

Modernización de Aplicaciones

Daniel Fernando Escobar Arenas

Diana Cárdenas Sánchez

Eddie Jairz Castro Vargas

Rolando Andrés Amarillo Pérez

21 de diciembre de 2015

ABSTRACT

Este documento es producto de un análisis referente a la modernización del software para mantener su valor de negocio, apoyándose en trabajos previos como “**Modernización Oracle Forms**”, “**A Tool to Assist Structural Understanding of Java JEE Applications Towards Modernization**”, y “**Transformación de aplicaciones JEE hacia una arquitectura de micro-servicios desplegada en nubes diferentes**” y centrados en la iniciativa de la OMG “**Architecture-Driven Modernization (ADM)**” se quiso proponer un modelo para generalizar y facilitar dicha modernización.

Teniendo como entrada principal aplicaciones JEE y usando herramientas como MoDisco, Epsilon y basándose en los meta-modelos Knowledge Discovery Metamodel (KDM), Abstract Syntax Tree Metamodel (ASTM), Software Metrics Metamodel (SMM) y Visualization Metamodel (VMM), se creó un proceso de análisis que pretende generar métricas para identificar el tamaño, la estructura y la complejidad de las aplicaciones, con el propósito de definir la viabilidad de la modernización del software objeto de análisis. Como resultado final se ofrece al usuario diagramas que sintetizan la arquitectura facilitando la toma de decisiones de acuerdo a sus necesidades y proponiendo una arquitectura orientada a micro servicios.

En este proceso se implementaron una serie de transformaciones modelo a modelo, que permitieron generar métricas representativas del software y su respectiva representación visual; a partir de lo anterior se pudo observar las ventajas que se obtienen en la automatización de procesos y el apoyo que da la propuesta de solución en la toma de decisiones para la reestructuración de la arquitectura.

ÍNDICE GENERAL

I PROBLEMA	6
1 INTRODUCCIÓN	7
2 TRABAJOS PREVIOS	8
2.0.1 A tool to assist Structural understanding of Java JEE applications towards modernization	8
2.0.2 Migrating JEE applications to a microservice-based architecture deployed on the cloud	9
2.0.3 Modernización Oracle Forms	9
2.0.4 The Design Maintenance System, A tool for automating software quality enhancemet	10
2.0.5 Moose	10
II SOLUCIÓN	11
3 PROPUESTA	12
3.1 Estrategia Global	13
3.2 KDM	14
3.2.1 El metamodelo KDM	14
3.2.2 KDM en la solución	15
3.2.3 Clustering por EJBs	15
3.3 Métricas	19
3.3.1 Definición	19
3.3.2 SMM	19
3.3.3 SDM	19
3.3.4 Métricas en SDM	20
3.4 Visualización	21
3.4.1 Definición	21
3.4.2 VMM	21
3.4.3 Sirius	21
3.4.4 Visualización en la solución	22
3.5 Implementación	23
3.5.1 MoDisco	23
3.5.2 Transformación KDM a Sub-Kdm	23
3.5.3 Transformación Sub-kdm a Métricas	27
3.5.4 Transformación Métricas a Representación	28
3.5.5 Transformación Visualización a Diagramación	31
3.5.6 Convenciones Métricas a Diagramación	33
III VALIDACIÓN	35
4 VALIDACIÓN Y RESULTADOS	36
4.0.7 Carrito de Compras	36
4.0.8 SisInfo	40

ÍNDICE GENERAL

5 CONCLUSIONES Y TRABAJO FUTURO	45
A OTROS ALGORITMOS DE CLUSTERING	47
A.1 Clustering por Árbol de Expansión Mínima (MST)	47
A.2 Cluster por Edge Betweenness	49
B LIMITACIONES TÉCNICAS	50
B.1 SonarQube	50
B.2 Análisis Aplicación Icfes	53
C INSTALACIÓN Y EJECUCIÓN	54
C.1 Repositorio	54
C.2 Clonación del Repositorio	54
C.3 Pre requisitos	55
C.4 Ejecución de la transformación en Eclipse	57
C.5 Visualización en Sirius	64

ACRONIMOS

MDE Model Driven Engineering

KDM Knowledge Discovery Metamodel

API Application Programming Interface

UML Unified Modeling Language

MST Minimum Spanning Tree (Árbol de Expansión Mínima)

EMF Eclipse Modeling Framework

JUNG Java Universal Network/Graph Framework

SMM Structured Metrics Metamodel

VMM Visualization Metamodel

SDM Simplified Decision Metrics Metamodel

Parte I
PROBLEMA

1

INTRODUCCIÓN

Actualmente, existe una gran variedad de software implementado bajo distintas tecnologías y usando infinidad de estilos arquitectónicos. Sin embargo a medida que transcurre el tiempo y las expectativas de las soluciones aumentan, se produce la denominada degradación del software que termina convirtiendo obsoletas estas soluciones. Para lo cual no queda otro camino que emprender iniciativas de modernización del mismo.

En la industria, es un problema constante enfrentarse a proyectos de modernización de soluciones existentes, no obstante en la ingeniería del software aún no se dispone de herramientas formales para afrontar los retos propuestos.

En el estado del arte se encuentran varias alternativas para afrontar eficientemente este tipo de problemáticas, pero se evidencian algunas limitaciones que no permiten escalar la solución a un abanico más amplio de contextos.

Retomando el camino recorrido y conforme a las nuevas herramientas disponibles que permiten implementar una propuesta robusta de acuerdo a las tendencias y estándares que se vienen implantando en el área de la ingeniería del software, se desea proponer una herramienta que facilite la comprensión de la estructura de software por medio de métricas y que la información se represente de manera que un arquitecto pueda fácilmente tomar decisiones con respecto a la propuesta de modernización.

2

TRABAJOS PREVIOS

En la actualidad, se realizan esfuerzos simultáneos por parte de la academia y de la industria con el fin de lograr mayor eficiencia en la modernización del software, lo cual se evidencia en trabajos adelantados en la Universidad de los Andes como:

A tool to assist Structural understanding of Java JEE applications towards modernization por Edgar David Sandoval Giraldo [San] y,

Migrating JEE applications to a microservice-based architecture deployed on the cloud por Mauricio Verano Merino [Mer].

El primero, se centra en el entendimiento y análisis estructural de aplicaciones JEE legado; y el segundo, en la rearquitecturación de las aplicaciones JEE monolíticas para llegar a arquitecturas basadas en micro servicios que puedan ser desplegadas en servicios de nube, como los ofrecidos por Amazon Web Services y Heroku.

2.0.1 *A tool to assist Structural understanding of Java JEE applications towards modernization*

En este trabajo, Edgar Sandoval afronta el problema del entendimiento del software JEE legado que se compone de muchos módulos y que carece de buena documentación, dificultando su comprensión. Para esto, se obtiene un modelo conforme al meta-modelo Java y posteriormente, se implementan cinco heurísticas que permiten obtener y completar un modelo conforme a un meta-modelo ad-hoc creado con el propósito de guardar la información relevante para el entendimiento de la aplicación. A éste modelo se le conoce como Modelo de Cartografía. (ver [San])

La principal problemática en el procesamiento mencionado, es la especialización del modelo conforme al meta-modelo de Java. Impidiendo realizar una extensión a su trabajo debido a que el modelo obtenido se encuentra estrechamente relacionado con el lenguaje. Por otro lado, existe un problema con el Modelo de Cartografía, en tanto sea un modelo ad-hoc ya que no es generalizable ni utilizable para otros propósitos.

Entre las heurísticas más relevantes para el contenido de este documento, se encuentra la heurística de agrupamiento o clustering. La

idea es obtener las relaciones entre los elementos que componen el software (proyectos y paquetes), analizando cómo se relacionan y visualizando los resultados, de manera que se pueda entender las dependencias entre los mismos. La heurística de agrupamiento depende de la heurística de relacionamiento, donde se analizan las relaciones entre clases e interfaces. La heurística de relacionamiento analiza los atributos, métodos y encabezados de la clase en búsqueda de referencias a otras clases, pero no alcanza a identificar las relaciones dentro del cuerpo de los métodos, lo que puede resultar en un entendimiento parcial de las relaciones entre clases.

Por último, la herramienta desarrollada por el autor, genera mecanismos de visualización y reorganización que le permite al arquitecto entender las relaciones entre los componentes del software analizado, dadas las mediciones obtenidas para las relaciones y el tamaño de los componentes.

2.0.2 Migrating JEE applications to a microservice-based architecture deployed on the cloud

El trabajo realizado por Mauricio Verano (ver [Mer]) está orientado a realizar la migración de aplicaciones JEE monolíticas a aplicaciones orientadas a micro-servicios. Para lo cual continua el trabajo propuesto por Edgar Sandoval, reestructurando el proyecto y adicionando anotaciones sobre los beans de sesión encargados de ser la puerta de entrada de servicios RESTful.

Para ello declara e implementa el procesamiento de anotaciones a nivel de clase, encargados de definir el recurso para el micro-servicio y anotaciones a nivel de método que definen los puntos de entrada del micro-servicio. De este procesamiento, se obtiene el código reorganizado para exponer los micro-servicios y preparado para compilar correctamente, después se encarga del aprovisionamiento en proveedores de nube y el despliegue del mismo.

2.0.3 Modernización Oracle Forms

Un trabajo cuya investigación se centra en la modernización del software, nace como propuesta a los inconvenientes que enfrentaba Asesoftware, una empresa colombiana de software con sus aplicaciones de Oracle Forms (ver [Gar+]).

Esta empresa tenía dificultades con el soporte de estas aplicaciones, ya que actualmente se encuentra poca mano de obra y son aplicaciones muy complejas, por otra parte entender la funcionalidad era un

proceso que requería mucho tiempo debido a dicha complejidad y dificultaba aún más la actualización de las mismas.

Por tal razón la solución que se propone ofrece diagramas y editores para facilitar el entendimiento y la configuración de la aplicación, un generador de código fuente de la aplicación modernizada y un generador de pruebas unitarias para la capa de persistencia, mejorando la calidad de la aplicación y agilizando el proceso de migración.

En la solución presentada se usa el algoritmo de EdgeBetweenness para el análisis de comunidades en grafos, con el propósito de presentar una división del software según las relaciones de los Forms con los elementos de datos.

2.0.4 *The Design Maintenance System, A tool for automating software quality enhancemet*

DMS (Design Maintenance System) es una herramienta de análisis y conocimiento de software orientada al entendimiento y mantenimiento del software legado. Como entradas, la herramienta requiere el código fuente del software y el conocimiento de ingeniería asociado al lenguaje y al negocio (ver [Bax]).

DMS no es una solución orientada a MDE y la problemática asociada se encuentra en que la herramienta de análisis se ve fuertemente acoplada al lenguaje de programación.

2.0.5 *Moose*

Moose es una plataforma de código abierto para el análisis de datos; define un meta-modelo llamado FAMIX similar a KDM que permite modelar componentes de software y su sistema operacional en instancias definidas en MSE (similar a XMI).

El problema de los trabajos expuestos se concentra en la reorganización del código, algunas de las preguntas esenciales que surgen en la migración de la arquitectura son: ¿Qué tamaño debe tener cada micro-servicio?, ¿Qué clases componen el micro-servicio?, ¿Es posible que los servicios identificados puedan ser divididos en sub-unidades más atómicas?. Es probable que el entendimiento inicial de la aplicación no sea lo suficientemente profundo y que cada módulo identificado pueda ser dividido en responsabilidades casi disyuntas, dependiendo de las relaciones de las clases, los entities y los beans de sesión.

Parte II
SOLUCIÓN

3

PROPIEDADES

La propuesta expuesta en este documento intenta resolver algunos de los problemas que se presentan en los trabajos anteriormente descritos (ver 2). En primer lugar se quiere evitar en lo posible el uso de meta-modelos ad-hoc, usando y extendiendo meta-modelos reconocidos por la comunidad de desarrollo de software.

