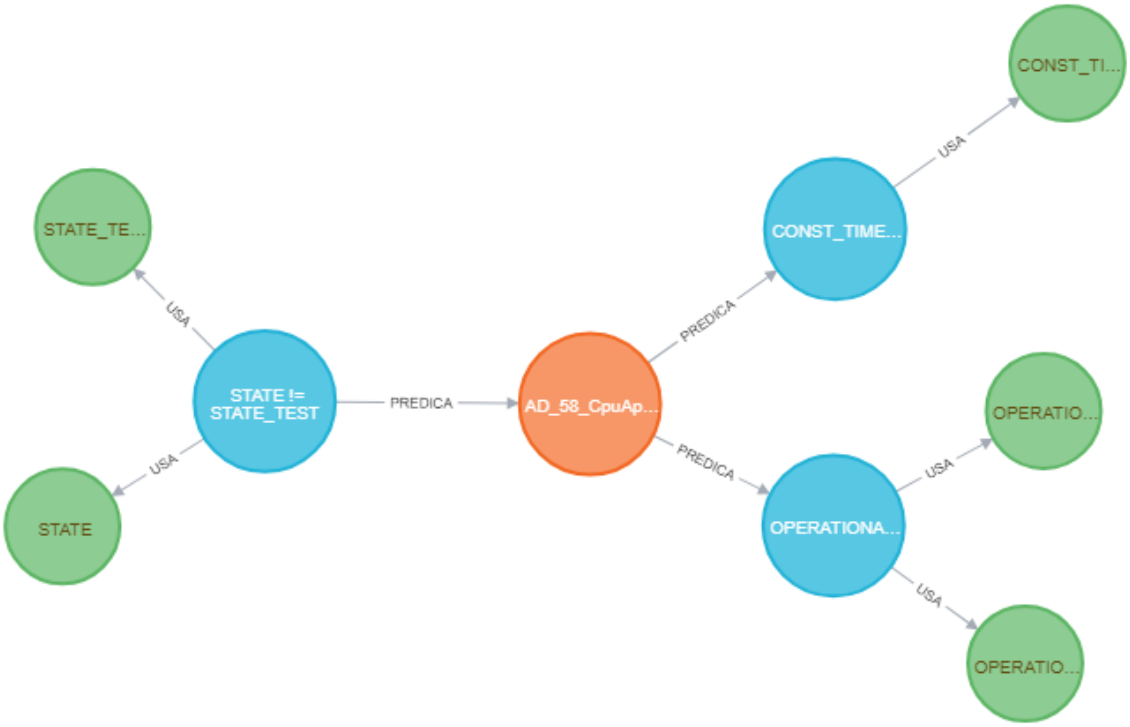

Figura 22: Requisito en Neo4j

Por lo tanto viendo el resultado y comparándolo con el texto se puede afirmar que está bien importado.

Detección del nivel de acoplamiento:

Se ha puesto en marcha el cálculo del nivel de acoplamiento en este 'benchmark' obteniendo como resultado **42,804** en un tiempo de **0,018 segundos**.

Detección de Bad Smells entre predicados:

Al igual que en el ejemplo anterior, se ha ejecutado el comando que permite visualizar si existe alguna dependencia de requisitos en el grafo, y únicamente ha aparecido una. Después se ha buscado en el documento si existe algún predicado compuesto por algún 'OR', y había algunos, pero ninguno de ellos está compuesto únicamente por 'OR', están mezclados con 'AND'. Entonces, visto esto, en este grafo no habría ningún Bad Smell entre predicados.

7.5.3. Conclusión de los resultados:

Viendo los resultados obtenidos con un archivo grande y uno pequeño, se puede apreciar cómo aunque la cantidad de datos aumente, el tiempo requerido para importar y detectar los Bad Smells casi no aumenta, además, el tiempo transcurrido es muy corto.

7.6. Problemas en el desarrollo

Durante todo el desarrollo han surgido muchos problemas, la mayoría relacionados con la detección de Bad Smells.