En segundo lugar, se plantea extender el entendimiento del software con el uso de algoritmos que permitan la identificación de unidades funcionales de software y la visualización de los elementos del mismo con vistas fácilmente entendibles, que permitan al arquitecto agilizar y optimizar el proceso de modernización y en especial la migración a arquitecturas orientadas a micro-servicios.

Adicionalmente, se busca identificar a través de métricas la complejidad del software actual y la estructura del código interno.

3.1 ESTRATEGIA GLOBAL

La estrategia global se centra en el uso de MDE y en especial de las herramientas y lenguajes provistos por Epsilon (ver [Foua]) para el análisis de las aplicaciones JEE.

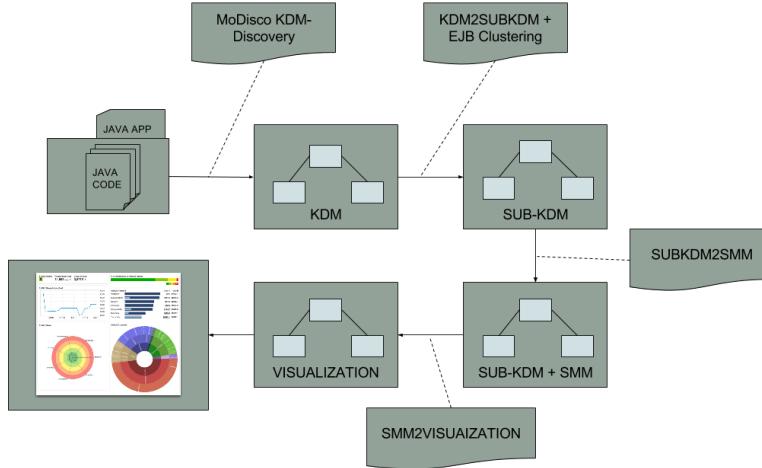


Figura 1.: Estrategia Global

Se parte de un modelo KDM de la aplicación a analizar, para luego realizar un refinamiento a la versión simplificada del modelo (sub-kdm), y posteriormente obtener una división por responsabilidades del código como primera aproximación (ver sección 3.2).

Posteriormente, se realiza un análisis basado en métricas sobre el modelo conforme a sub-kdm, el cuál agrega al modelo obtenido la terminología y estructura simplificada de SMM. Éste meta-modelo simplificado se define como Simplified Decision Metrics (SDM) (ver sección 3.3).

Como último paso, se sintetizan los conceptos de las métricas y el subconjunto de KDM para usar los conceptos de VMM en un modelo que pueda ser representado gráficamente, usando Sirius. Una vez terminado el procesamiento se espera que un arquitecto de software, encargado de tareas de modernización, pueda tener un mayor entendimiento de la aplicación y proponer una arquitectura de micro-servicios (ver sección 3.4).

3.2 KDM

3.2.1 *El metamodelo KDM*

Knowledge Discovery Metamodel (KDM) o Meta-modelo de Descubrimiento de Conocimiento, es un esfuerzo del Object Management Group (OMG) por representar el software existente y su sistema operacional, de tal forma que sea independiente del lenguaje y su ambiente de ejecución.

KDM se divide en 12 paquetes que contemplan los conceptos de software, estos paquetes se agrupan en 4 capas. (ver [KDM]):

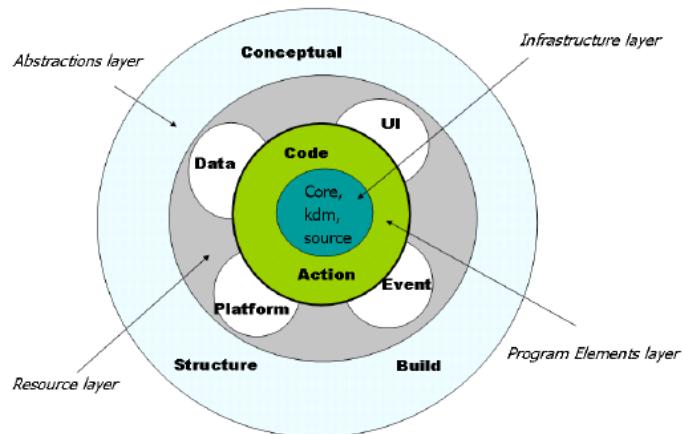


Figura 2.: Capas KDM

- Capa de abstracción, conformada por los paquetes: Conceptual, Estructura y Construcción.
- Capa de recursos, conformada por los paquetes: Datos, Plataforma, Eventos e Interfaz de Usuario.
- Capa de Elementos de Programación, conformada por los paquetes: Código y Acción.
- Capa de infraestructura, conformada por los paquetes: Núcleo, kdm y fuente

Este documento se centra en la capa de elementos de programación. La principal razón para usar KDM como parte de la solución es su amplia aceptación como estándar, de manera que es reconocido por la comunidad de desarrollo de software, lo que permite plasmar los conceptos del software en un modelo que no depende del lenguaje de programación, e identifica los elementos del código de forma atómica y las relaciones entre pares de elementos, posibilitando el entendimiento del software.

3.2.2 KDM en la solución

Dada la cadena de pasos introducida en la figura 1, se describen los pasos 1 y 2 que detallan el descubrimiento del modelo de conocimiento para el software que se desea analizar.

En el primer paso, a través de una herramienta de descubrimiento, se realiza el levantamiento inicial de la información del software, obteniendo un modelo conforme a KDM. La implementación de esta herramienta está fuera del alcance del proyecto y para el software basado en el lenguaje JAVA se usará la herramienta MoDisco (ver [Foub]) que se describe brevemente en la sección 3.5.

Una herramienta que haga el levantamiento inicial del modelo conforme a KDM (como MoDisco), debe obtener un modelo acorde a los nueve paquetes comprendidos por las capas de recursos, elementos de programación e infraestructura. Para el análisis se necesitan los paquetes de Código y Acción que comprenden todos los elementos del programa y sus relaciones, por lo cual se requiere un segundo paso centrado en la extracción de un subconjunto de esta información.

KDM permite obtener, entre otros elementos, la siguiente información:

- Unidades de código: ClassUnit, TemplateUnit, InterfaceUnit y MethodUnit.
- relaciones: StorableUnit, ActionElement(Method Invocation y variable declaration), Parameter Unit, Implements, Extends.

Analizando estos elementos se pueden obtener las relaciones entre las clases, incluso con llamados atómicos de los métodos de una clase a propiedades y servicios de otras.

3.2.3 Clustering por EJBs

KDM permite encontrar todas las relaciones que categorizamos a continuación:

Se define que una Clase A está relacionada con una clase B si:

- A tiene un atributo de tipo B
- A llama un método de B
- A tiene un método que referencia a B como parámetro o como retorno
- A extiende o implementa a B

- Existe una clase A', tal que A extiende o implementa A', que está relacionada con B en alguna de las formas anteriores.

Los primeros cuatro puntos son las relaciones normales que pueden extraerse de un modelo UML. El quinto punto es una propiedad de la extensión en objetos.

En KDM es posible encontrar todos los nodos usando los conceptos de ClassUnit, InterfaceUnit y TemplateUnit como clases y las relaciones entre ellos en una navegación por sus elementos como StorableUnit (se refiere a variables locales y globales, ie. atributos), MethodUnit (se refiere a funciones y métodos), ActionElement (se refiere a acciones de código, por ejemplo llamados, instanciación, escritura, lectura, entre otros), Implements y Extends.

Una vez encontradas estas relaciones, se hace un análisis de código que permita a la herramienta obtener clusters separados por responsabilidades, definiendo las responsabilidades del software como los servicios ofrecidos por un Bean de Sesión (EJBs), y un cluster como una unidad del software que podría ser completamente funcional por si sola y provee algunos de los servicios que el software ofrece.

El objetivo es identificar todos los componentes del software (clases, interfaces, etc.) que tienen relación con el EJB y que conectan los servicios con la capa de datos, que para este caso lo representamos con los Beans de Entidad (Entity Beans). El primer paso consiste en analizar la estructura como un grafo dirigido, después se identifica el árbol de herencia para cada EJB y cada Bean de Entidad y por último se revisan los caminos que llevan de cada elemento del árbol de herencia de un EJB a los elementos de cada árbol de herencia de un Bean de entidad.

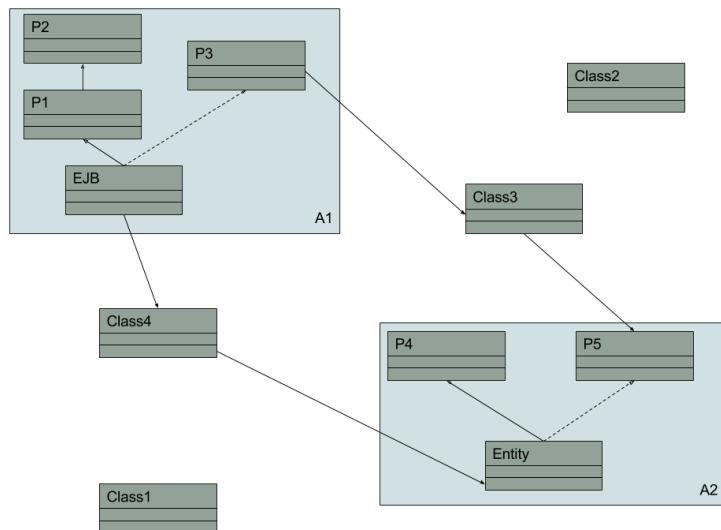


Figura 3.: Algoritmo de Clustering EJB

La figura 3 ilustra el funcionamiento del algoritmo, los conjuntos A1 y A2 representan los arboles de herencia para EJB y Entity. Para cada elemento de A1 se buscan los caminos que los conectan con los elementos de A2, en este caso se encuentran dos caminos que pasan por Class3 y Class4. Todos los elementos que se identifiquen como conectados de esta forma son incluidos en el cluster identificado para EJB.

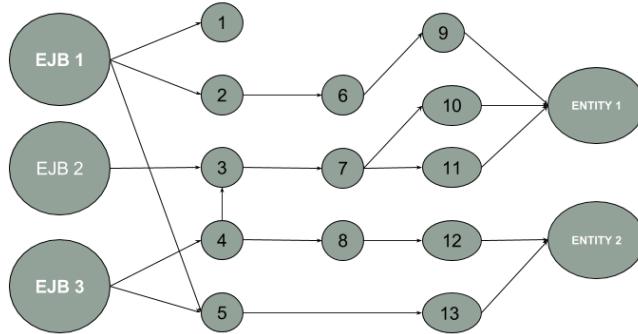


Figura 4.: Identificación de Caminos EJBs a Entities

Suponiendo ahora que se tienen los elementos del software representados como un grafo, ver la figura 4, donde cada círculo representa una clase, se identifican los caminos que conectan a cada EJB con cada Entity. También se hace necesario identificar caminos entre el EJB y clases utilitarias como se ilustra en la misma figura con *EJB1* y la clase 1. En la figura 5 se muestra la separación por responsabilidades, donde para cada conjunto se encuentran todas las clases que intervienen en los caminos ya identificados.

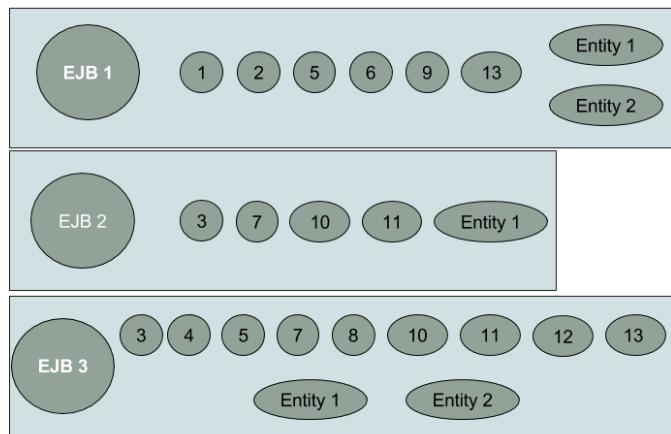


Figura 5.: Clusters EJB

Por último, se puede representar la información obtenida nuevamente como un grafo, donde los nodos representan cada conjunto

identificado y los arcos representan la intersección de sus elementos como se hace en la figura 6. La intersección se representa como una medida de cuantas clases son compartidas, por los EJBs relacionados, para poder prestar sus servicios. Esta representación le permite al usuario identificar formas de agrupar las clases con el objetivo de definir micro-servicios, en este caso es posible definir el agrupamiento por un nivel de intersección superior al 80 % y obtener una división como se ilustra en la figura 7.