Para poder detectar los Bad Smells hacía falta que el modelo de grafo fuese el adecuado, y durante todo el desarrollo se han ido haciendo diferentes modelos hasta dar con el adecuado, esto a veces suponía que había que rehacer incluso el esquema del JSON, además de rehacer la mayoría de veces el comando para importar el JSON.

Una vez se encontró el modelo adecuado también surgieron problemas, tal y como se han comentado anteriormente. El más perjudicial ha sido el de los predicados con algún 'OR', aunque ya se ha encontrado la solución.

8. Gestión del proyecto

En este capítulo se muestra como ha sido gestionado el proyecto a lo largo de su desarrollo. Para que se refleje lo más correcto posible, se ha analizado la gestión sobre el alcance del proyecto, además de llevar un recuento sobre las horas de dedicación.

8.1. Gestión del alcance

Para llevar a cabo este proyecto se marcó un alcance a la hora de hacer la planificación, el cual se ha visto perjudicado en algunos aspectos. Esto ha podido deberse mayormente a la falta de conocimiento sobre las tecnologías utilizadas, porque la gestión del tiempo empleado ha sido correcta.

Este proyecto tenía como objetivo usar la teoría de grafos para detectar Bad Smells que podían surgir entre requisitos de un componente. Tal y como se ha comentado antes, la falta de conocimiento, ha hecho que no se haya podido completar del todo el apartado sobre detectar Bad Smells entre predicados, ya que se sobrepasaba demasiado el tiempo establecido de 300 horas.

Tal y como se ha mencionado, aunque no se haya podido completar todo el alcance, sí que se tienen los conocimientos sobre como poder finalizarlo en un futuro.

8.2. Dedicaciones

Al hacer la planificación se estimaron unas horas de dedicación a cada paquete de trabajo, y durante el desarrollo del proyecto se han ido contabilizando las horas reales dedicadas, y así poder verse reflejado todo en una tabla, y hacer una comparación con las horas estimadas.

PAQUETES DE TRABAJO	DEDICACIÓN real	DEDICACIÓN planeado	DESVIACIÓN
Desarrollo de tecnologías			
Adquisición de conocimientos	40	50	-10
Desarrollo del grafo	60	20	40
Importación de grafos	50	50	0
Desarrollo del script	30	30	0
Detección de Bad Smells	70	50	20
Total	250	200	50
Gestión del proyecto			
Planificación	10	15	-5
Seguimiento y Control	7	20	-13
Total	17	35	-18
Defensa del proyecto			
Memoria	35	30	5
Presentación	0	20	-20
Total	35	50	-15
TOTAL PROYECTO	302	285	17

Tabla 5: Comparativa de dedicaciones

Como se puede apreciar en la tabla, las dedicaciones reales han sido mayores en el desarrollo de tecnologías, por lo comentado sobre la falta de conocimientos. Los otros paquetes han requerido menor tiempo, aunque el de la defensa del proyecto tiene la presentación del proyecto a 0 ya que eso se realizará tras entregar esta memoria.

Aunque hayan subido los tiempos de dedicación, la dedicación total del proyecto ha sido de 302 horas, que en comparación con el tiempo marcado de 300 horas para realizarlo, solo ha aumentado 2 horas, que no es ni un 1% del total.

9. Conclusiones

En este capítulo se llevan a cabo las conclusiones sobre el proyecto. Por un lado se hablará sobre las metodologías utilizadas, y después sobre el futuro de la herramienta.

9.1. Metodologías utilizadas

El proyecto se ha llevado muy bien desde todos los aspectos. La relación que ha habido con la empresa ha sido excepcional, ayudando a organizar el proyecto y buscando soluciones cuando se ha hecho falta, además de ofrecer un lugar de trabajo en la oficina. Por otra parte, con el tutor del proyecto todo ha sido muy correcto, realizando las reuniones cuando han hecho falta, respondiendo a los emails cuando ha podido y siempre buscando la manera de ayudar.