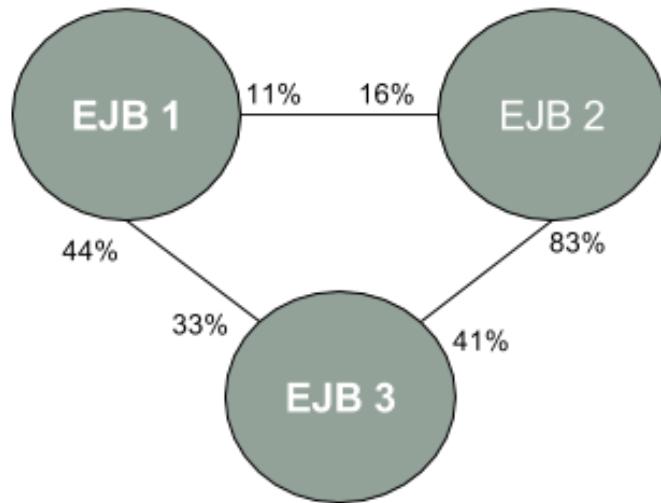


Figura 6.: Representación por intersección

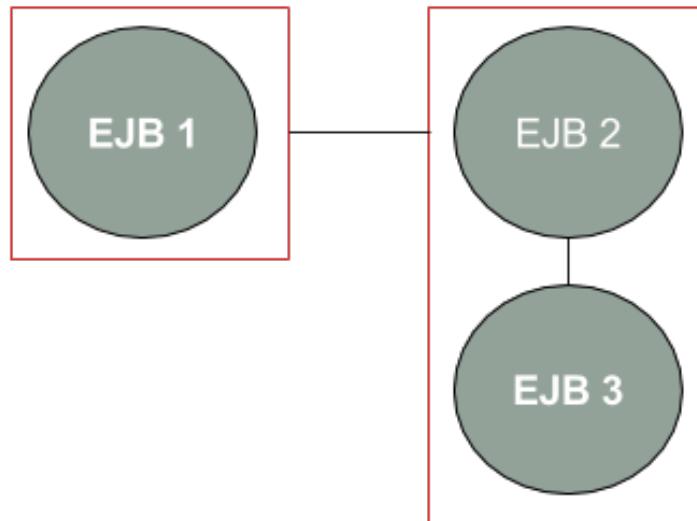


Figura 7.: Separación de clusters por porcentaje

3.3 MÉTRICAS

3.3.1 Definición

Métricas se refiere a todo lo relacionado a sistemas de medidas, en el caso del desarrollo de software, se relaciona a las medidas que se utilizan para valorar cuantitativamente la calidad del software. Existen sistemas de medida comunes para características que comparte el desarrollo del software, como también existen métricas especializadas en diferentes características únicas según la tecnología, ambiente, lenguaje, etc.

3.3.2 SMM

Structured Metrics Meta-model (SMM) es una especificación del Object Management Group (OMG) para definir un meta-modelo que represente datos de medidas relacionadas con cualquier modelo de información estructurada focalizada inicialmente al diseño, desarrollo y operación del software. [OMG]

Ésta especificación se concentra en la definición de medidas y la representación de los resultados de medición. Aunque incluye una librería mínima de medidas de software, SMM no afirma que sean normativas. SMM se usa para extraer métricas de elementos MOF (Meta-Object Facility) siendo artefactos de software representados usando Knowledge Discovery Metamodel (KDM).

3.3.3 SDM

Teniendo en cuenta que SMM es utilizado para extraer métricas a un nivel abstracto de objetos KDM, y su necesidad de extracción para poder completar satisfactoriamente el propósito del proyecto, se decidió tomar elementos simplificados de la especificación SMM para realizar la extracción y definir posteriormente métricas específicas para la solución del problema. A ésta especificación, se le denominó Simplified Decision Metrics (SDM).

SDM utiliza cuatro conceptos clave de SMM para definir métricas:

- Medida (Measure) - Método de asignación numérica o simbólica que caracterizan un atributo de las entidades.
- Medición (Measurement) - Valor numérico o simbólico de una medida asignado a una entidad por medida.
- Mensurado (Measurand) - Entidad cuantificada por medición.
- Observación (Observation) - Mediciones aplicadas a un conjunto de mensurados.

3.3.4 Métricas en SDM

SDM está centrado en la medición de acoplamiento entre clases aunque incluye métricas generales como líneas de código y número de clases. Para realizar la medición del acoplamiento se utilizaron dos de las seis métricas del suite de métricas orientada a objetos de Chidamber & Kemerer (ver [CK]):

- WMC (Weighted Methods per Class) - Número de métodos definidos en una clase. Cuando la complejidad es variable entre métodos, WMC es la suma de la complejidad de los métodos que conforman una clase. [Sán]
- CBO (Coupling between Object Classes) - Número de clases a la cual una clase está acoplada. Se cuentan las llamadas de los métodos y variables referenciadas desde una clase a otra. Múltiples accesos a una misma clase se toman como un acceso. [Sán]

Estas métricas se eligieron ya que están centradas en las principales características para la reestructuración y mantenimiento del código: el acoplamiento y la cohesión.

Junto a estas métricas, se definieron métricas específicas para dar más información sobre el acoplamiento del sistema:

- NOU (Number Of Uses) - Número de relaciones de uso que tiene una clase. Una clase es usada o usa otra clase. Cada uso cuenta como una relación
- NOCon (Number of Containments) - Número de relaciones de contención que tiene la clase. Una clase puede estar contenida en una clase o contiene otras clases. Cada contención cuenta como una relación.
- CCM (CalculatedCouplingMeasure) - Sumatoria de las dos métricas anteriores multiplicadas cada una por un factor dado por el usuario.
- CRCP (Cluster Relation Conflict Percentage) - Porcentaje de conflicto entre relaciones de cluster. Se toman el número de clases de la relación en conflicto sobre el número de clases que tiene el cluster.

La definición de los valores para la métrica CCM se dejó a cargo del usuario debido al inconveniente de comparar y dar un valor cuantitativo a la comparación entre relaciones de clases, lo que se considera probablemente abstracto para cada tecnología, ambiente, lenguaje, proyecto, desarrollador, etc.

3.4 VISUALIZACIÓN

3.4.1 *Definición*

Una definición amplia de "visualización", aplicable al contexto de la temática del proyecto, puede ser: "Representación gráfica y visual (imágenes) de datos complejos en su naturaleza con el objetivo de dar a entender el significado de los mismos de forma clara y resumida".

Para lograr tal propósito, en primera instancia se hace necesario definir el tipo de información a visualizar así como el tipo de audiencia que va a interpretar la información.

La información a representar es: en primer lugar, las métricas que permiten evidenciar la calidad del software así como para poder identificar los puntos críticos que repercuten negativamente en las mediciones; y en segundo lugar el resultado de los algoritmos de identificación de clusters aplicados a los modelos conformes con sub-kdm.

La representación se basa en UML ya que es comúnmente conocido en el ámbito del diseño y arquitectura. A continuación se describen las iniciativas que guiaron la propuesta de visualización contenida en el presente documento.

3.4.2 *VMM*

Visualization Metamodel (VMM) es una especificación del Object Management Group (OMG) para describir construcciones tridimensionales que permitirán visualizar los sistemas de software. El objetivo de ésta es reproducir diversos tipos de escenarios visuales para representar la estructura, dinámicas y métricas del software. VMM es una iniciativa actualmente obsoleta, aunque los conceptos allí tratados fueron retomados para la propuesta de visualización presentada. La última versión fue lanzada en Febrero de 2010.

Se tomaron varios elementos constitutivos de la plataforma Moose (ver [Tec]) para la definición y construcción del metamodelo simplificado de visualización propuesto.

3.4.3 *Sirius*

Framework del proyecto Eclipse el cual proporciona un esquema genérico de visualización basado en MDE. Permite crear un esquema de modelado gráfico mediante el aprovechamiento de las tecnologías Eclipse Modeling.

Dentro de las restricciones definidas en la propuesta de visualización, Sirius permite representar adecuadamente las métricas y soluciones alternativas de agrupamiento, donde la herramienta se encarga de la generación de los gráficos.

3.4.4 *Visualización en la solución*

Como punto de entrada se tiene un modelo conforme al metamodelo subkdm con las métricas calculadas, a partir del cual se realizan dos transformaciones ETL, la primera orientada a representar los elementos de software como nodos y dependencias y la segunda a obtener una representación gráfica de los elementos. Como resultado se obtiene un modelo conforme al metamodelo de visualización (ver 3.5.4).

El modelo resultante se procesa en la herramienta Sirius, la cual previamente tiene configurado los elementos gráficos con respecto al modelo de visualización, lo que permite desacoplar la lógica de visualización.

Los diagramas diseñados en Sirius para representar la solución son:

- Diagrama de clusters: Diagrama principal, donde se representa visualmente la separación por clusters. La organización de los elementos del diagrama (clusters) se realiza de forma radial. Este tipo de reorganización se reutilizó del algoritmo implementado en el proyecto de Edgar Sandoval (ver [San]).
- Diagrama de clusters simplificado: Diagrama que agrupa los clusters según el porcentaje de intersección entre ellos. La organización de los elementos del diagrama (grupo de clusters) se realiza de forma radial. Este tipo de reorganización se reutilizó del algoritmo implementado en el proyecto de Edgar Sandoval (ver [San]).
- Diagrama de contenedores: Diagrama genérico que permite presentar paquetes, o clases con sus relaciones, así como el contenido interno de las mismas (clases, atributos, métodos).

En Sirius se puede navegar entre los elementos visuales de los diagramas generados, lo que permite generar nuevos diagramas. De esta manera se puede generar nueva información que permita mejorar la toma de decisiones con respecto a la modernización del software.

En el diagrama de clusters, tiene adicionalmente la opción de navegar entre sus relaciones, lo que genera un nuevo diagrama para visualizar los componentes que comparten los clusters relacionados.

3.5 IMPLEMENTACIÓN

Para la solución propuesta se genera un modelo conforme a KDM con ayuda de MoDisco y a partir de éste se implementan 4 transformaciones modelo a modelo con el fin de obtener una representación gráfica de la estructura del software y de las métricas obtenidas del código.

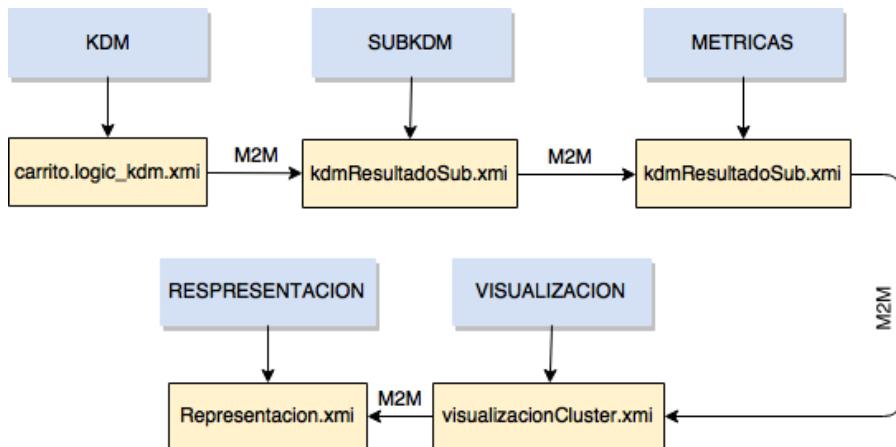


Figura 8.: Cadena de transformación

3.5.1 *MoDisco*

MoDisco ofrece una implementación del meta-modelo Knowledge Discovery Metamodel (KDM) cuyo objetivo es representar artefactos de software como entidades y relaciones, por lo cual fue utilizada para obtener un modelo conforme a KDM del software a analizar. Este modelo es la entrada a la primera transformación que se implementa en el proceso.

MoDisco hace una lectura de los archivos del proyecto y extrae un modelo conforme al meta-modelo de Java. Posteriormente procesa ese modelo y obtiene un modelo conforme al meta-modelo de KDM.

3.5.2 *Transformación KDM a Sub-Kdm*

A partir del modelo KDM obtenido con MoDisco se realiza una transformación ETL modelo a modelo para obtener un subconjunto de éste con los principales elementos para la representación del código identificando paquetes, clases, y relaciones (ver figura 9), el resultado obtenido es un modelo conforme a Sub-Kdm.

El meta-modelo Sub-Kdm contiene tres paquetes, el primero de ellos orientado a los elementos del código (kdmObjects) y contiene

únicamente los elementos que se analizarán en este proyecto, el segundo contiene los elementos que modelan las relaciones entre los elementos de código (kdmRelations) y por último un paquete con los elementos que modelan las métricas del software (SimplifiedDecisionMetrics)

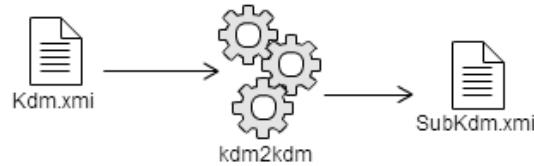


Figura 9.: Transformación KDM 2 Sub-kdm

Al metamodelo se agregó un elemento “*Cluster*” al paquete kdmObjects con el propósito de modelar los clusters como conjunto de objetos de código, de igual manera se agrega un elemento de tipo relación llamado ClusterRelation en el respectivo paquete de kdmRelations, con el propósito de modelar la relación entre los clusters encontrados (ver figura 10).

Así mismo se adicionó una relación “*ClassLevelRelation*”, con el propósito de agrupar todas las relaciones encontradas entre dos clases o interfaces (ver figura 11). De acuerdo a las relaciones encontradas entre las clases, una relación ClassLevelRelation puede contener un conjunto de elementos de tipo TypeRelation, creados para resumir que tipos de relaciones existen (Uso, Implementación, Extensión o Contenencia).

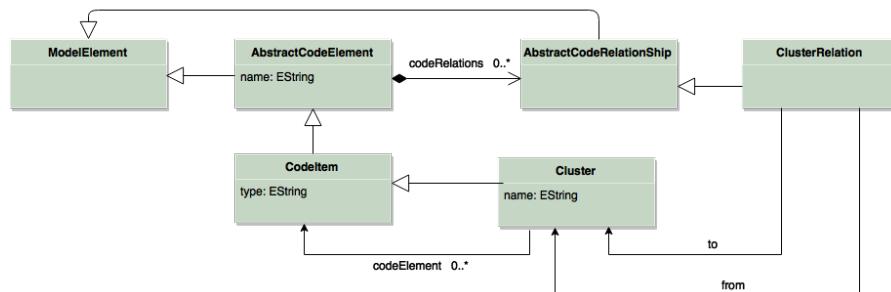


Figura 10.: Diagrama de clases Cluster y ClusterRelation

Adicionalmente en esta transformación se realiza un proceso de clustering para identificar subconjuntos de acuerdo a los EJB de la aplicación. El proceso de clustering es realizado por un plugin que se creó para tal propósito. El plugin cuenta con los algoritmos de

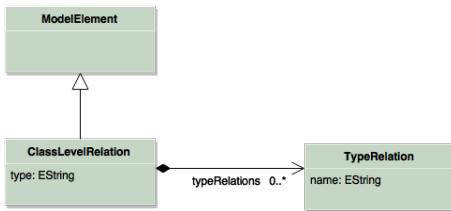


Figura 11.: Diagrama de clases ClassLevelRelation

clustering por EJBs y clustering por medida de acoplamiento. Las transformaciones ETL y en general todas las herramientas de EMF y Epsilon permiten extender su comportamiento con plugins escritos en JAVA y llamarlos como en este caso desde transformaciones modelo a modelo.

El plugin contiene los siguientes paquetes:

- clusterer
 - ClusterCalculation: Un experimento para crear clusters con el algoritmo de EdgeBetweennesClusterer (ver [OFW]).
 - MSTClusterCalculation: Calculo de clusters usando Árboles/Bosques de Expansión mínima. (Ver A.1)
 - EJBClusterer: Cálculo de clusters basado en EJBs. (Ver 3.2.3)
 - graph
 - WeightedGraph: Una implementación genérica de un grafo no dirigido con peso y una implementación del algoritmo de Prim que obtiene el Árbol/Bosque de Expansión Mínima. (Ver [SW11])
 - test
 - WightedGraphTest: Una prueba básica del funcionamiento del algoritmo de Prim y del grafo no dirigido con peso.

3.5.2.1 ClusterCalculation

Este algoritmo usa la estructura de grafos no dirigidos y el algoritmo de EdgeBetweennessCluster de las librerías JUNG (ver [OFW]). Es una implementación similar a la que se hace en [Gar+], el algoritmo tiene algunas propiedades que pueden dificultar su uso en el análisis de software.

Primero, el algoritmo está orientado a separar comunidades densamente relacionadas y que tienen relaciones no muy densas con otras comunidades. En el caso de las clases y relaciones entre éstas, la medida total del acoplamiento de todas las clases entre sí no debe ser muy alta pues no se podrían reconocer realmente comunidades.

Segundo, el algoritmo se basa en eliminar los arcos que más intermedian en los caminos cortos entre cada par de nodos para el grafo. Esto quiere decir que en cada iteración se elimina un único arco, la implementación del algoritmo recibe como parámetro el número de relaciones que deben ser eliminados y se hace difícil estimar el número correcto de iteraciones que se deben realizar para obtener clusters correctamente separados.

Como resultado de la problemática anteriormente descrita, si se tiene un grafo densamente conectado y no se estima correctamente el número de iteraciones a realizar, es posible que el cluster obtenido sea el conjunto total de nodos. Si se tiene un grafo que no está densamente conectado y no se estima bien el número de iteraciones puede terminar con muchos clusters de muy pocos elementos.

En términos de software, un grafo densamente conectado, en nuestro contexto, implica un acoplamiento muy alto y puede que el resultado del algoritmo sea el correcto, es decir no es posible separar el software en micro servicios; pero al no saber el número correcto de iteraciones a ejecutar tampoco se puede reconocer como un buen resultado. En el caso de un grafo que no está densamente conectado, la división puede sugerir clusters de una clase y obtener separaciones en relaciones que no deben separarse. Algunos detalles de este algoritmo se encuentran en A.2.

3.5.2.2 *EJBClusterer*

El algoritmo de clustering por EJBs se implementó con la ayuda de las librerías de grafos y algoritmos creada por JUNG (Java Universal Network/Graph Framework [OFW]) para el estudio de los grafos. El constructor de esta clase no recibe ningún parámetro y es llamado por la transformación ETL "kdm2kdm", posteriormente se llama el método "constructGraph" que recibe como parámetros todos los elementos de tipo ClassUnit, InterfaceUnit y TemplateUnit como un conjunto y todos los elementos de tipo ClassLevelRelation, creando un grafo dirigido. Por último se llama el método makeCluster que recibe como parámetros todos los elementos identificados como EJBs (anotados con @Stateless o @Stateful) en un conjunto y todos los elementos identificados como entities (anotados con @Entity) en otro conjunto.

Para cada EJB y cada Entity se extrae recursivamente el árbol de herencia, es decir todas las clases relacionadas a cada uno por extensión o implementación. Usando el algoritmo de Dijkstra es posible encontrar los caminos cortos que van desde un elemento del árbol de herencia del EJB a cada elemento del árbol de herencia de cada Entity como se ilustra en las figuras 3 y 4. Para cada EJB se crea un cluster como en la figura 5, el algoritmo de Dijkstra (ver [SW11]) permite

encontrar los caminos cortos de un nodo a otro para grafos dirigidos, todos los nodos del grafo encontrados en el camino serán agregados al cluster. Los detalles precisos del procedimiento que se realiza para obtener la separación se encuentran en 3.2.3.

3.5.2.3 *MSTClusterCalculation*

El algoritmo de clustering por Árbol de Mínima Expansión usa la estructura de grafo no dirigido con peso (WeightedGraph) y su implementación del algoritmo de Prim. La transformación se encarga de instanciar la clase por medio del constructor sin parámetros y después debe llamar el método constructGraph que recibe como parámetros los nodos (de tipo ClassUnit, TemplateUnit e InterfaceUnit). Después debe llamar el método addClassLevelRelation que recibe un elemento ClassLevelRelation y su peso (1/CCM donde CCM es el valor del acoplamiento entre las clases relacionadas por ClassLevelRelation) por cada relación encontrada para el software. Por último debe llamar el método getClusters que recibe como parámetro un entero mayor o igual a cero que va a significar el número de relaciones que desea eliminar de cada elemento del bosque de mínima expansión.

Los detalles de como funciona el algoritmo se encuentran en A.1, cabe decir que este algoritmo solo puede ser ejecutado después de realizar el análisis de métricas que se detalla en 3.5.3.

3.5.3 *Transformación Sub-kdm a Métricas*

El modelo Sub-kdm se extiende con la transformación modelo a modelo de métricas, por cada clase se analizan métricas como líneas de código, cohesión y acoplamiento teniendo en cuenta las relaciones entre clases y un peso dado según el usuario.

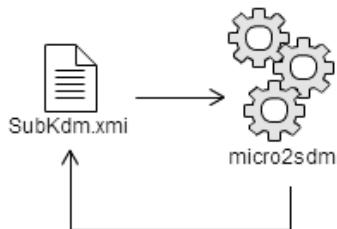


Figura 12.: Transformación Sub-kdm a Métricas

En esta transformación se definen actualmente métricas por clases, relaciones entre clases, clusters y relaciones entre clusters, las cuales ayudarán a identificar que tan acoplada es una aplicación y el nivel

de calidad de codificación que maneja. El modelo resultado es conforme al metamodelo SDM (ver figura 13). Como se pude observar, se puede definir una métrica por cada elemento del modelo sub-kdm, el cual es identificado como un MOFElement en SDM, así mismo se puede extender el metamodelo con nuevas métricas que aporten en el análisis de la calidad del software.

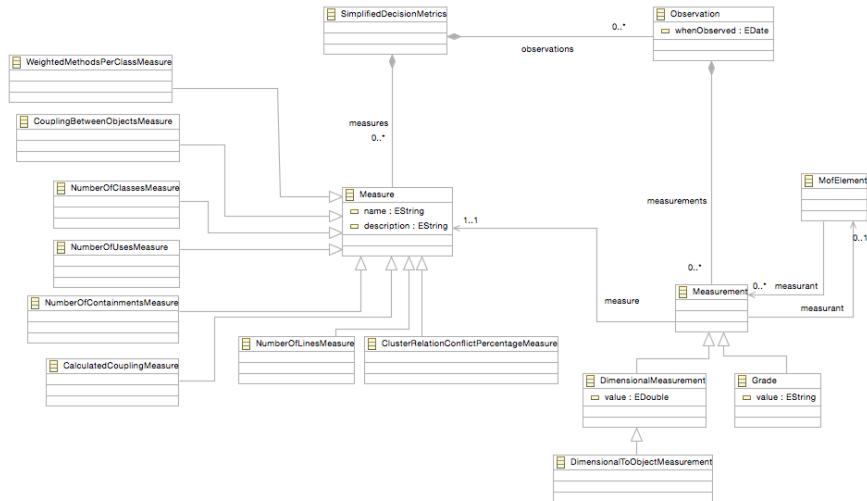


Figura 13.: Metamodelo SDM

La métrica que definirá la agrupación en micro servicios es CRCP (Cluster Relation Conflict Percentage) (ver 3.3.4), partiendo del valor calculado posteriormente en la visualización se podrá obtener un cluster similar a los mostrados en las figuras 6 y 7

3.5.4 Transformación Métricas a Representación

A partir del modelo conforme a sub-kdm, se realiza una transformación ETL modelo a modelo para sintetizar las métricas y los cluster, obteniendo así por cada cluster las clases que contiene y las métricas necesarias mapeadas a los conceptos definidos para la visualización final.