Desde mi punto de vista como desarrollador, la planificación y la gestión han sido excelentes, llevando todo a tiempo, y finalizando en el plazo establecido. Además de esto, el aprendizaje que he logrado en este proyecto ha ido desde aprender a utilizar Neo4j con mucha soltura, a mejorar en el desarrollo en el lenguaje Python.

En resumen, este proyecto ha sido muy ameno, en ningún momento he parado de aprender, además de aprender mucho durante los meses de desarrollo.

9.2. Futuro del proyecto

Tal y como se planteó en el dominio, hay apartados que no han podido ser llevados a cabo por falta de tiempo, pero que si la viabilidad futura del proyecto fuese positiva, sí se desarrollarían.

En caso de ser viable, lo siguiente que habría que hacer para finalizar toda la herramienta sería finalizar el apartado de la detección de Bad Smells en predicados. Una vez hecho esto, el siguiente paso sería implementar la herramienta en la plataforma propia de la empresa, investigando sobre cómo implementar una base de datos de Neo4j en un sistema web, además de saber cómo interaccionar con ella.

Teniendo esto en cuenta, la opinión de la empresa ha sido que este proyecto podría tener futuro en su plataforma, y que por lo tanto su viabilidad sería positiva.

10. Referencias bibliográficas

[1] Díaz, Oscar & Piattini, Mario & Calero, Coral. (2001). Measuring triggering-interaction complexity on active databases. Information Systems. 26. 15-34. 10.1016/S0306-4379(01)00007-2.

[2] Post, Amalinda & Menzel, Igor & Podelski, Andreas. (2011). Applying Restricted English Grammar on Automotive Requirements - Does it Work? A Case Study.. 166-180. 10.1007/978-3-642-19858-8_17.

[3] Post, Amalinda (2011). Effective Correctness Criteria for Real-time Requirements, Chapter 1, Characteristics of Automotive Requirements, pp. 5-16.

[4] The Neo4j Team (2020). The Neo4j Cypher Manual v4.0 [Archivo PDF]. https://neo4j.com/docs/pdf/neo4j-cypher-manual-4.0.pdf

[5] The Neo4j Team. Graph Data Modeling for Neo4j [Diapositivas de PowerPoint]. https://www.dropbox.com/s/2q3koepmp5xcbmf/GraphDataModelingForNeo4j_AllSlides.pptx?dl=0#

[6] Neo4j. (31 de mayo de 2017). Graph Analysis over JSON Data — Omar Rampado, Larus [Archivo de vídeo]. Youtube. https://www.youtube.com/watch?v=scTszmVj-VM

[7] William Lyon. (10 de marzo de 2017). Loading JSON with Neo4j [Archivo de vídeo]. https://www.youtube.com/watch?v=iyjgOR7nBck

11. Anexos

ANEXO 1: Script para transformar a JSON

```
import json
import re
import sys
import argparse
import time

def convert_req2json(req_input: str):

    requisitos = []
    variables = []
    observables = {}
    requirements = []
    obs_regexp = re.compile(r"(input|output|const|internal) (.*?) is (.*?)\s?$",
                            re.IGNORECASE)
    req_regexp = re.compile(r"(\S*): (.*?)\s?$")
    obsgroups = {'input': "input",
                 'output': "output",
                 'const': "constants",
                 'internal' : "internal"}

    for line in req_input.splitlines():
        # Comment -> Ignore
        if line.startswith("//") or line.startswith("#"):
            continue

        # Variable definition
        obs_matches = obs_regexp.findall(line)
        if obs_matches:
            try:
                obstype, varname, type_or_value = obs_matches[0]
            except ValueError:
                continue

            obstype = obsgroups[obstype.lower()]
            fieldname = "value" if obstype == "constant" else "type"
            variables.append({"nombre":varname, "tipo_objeto":obstype, "tipo_valor": type_or_value})
            continue
        # Requirement
        req_matches = req_regexp.findall(line)
        if req_matches:
            try:
                reqid, reqtext = req_matches[0]
            except ValueError:
                continue

            ambito = {}
            patron = {}

            texto_dividido = reqtext.split(",", 1)
            texto_ambito = texto_dividido[0]
            texto_patron = texto_dividido[1]

            ambito = create_ambito(texto_ambito, variables)
            patron = create_patron(texto_patron, variables)

            requisitos.append({"id_requisito": reqid, "ambito": ambito, "patron": patron})
            continue

    component = {"requisitos": requisitos}

    return json.dumps(component, indent=4)
```

```python
def convert_req2json_main():
    ap = argparse.ArgumentParser()
    ap.add_argument("input_file", nargs='?',
                    help=".req file to convert. Default read from stdin.")
    ap.add_argument("-o", "--output-file", type=str,
                    help="path to output file. Default output to stdout")
    args = ap.parse_args()

    if args.input_file:
        with open(args.input_file) as f:
            input_req = f.read()
    else:
        input_req = sys.stdin.read()

    output = convert_req2json(input_req)
    if args.output_file:
        with open(args.output_file, 'w') as f:
            f.write(output)
    else:
        print(output)
```

```python
def create_patron(text: str, list_variables: list):
    patron = {}
    predicados = []
    #obtenemos todos los predicados menos los de tiempo
    predicados_todos = re.findall('\".*?\"', text)

    #obtenemos todos los predicados de tiempo
    tiempos = re.findall('least .+? time', text)
    tiempos = tiempos + re.findall('than .+? time', text)
    tiempos = tiempos + re.findall('every .+? time', text)
    tiempos = tiempos + re.findall('most .+? time', text)

    #se quita la primera palabra del tiempo
    count_t = 0
    for t in tiempos:
        if 'least' in t:
            tiempos[count_t] = t.replace('least', '')
        elif 'than' in t:
            tiempos[count_t] = t.replace('than', '')
        elif 'every' in t:
            tiempos[count_t] = t.replace('every', '')
        elif 'most' in t:
            tiempos[count_t] = t.replace('most', '')
        count_t += 1

    #se quita la segunda palabra del tiempo
    count_t = 0
    for t in tiempos:
        tiempos[count_t] = (t.replace('time', '')).strip()
        #mirar si el predicado ya se encuentra en la lista para no introducirlo 2 veces
        if tiempos[count_t] not in predicados_todos:
            #si en el patron hay un afterwards y además es el primero de tiempo
            #deberá meterlo en la posición 1
            if 'afterwards' in text and count_t == 0:
                predicados_todos.insert(1, tiempos[count_t])
            else:
                predicados_todos.append(tiempos[count_t])
        count_t += 1
```

```
#Tras obtener todos los predicados
#se procede a sustituir el texto por su letra correspondiente
count = 1
text = text.replace('(','')
text = text.replace(')','')
text = text.replace('+', '')
text = text.replace('|', '')
text = text.replace('*', '')

for pred in predicados_todos:
    pred = pred.replace('(','')
    pred = pred.replace(')','')
    pred = pred.replace('+' , '')
    pred = pred.replace('|', '')
    pred = pred.replace('*', '')

    if count == 1 :
        text = re.sub(pred, "{R}" , text , 1)
    elif count == 2 :
        text = re.sub(pred, "{S}" ,text,1)
    elif count == 3 :