Figura 14.: Transformación Métricas a Representación

El diseño del meta-modelo está pensado para representar grafos (nodos y arcos), además de implementar el patrón composite. A continuación se explica los elementos más significativos del meta-modelo:

- Arco: Representa las relaciones entre elementos de software.
- Elemento: Componente base que representa elementos de software, es un elemento abstracto.
- Nodo: Subtipo de «Elemento», sirve como contenedor de otros elementos de software.
- Hoja: Subtipo de «Elemento», elementos de software atómicos.
- Vínculo: Subtipo de «Elemento», Mecanismo que sirve como enlaces (links) a subtipos de elementos de software existentes en el modelo y con el cual se puede generar distintas agrupaciones (diagramas) reutilizando instancias del modelo.
- Medible: Interface que permite dotar de métricas a los componentes del modelo que la implementan, en este caso para «Arco» y «Elemento».
- Metrica: Elemento que contiene las medidas obtenidas en el proceso de análisis de métricas, aplica para cualquier elemento «Medible»
- MetricaRelacion: Subtipo de «Metrica», la cual permite relacionar una métrica con más de un elemento, particularmente usada para métricas asociadas a arcos y se requiera relacionar a su source o target

El meta-modelo usado para esta transformación se compone de nodos que contienen elementos y arcos que representan las dependencias entre ellos, tanto nodos como arcos pueden ser medibles, es decir pueden contener métricas con valores numéricos que se usarán en la representación gráfica del software (ver figura 15).

Adicionalmente, se creó una extensión del meta-modelo base para poder implementar las distintas agrupaciones de elementos de software, y de esta manera poder desacoplar las representaciones que pueden darse y hacer la solución más extensible. A continuación se explican los elementos más significativos del meta-modelo:

- Las representaciones de software que fueron usadas para «Nodo» son:
 - Clase
 - Paquete

3.5 IMPLEMENTACIÓN

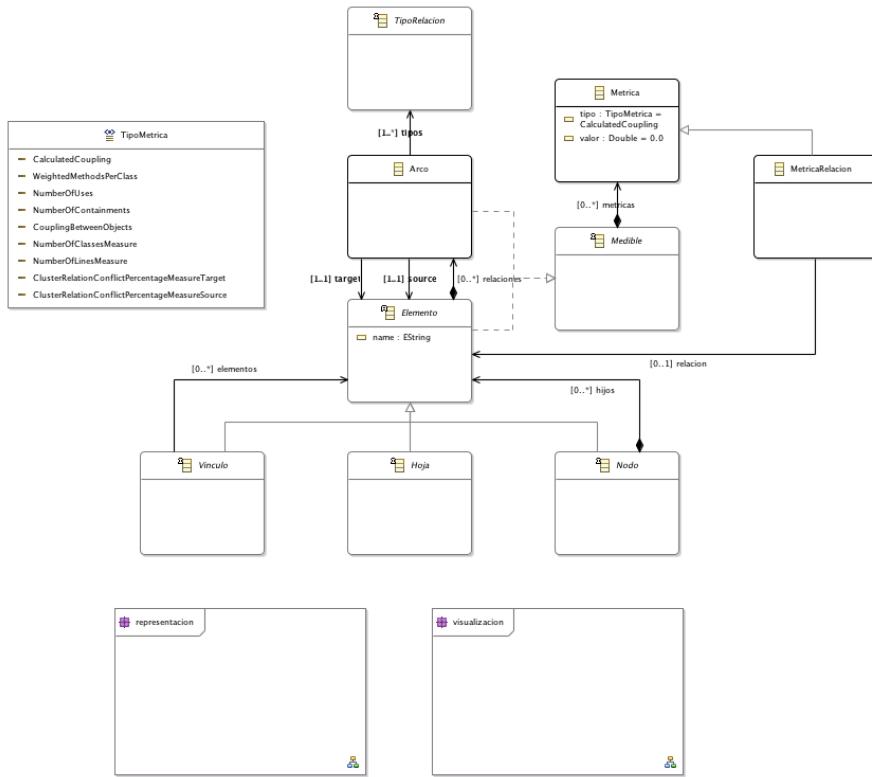


Figura 15.: Metamodelo Visualización

- Las representaciones de software que fueron usadas para «Hoja» son:
 - Atributo
 - Método
- Las representaciones usadas para «Vínculo» son:
 - Cluster: Agrupación lógica de elementos software autocontenido, en este componente se representa el resultado de la aplicación del algoritmo de Clustering sobre el modelo subkdm
 - Intersección: Elemento que agrupa los elementos de software en común entre clusters, está asociado al «Arco» que relaciona dos clusters

En esta transformación se recorren los cluster del modelo conforme a sub-kdm, los mapea al respectivo elemento de cluster del metamodelo de visualización y para cada uno hace la búsqueda de las relaciones asociadas a éste. De igual forma asocia sus respectivos paquetes, clases y dependencias con otros clusters.

Entre las métricas que se asocian a un cluster se pueden identificar, número de clases, número de líneas de código y la medida en porcen-

3.5 IMPLEMENTACIÓN

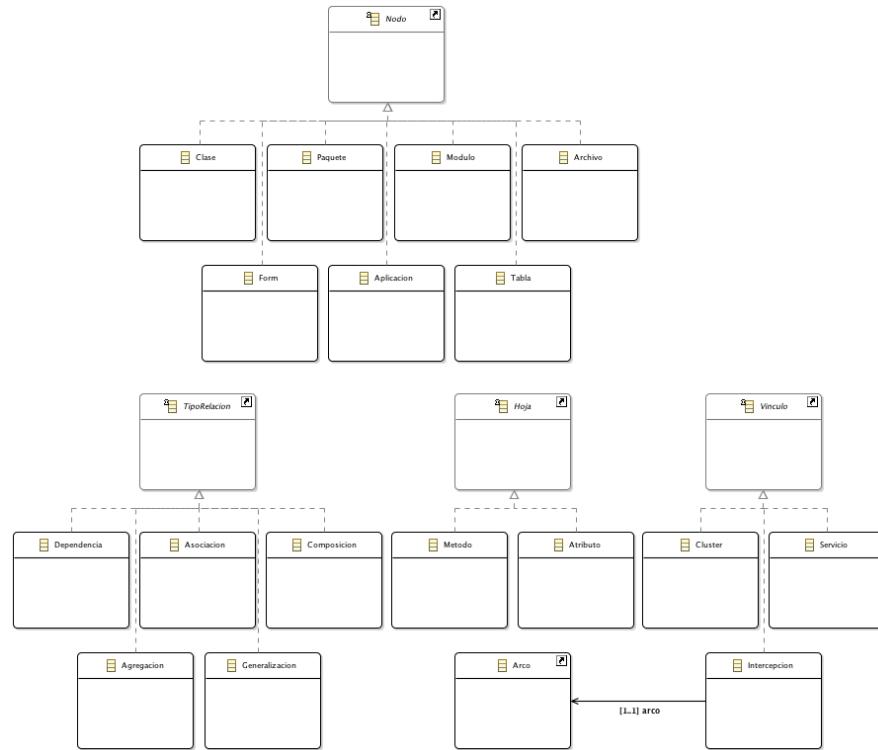


Figura 16.: Metamodelo Visualización-Representación

taje de dependencia entre un cluster y otro. A partir del porcentaje de dependencia se podrá analizar y obtener una propuesta inicial de la arquitectura orientada a micro servicios.

Finalmente por cada clase se asocian sus relaciones y sus respectivas métricas, tales como: nivel de acoplamiento, número de líneas de código, número de relaciones de tipo uso, número de relaciones de tipo herencia entre otras.

El modelo obtenido procura ofrecer una estructuración del software que pueda ser representado gráficamente.

3.5.5 Transformación Visualización a Diagramación

A partir del modelo conforme al meta-modelo de visualización - representación, se realiza una transformación ETL modelo a modelo para mapear la forma de visualización con respecto a las representaciones encontradas en el modelo.

Paralelamente en la herramienta Sirius, se crea un *Viewpoint Specification*, conforme al meta-modelo de visualización, lo que le permite a Sirius dibujar elementos gráficos acorde a los elementos encontrados en el modelo.

Con el modelo resultado de la transformación, se hace una configuración para que sea procesado por SIRIUS, con lo que se generan los diagramas junto con los elementos gráficos mapeados en el *Viewpoint Specification*.



Figura 17.: Transformación Visualización a representación

El meta-modelo usado para esta transformación se compone de elementos gráficos tales como figuras, contenedores y conectores, así como componentes que *decoran* los elementos gráficos. A continuación se explican los elementos más significativos del meta-modelo:

- **ElementoDiagrama:** Elemento base que representa los nodos dibujables, está *decorado* con un elemento de tipo «Label»
- **Label:** Elemento que contiene todas las características que Sirius implementa para los labels.
- **Conector:** Elemento gráfico que relaciona dos «ElementoDiagrama» a través de una línea. Este elemento está especializado en «Asociacion», «Herencia», «Implementacion», «Dependencia», «Agregacion» y «Composicion», donde en el *Viewpoint Specification* está especificada la forma de dibujar estos tipos de conectores.
- **Contenedor:** Subtipo de «ElementoDiagrama», que representa elementos gráficos que contienen a su vez más elementos. Está configurado en el *Viewpoint Specification* para que cada elemento de este tipo, sea capaz de generar un nuevo diagrama para poder visualizar el contenido del mismo.
- **FiguraGeometrica:** Subtipo de «ElementoDiagrama», que representa las características comunes a las figuras dibujables; tal como color, largo y ancho.
- **NavegacionDiagrama:** Interface que permite hacer de un «ElementoDiagrama» un enlace a un nuevo diagrama, en Sirius se ve reflejado al momento de interactuar con un elemento, generando un nuevo diagrama.
- **Diagrama:** Subtipo de «Contenedor». Dado que «Contenedor» es una interface, los elementos tipo «Diagrama» son los diagramas genéricos que representan elementos de software como clases o paquetes.

3.5 IMPLEMENTACIÓN

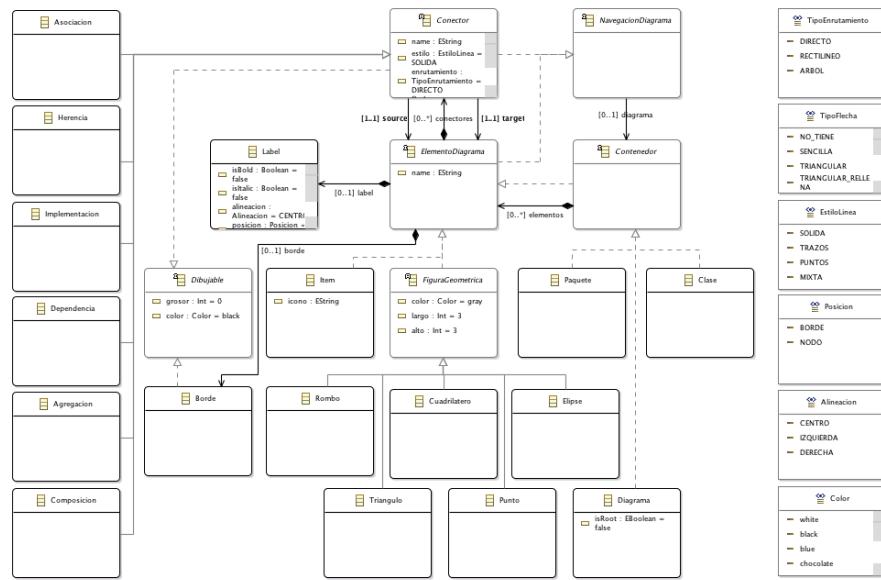


Figura 18.: Metamodelo Visualización-Diagramación

Disponer de un meta-modelo aparte de diagramación, permite a futuro implementar cualquier herramienta, no solo Sirius, para la generación de los diagramas.

Como resultado de la ejecución de todos estos elementos, en Sirius se visualizan dos tipos de diagramas, el primero, un diagrama personalizado de Cluster, que representa un cluster por cada EJB de la aplicación y la dependencia entre estos. En este diagrama se representa con el color el número de clases, con el tamaño el número de líneas en total y el grosor de las relaciones representa que tan dependiente es un cluster de otro.

En el diagrama, dando clic sobre un cluster se puede visualizar un diagrama de paquetes y clases que están contenidas en éste, igualmente dando clic sobre la relación entre dos cluster se visualiza las clases comunes entre ambos.

En el segundo diagrama, el cual es un diagrama genérico, se representa la estructura del software en paquetes, clases y relaciones. En este diagrama se puede navegar entre paquetes para visualizar por cada uno las clases contenidas.

3.5.6 Convenciones Métricas a Diagramación

Con el fin de enriquecer los diagramas generados, se tomaron las siguientes decisiones para representar visualmente las siguientes métricas:

- NumberOfClassesMeasure: En el diagrama de clusters se representa el número de clases a través del radio de la circunferencia de cada cluster, de esta manera los clusters más grandes son los que contienen la mayor cantidad de clases. Para evitar desproporciones en los distintos tamaños de los clusters, se determina el tamaño mayor (20 px) y menor (5 px), y se escalan los distintos tamaños con referencia a estos límites. Para las relaciones entre clusters, también se representó esta métrica definiendo el grosor de la relación, escalando los valores máximos (5 px) y mínimos (1 px) y de esta manera visualizando las relaciones más «densas» dentro del diagrama.
- NumberOfLinesMeasure: En el diagrama de clusters se representa el número de líneas por medio del color en cada cluster. Se usó como referencia el código de color de las alertas por huracán para representar el número de líneas que tiene cada componente con respecto a al componente con el mayor y menor número de líneas:
 - azul: 20 % o menos
 - verde: Entre el 21 % y el 40 %
 - amarillo: Entre el 41 % y el 60 %
 - naranja: Entre el 61 % y el 80 %
 - rojo: 81 % o más
- CouplingBetweenObjects: En las relaciones cuando viene dicha métrica, se indica el valor como una etiqueta (label) sobre la relación
- ClusterRelationConflictPercentageMeasure: En el diagrama de clusters se coloca en los extremos de las relaciones entre clusters el porcentaje de relación

Parte III
VALIDACIÓN

4

VALIDACIÓN Y RESULTADOS

La validación de la solución que se propone se realizó sobre dos casos de estudio: Carrito de compras y SisInfo.

4.0.7 *Carrito de Compras*

Carrito es un sitio virtual en el cual un cliente puede realizar la compra de productos. Esta es una aplicación que implementa una arquitectura por niveles. El primer nivel contiene las capas de persistencia y negocio, el segundo contiene la capa de servicios, y el tercero contiene la capa de presentación. Finalmente la capa de presentación sigue una arquitectura MVC con eventos que están implementados utilizando el framework Backbone.

La cadena de transformación implementada se aplica sobre la capa de persistencia y negocio, sobre ésta se logra obtener un análisis rápidamente y sin inconvenientes, ya que es una aplicación pequeña y bien desarrollada. De ésta se obtuvo una gráfica en la que se muestran los cluster obtenidos con sus respectivas dependencias.

Cada cluster identificado con una circunferencia varía en color y tamaño según su número de clases y número de líneas de código. A partir de esta gráfica por cada relación se pueden identificar las clases en común entre dos cluster, con lo cual se puede determinar el nivel de dependencia entre ellos, lo cual permitirá posteriormente diseñar los micro servicios. Así mismo, también se puede identificar el nivel de acoplamiento entre cada cluster a través de los porcentajes dibujados en las relaciones.

A partir de las gráficas obtenidas de esta aplicación se puede observar que con la arquitectura implementada se proporciona un bajo nivel de acoplamiento. Se pudo identificar una relación fuerte entre el item y el master, ya que el master controla las operaciones principales y el item contiene los productos que adquiere un cliente, de esta forma se puede determinar que la aplicación podría migrarse a una arquitectura con 3 micro servicios: producto, cliente y master.

VALIDACIÓN Y RESULTADOS

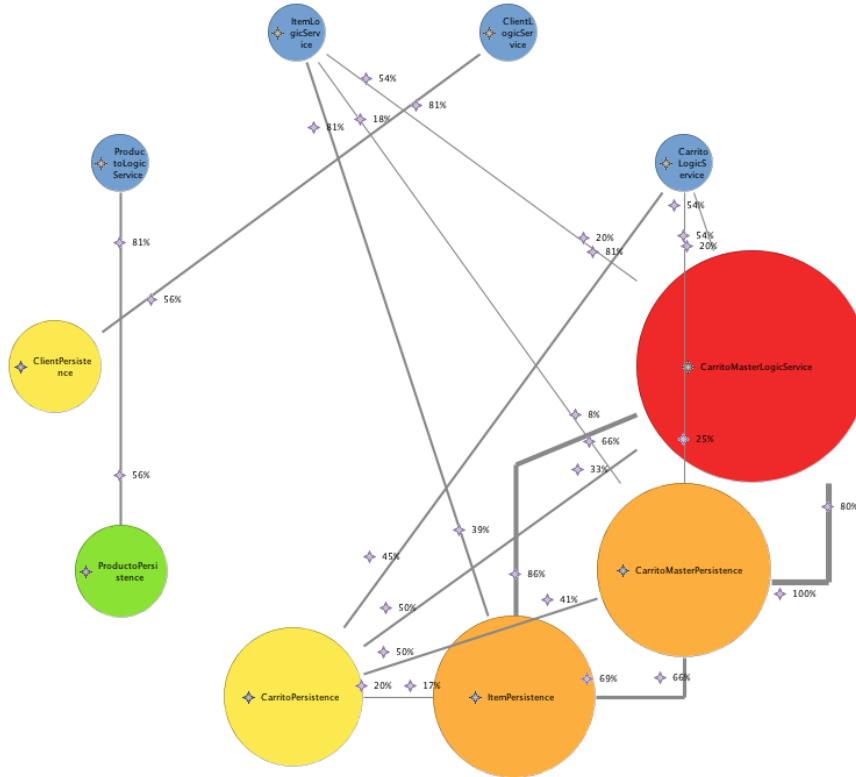


Figura 19.: Diagrama de Cluster

Como se observa en la figura 19, producto y cliente son independientes por lo cual contendrían la capa de persistencia y servicios, cada uno con sus operaciones propias y por su parte el master tendría la administración del ítem y la administración principal de las operaciones que implican la compra de productos.

Como resultado de las métricas obtenidas, se planteó realizar una simplificación de los cluster y de esta manera ofrecer un diagrama que agrupara los cluster con mayor porcentaje de acoplamiento. Esta agrupación sería la propuesta inicial de micro servicios.

En las figuras 20, 21, 22 y 23 se pueden observar varias alternativas de simplificación.

VALIDACIÓN Y RESULTADOS

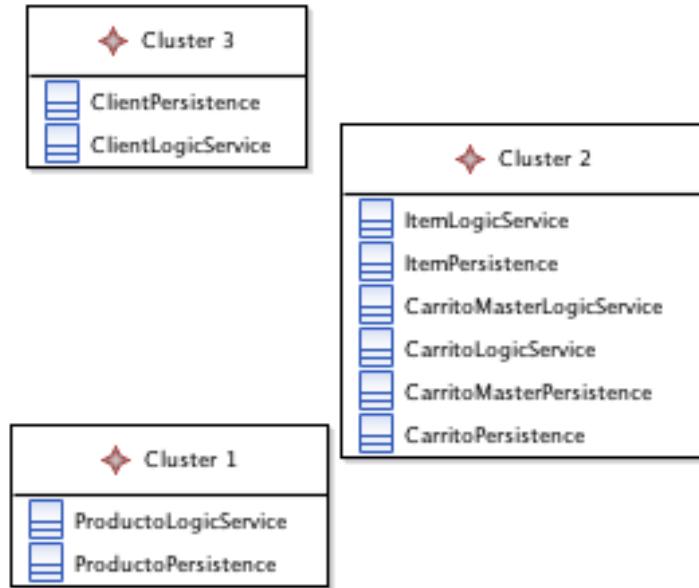


Figura 20.: Diagrama de cluster simplificado al 50 % de acoplamiento

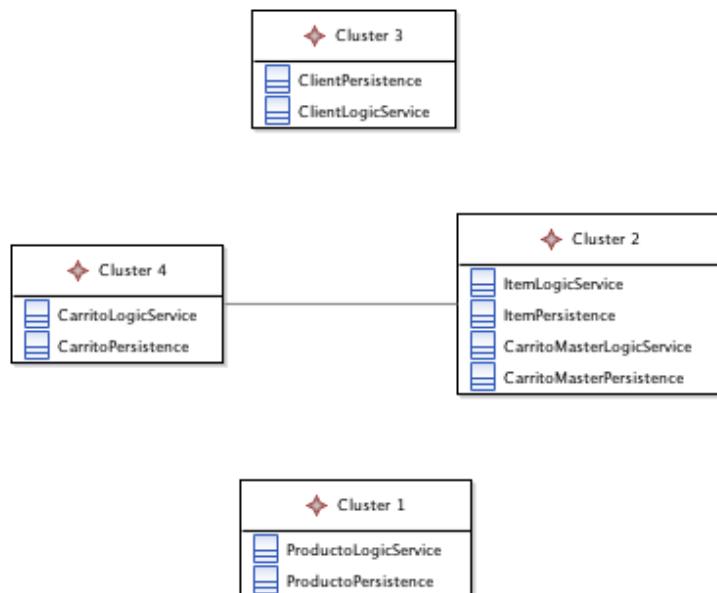


Figura 21.: Diagrama de cluster simplificado al 65 % de acoplamiento

VALIDACIÓN Y RESULTADOS



Figura 22.: Diagrama de cluster simplificado al 80 % de acoplamiento

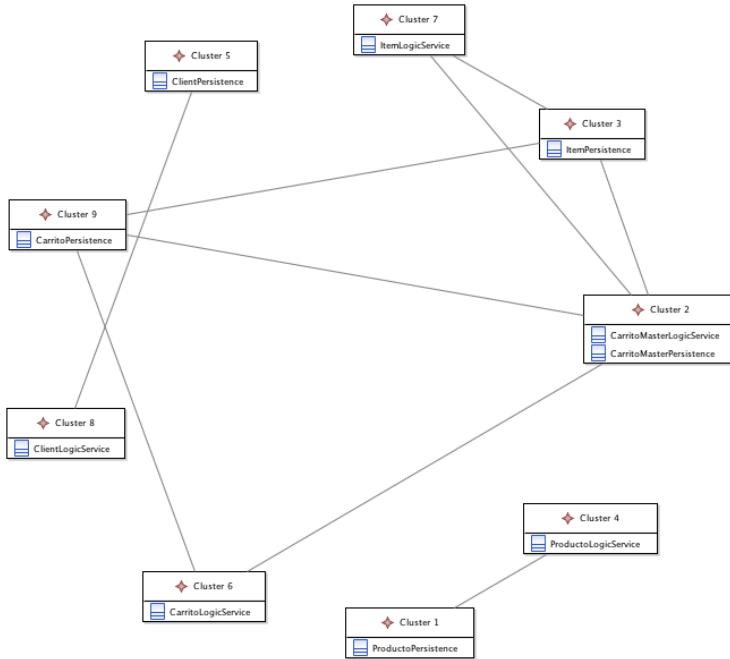


Figura 23.: Diagrama de cluster simplificado al 90 % de acoplamiento

En los ejemplos de simplificación y agrupación con las métricas de porcentaje de pertenencia entre clusters, brinda distintas alternativas de creación de servicios dependiendo del nivel de independencia que se requiera entre los componentes identificados.