        text = re.sub(pred, "{T}" ,text,1)
    elif count == 4 :
        text = re.sub(pred, "{U}" ,text,1)

    count += 1

#Tipo de patrón que es
text = text.lstrip()
tipos_predicados = type_patron(text)

#lista de predicados que irán dentro del
predicados = []

#introducir los predicados en la lista
count_pred = 0
for pred in predicados_todos:
    if count_pred == 0:
        text_pred = predicados_todos[0].replace('\"', '')
        variables_using_predicado = variables_using(list_variables, text_pred)
        predicados.append({"predicado": {"texto": text_pred, "indicador": "R",
            "tipo_predicado": tipos_predicados[0] , "variables": variables_using_predicado}})
    elif count_pred == 1:
        text_pred= predicados_todos[1].replace('\"', '')
        variables_using_predicado = variables_using(list_variables, text_pred)
        predicados.append({"predicado": {"texto": text_pred, "indicador": "S",
            "tipo_predicado": tipos_predicados[1] , "variables": variables_using_predicado}})
    elif count_pred == 2:
        text_pred = predicados_todos[2].replace('\"', '')
        variables_using_predicado = variables_using(list_variables, text_pred)
        predicados.append({"predicado": {"texto": text_pred, "indicador": "T",
            "tipo_predicado": tipos_predicados[2] , "variables": variables_using_predicado}})
    elif count_pred == 3:
        text_pred = predicados_todos[3].replace('\"', '')
        variables_using_predicado = variables_using(list_variables, text_pred)
        predicados.append({"predicado": {"texto": text_pred, "indicador": "U",
            "tipo_predicado": tipos_predicados[3] , "variables": variables_using_predicado}})
    count_pred += 1

patron = {"texto": text, "predicados": predicados}

return patron
```

```python
def create_ambito(text: str, list_variables: list):
    ambito = {}
    predicados = []
    predicados_todos = re.findall('\".*?\"', text)
    count = 1
    for pred in predicados_todos:
        if count == 1 :
            text = re.sub(pred, "{P}" , text , 1)
        elif count == 2 :
            text = re.sub(pred, "{Q}" , text , 1)
        count += 1

    predicados = []

    count_pred = 0
    for pred in predicados_todos:
        if count_pred == 0:
            text_pred = predicados_todos[0].replace('\"', '')
            variables_using_predicado = variables_using(list_variables, text_pred)
            predicados.append({"predicado": {"texto": text_pred,
                "indicador": "P", "variables": variables_using_predicado}})
        elif count_pred == 1:
            text_pred = predicados_todos[1].replace('\"', '')
            variables_using_predicado = variables_using(list_variables, text_pred)
            predicados.append({"predicado": {"texto": text_pred,
                "indicador": "Q", "variables": variables_using_predicado}})
        count_pred += 1


    ambito = {"texto": text, "predicados": predicados}

    return ambito
```

```python
def variables_using(list_variables: list, text: str):
    variables_return = []
    for var in list_variables:
        if len(re.findall('\\b'+var.get('nombre')+'\\b', text)) > 0:
            if var.get('nombre') not in variables_return:
                variables_return.append({'variable':{'nombre':var.get('nombre'),
                 'tipo': var.get('tipo_objeto'), "valor": var.get('tipo_valor')}})
    return variables_return

def type_patron(text: str):
    types_predicados = []
    if text == "it is always the case that if {R} holds, then {S} holds after at most {T} time units":
        types_predicados = ["accionador", "accionante", "tiempo"]
    elif text == "it is always the case that if {R} holds, then {S} holds as well":
        types_predicados =["accionador", "accionante"]
    elif text == "it is always the case that if {R} holds for at least {S} time units, then {T} holds afterwards":
        types_predicados =["accionador", "tiempo", "accionante"]
    elif text == "it is always the case that {R} holds":
        types_predicados =["accionante"]

    elif text == "it is never the case that {R} holds":
        types_predicados = ["accionante"]
    elif text == "it is always the case that if {R} holds, then {S} holds for at least {T} time units":
        types_predicados = ["accionador", "accionante", "tiempo"]
    ##Cuando se encuentre un patrón que no esté aquí se añadirá
    return types_predicados


if __name__ == "__main__":
    start = time.time()
    convert_req2json_main()
    end = time.time()
    print("tiempo total transcurrido:")
    print(str(end - start)+ ' segundos')
```

En caso de no poder leer el código, está disponible en el siguiente enlace:

https://github.com/Gorks/Files/blob/master/conversion_gorka_v6.py