A medida que el porcentaje de acoplamiento es menor, se identifican los servicios con nula interdependencia entre sus componentes; conforme se aumenta el porcentaje de acoplamiento crece el número de servicios pero así mismo se empiezan a identificar relaciones entre los servicios.

Finalmente con un alto porcentaje de acoplamiento, el resultado se asemeja al diagrama original por clusters.

Al realizar una comparación con la propuesta de Edgar Sandoval [San], se encuentra un resultado similar teniendo en cuenta que el algoritmo implementado para analizar la dependencia entre cluster es diferente, en ambos resultados se identifica la relación fuerte existente entre master, item y carrito y totalmente independiente el módulo de producto.

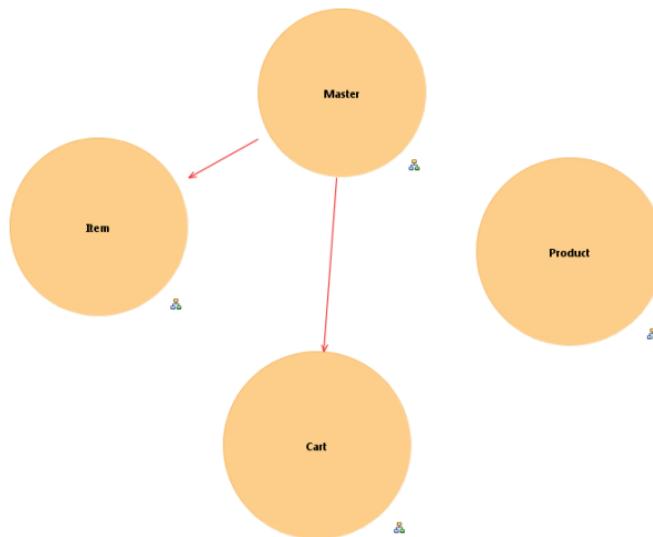


Figura 24.: Modelo Radial Carrito - tomado de [San]

4.0.8 *SisInfo*

SisInfo es una aplicación de la universidad de los Andes, encargada de gestionar procesos del departamento de sistemas y computación, como el desarrollo de tesis, programación de eventos, ofertas de empleo, calificaciones de los cursos, consultoría, estadísticas estudiantiles, graduaciones e información bibliografía docente.

Esta es una aplicación compleja que no cumple con muchos de los estándares de buena codificación, y por lo mismo la ejecución de la cadena de transformación es un proceso demorado (de varias horas) y debido a su complejidad, las gráficas obtenidas son difíciles de analizar. En primer lugar se identificó que la obtención del modelo KDM por MoDisco no pudo ser obtenida para todo el proyecto y se elimi-

naron los paquetes relacionados con pruebas unitarias y migración de datos históricos. En segundo lugar la identificación y sintetización de las relaciones es un proceso que, para este proyecto en particular, toma hasta 18 horas, lo cual da un primer vistazo a su complejidad y acoplamiento. En general se observa que para proyectos grandes y complejos (proyectos muy acoplados y de mas de 1000 clases e interfaces) MoDisco no es capaz de obtener el modelo de las aplicaciones y almacenarlo en el disco duro.

Al analizar la gráfica obtenida, se puede observar la existencia de clusters aislados que podrían ser candidatas para su eliminación o identificarlos como servicios utilitarios para ser agrupados; adicionalmente se puede observar que es una aplicación altamente acoplada lo cual dificulta una posible migración a micro servicios.

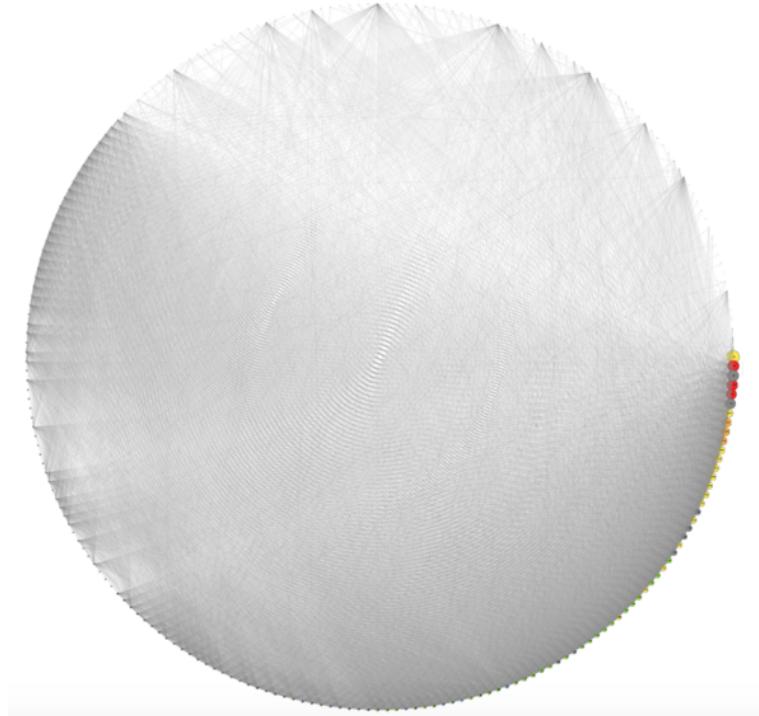


Figura 25.: Cluster SisInfo

En el siguiente enlace se puede descargar la gráfica para ser visualizada adecuadamente [ClusterSisInfo.svg](#)

De igual forma, se realizó un proceso de simplificación de los clusters identificados, para obtener la propuesta inicial de microservicios para SisInfo. En las figuras 26, 27 y 28 se pueden observar varias alternativas de simplificación.

VALIDACIÓN Y RESULTADOS

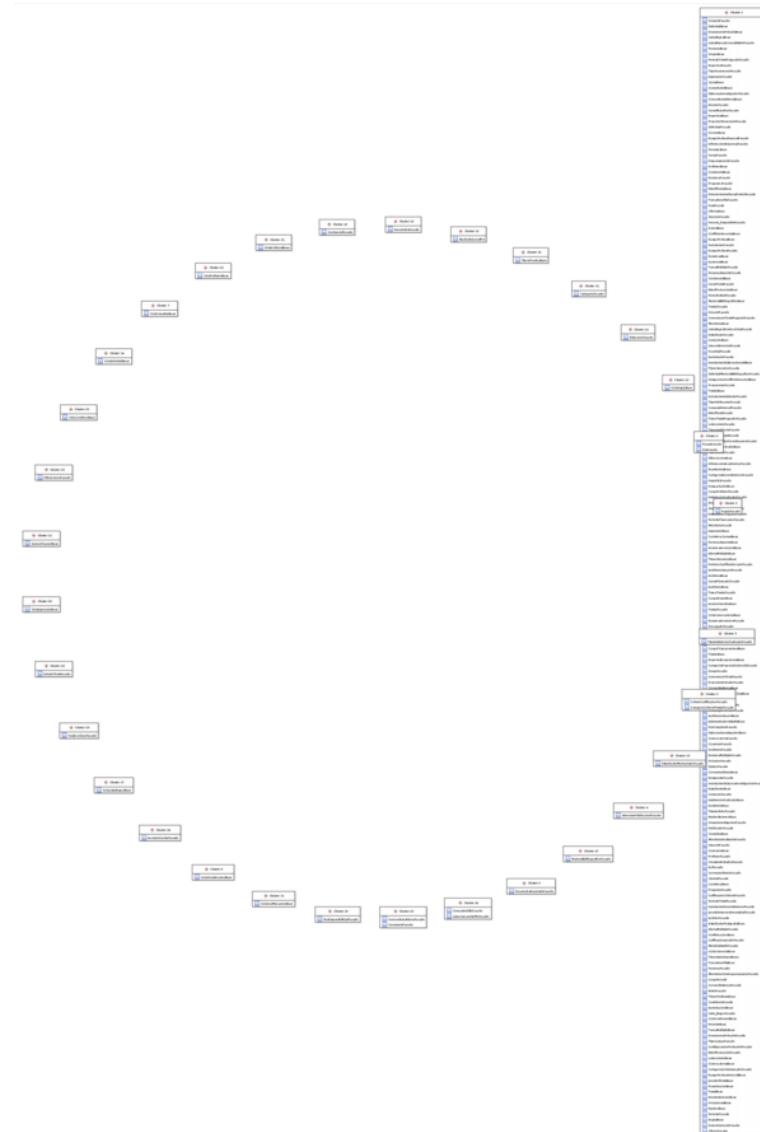


Figura 26.: Diagrama de cluster SisInfo simplificado al 5% de acoplamiento

VALIDACIÓN Y RESULTADOS

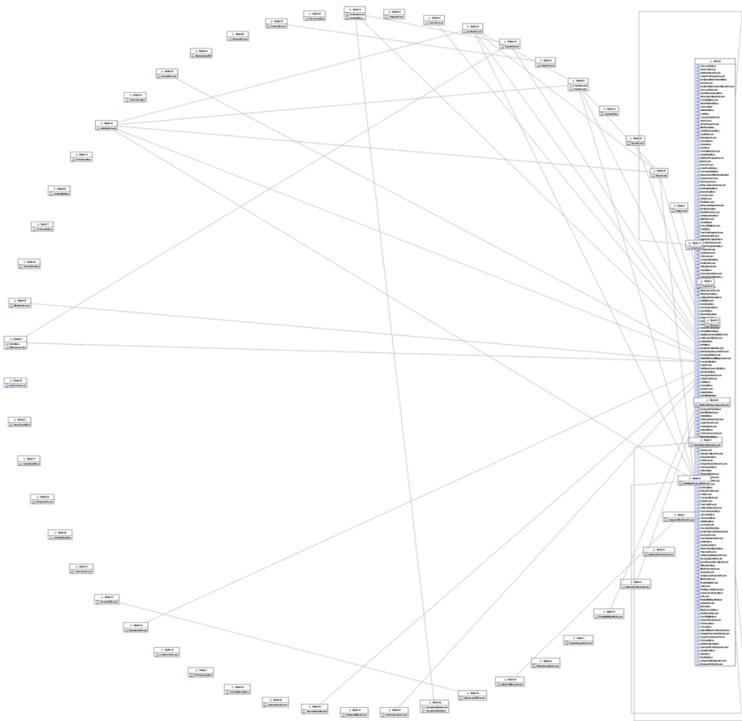


Figura 27.: Diagrama de cluster SisInfo simplificado al 50 % de acoplamiento

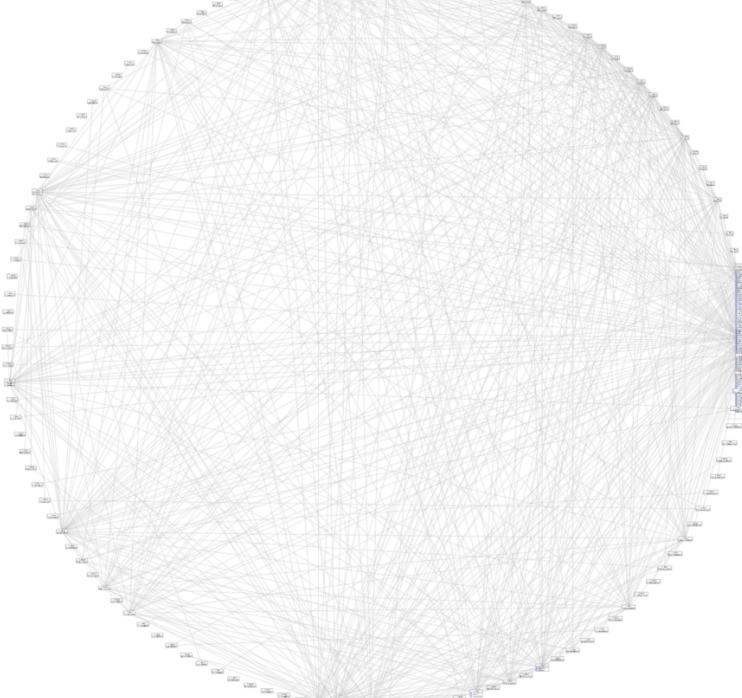


Figura 28.: Diagrama de cluster SisInfo simplificado al 75 % de acoplamiento

- Simplificación al 5 %: Por el bajo nivel de acoplamiento necesario para agrupar clusters en servicios, se encontraron 33 servicios totalmente independientes, donde casi en su totalidad todos los servicios tienen solo un elemento, otros 3 servicios con dos elementos y un servicio llamado «Cluster 1» con prácticamente la totalidad de los componentes de la aplicación (Figura 26). Descargar en [ClusterSimplificado5%.svg](#)
- Simplificación al 50 %: Se encuentran 55 servicios, donde casi en su totalidad todos los servicios tienen solo un elemento, otros 4 servicios con dos elementos y un servicio llamado «Cluster 1» con prácticamente la totalidad de los componentes de la aplicación; en este diagrama se encuentran alrededor de 30 relaciones entre los distintos servicios (Figura 27). Descargar en [ClusterSimplificado50%.svg](#)
- Simplificación al 75 %: Este diagrama ya presenta cierta dificultad para su correcta interpretación, el servicio llamado «Cluster 1» es significativamente menor y así mismo a crecido el número de servicios y su relaciones. Ya se pueden indentificar servicios de 3 elementos (Figura 28). Descargar en [ClusterSimplificado75%.svg](#)

Realizando una comparación con la reorganización propuesta en [San], se observa de igual forma que hay una gran cantidad de dependencias entre módulos

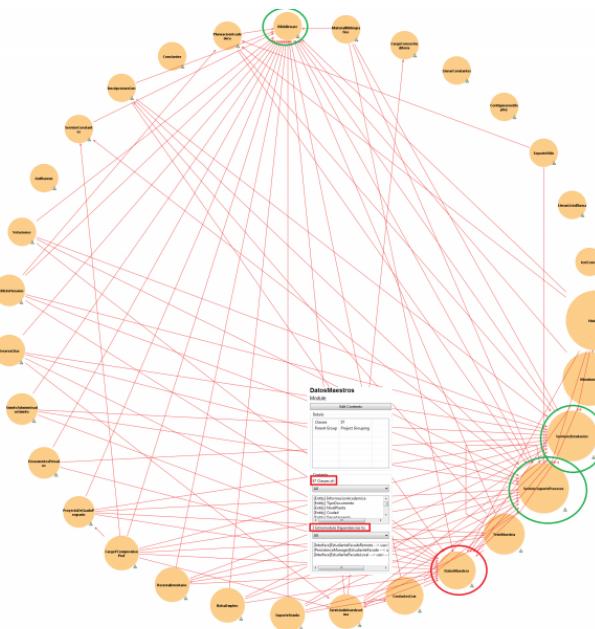


Figura 29.: Modelo Radial SisInfo - tomado de [San]

5

CONCLUSIONES Y TRABAJO FUTURO

Esta solución se realizó con el propósito de retomar la problemática planteada en [San]:

- Recolección de la información: Como recolectar la información necesaria desde el código fuente.
- Construcción de información estructurada: Como unir toda la información recolectada y la organización estructurada de la misma.
- Visualización de la información estructurada: Cuales son las mejores estrategias para presentar la información estructurada.
- Reorganización: Proveer o sugerir la reorganización del código fuente y la estructura del software.

El uso de KDM permite desacoplar la solución de la herramienta que obtiene el modelo inicial y da un estándar para estructurar la información en un modelo aceptado por la comunidad de ingeniería de software. Es responsabilidad de aquel que implemente la transformación del software al modelo KDM ajustarse al estándar y a las definiciones asociadas al mismo. Con respecto a los modelos KDM, SMM y VMM la ventaja es que se obtiene una terminología amplia y completa para entender y representar el conocimiento del software.

La herramienta implementada, ofrece a los usuarios una ayuda para determinar si las aplicaciones son aptas para ser modernizadas a micro servicios, ya que el análisis que se realiza al software logra identificar la complejidad y apoya en el entendimiento del mismo.

La representación gráfica que se obtiene muestra claramente al arquitecto la estructura completa y le ayuda a identificar posibles inconvenientes de la implementación actual, adicionalmente le da una propuesta inicial de la arquitectura orientada a micro servicios, basada en un porcentaje de dependencias entre cluster.

A partir de este porcentaje se puede definir las clases que pondrían los micro servicios y por ende el tamaño de estos. Teniendo en cuenta que un cluster está compuesto por clases y dependencias

con otros cluster se concluye que si el porcentaje de dependencia es muy alto las clases entre ambos cluster conformarían un micro servicio. Finalmente el arquitecto partiendo de esta propuesta puede redefinir los micro servicios según sus necesidades.

Con respecto a carrito, por su tamaño y estructura es fácil identificar una separación por micro-servicios, pero los obtenidos son muy pequeños para ser favorecidos por este estilo de arquitectura. Esto se debe a que la aplicación fue diseñada desde el principio pensando en separación de responsabilidades. Sisinfo, por otro lado, es un caso opuesto, como se puede ver en la figura 25 la gráfica de clusters es muy amplia y difícil de entender por lo cual se hace necesario incluir una gráfica que muestre como se agrupan por el porcentaje de la relación. Ejecutar la cadena de transformaciones completa sobre un proyecto de tal tamaño y complejidad puede tardar más de un día, por lo cual se debe proponer una forma más eficiente de realizar las transformaciones.

Para posteriores trabajos, se sugiere continuar implementando una solución que le permita al arquitecto redefinir los micro servicios desde la misma herramienta realizando las modificaciones según su criterio. Dado que los meta-modelos y el análisis de los mismos usados en el proyecto son independientes del lenguaje y hay cierta independencia sobre la herramienta que obtiene el modelo KDM, es importante continuar trabajando con los algoritmos de separación por clusters incluidos en el plugin encontrando soluciones a sus problemáticas (ver anexo A). Con respecto al cluster por EJBs, es una herramienta de análisis orientada al lenguaje JAVA, pero su concepto puede ser transformado a cualquier lenguaje si se pueden identificar las unidades encargadas de ofrecer los servicios de negocio y las unidades encargadas de representar los datos.

De igual forma se puede proponer un generador de código fuente de la aplicación modernizada, incluyendo las modificaciones solicitadas por el arquitecto. MDE permite definir transformaciones bidireccionales. Es interesante permitir la reorganización del código o de los componentes en la representación de visualización y completar la cadena de transformación para obtener nuevamente el código reorganizado y exponiendo los micro servicios identificados.

Inicialmente se pensó en realizar un plugin sobre el framework SonarQube para hacer más fácil la instalación, ejecución y visualización de toda ésta transformación. Se encontraron limitaciones técnicas que no permitieron esto (ver anexo B.1).

A

OTROS ALGORITMOS DE CLUSTERING

Este anexo presenta otros algoritmos de clustering que se incluyeron en el proyecto (ver 3.5.2), pero que por su dificultad y necesidad de intervención por parte del usuario no fueron incluidos en la propuesta.

A.1 CLUSTERING POR ÁRBOL DE EXPANSIÓN MÍNIMA (MST)

El clustering por Árbol de Expansión Mínima, es una propuesta inicial para obtener divisiones del software dada la métrica del CCM. Su propósito es evidenciar los elementos o nodos del software (Clases, Interfaces, etc.) y las relaciones entre ellas como un grafo no dirigido, cuyas relaciones tienen un peso. El peso de una relación entre los nodos i y j se define de la siguiente forma:

$$W_{ij} = \frac{1}{CCM_{ij}} \quad (1)$$

Donde CCM_{ij} se refiere a la medida del acoplamiento entre los nodos i y j . Se observa que W_{ij} está definida únicamente si la relación entre i y j está definida. La figura 30 muestra un ejemplo inicial de un grafo, donde los valores de las relaciones corresponden al CCM (ver 3.3) entre los nodos asociados.

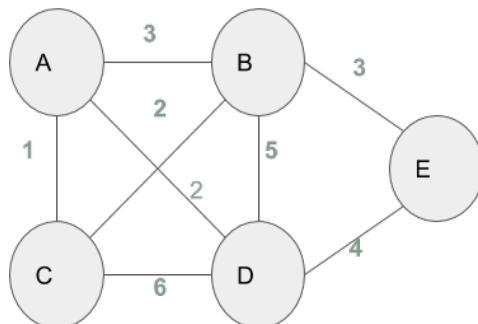


Figura 30.: Grafo inicial

A.1 CLUSTERING POR ÁRBOL DE EXPANSIÓN MÍNIMA (MST)

El siguiente paso es usar el *algoritmo de Prim* para obtener el Árbol o Bosque de Expansión Mínima (ver [SW11]). Se define un Bosque de Expansión Mínima como un conjunto de Árboles de Expansión Mínima y se obtiene si el grafo original no es conexo. El Árbol o Bosque obtenido asegura que las relaciones que se mantienen, son las de mínima distancia entre los nodos (ver la figura 31).

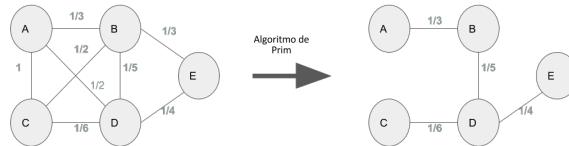


Figura 31.:

El último paso se refiere a la obtención de los clusters; en este momento el usuario debe tomar una decisión con respecto al número de clusters que se quieren obtener. Suponga que el resultado del algoritmo de Prim es un Árbol conexo y que se desean obtener n clusters, la idea es eliminar las $n - 1$ relaciones de mayor peso. Es importante mencionar que las relaciones eliminadas son las de menor CCM del Árbol de Expansión Mínima y que los n clusters obtenidos son los de mayor acoplamiento según se define la medida del CCM entre clases (ver la figura 32).

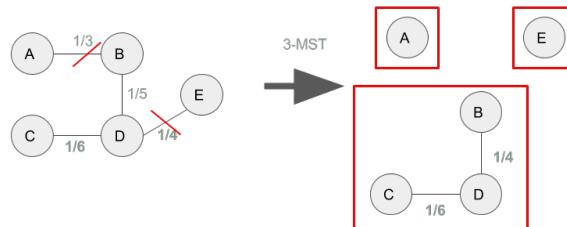


Figura 32.:

En la herramienta hay una versión de éste algoritmo implementada, sin embargo existe una problemática que impide incluirlo como parte del análisis. El primer problema es la medición del peso, se puede notar que es complicado definir una equivalencia para los diferentes tipos de relaciones. Por ejemplo la clase A puede hacer uso extensivo de la clase B, es decir que su acoplamiento es alto, y que además A extiende de la clase C; dependiendo del factor del usuario es posible

A.2 CLUSTER POR EDGE BETWEENNESS

que el acoplamiento entre las clases A y B sea más alto que el acoplamiento de las clases A y C pero no tiene sentido separar el grafo en una relación de extensión.

En los experimentos realizados se observa que sin un análisis correcto se obtienen $n - 1$ clusters de una o dos clases y un cluster con el resto de las clases. Esta problemática permite descartar el algoritmo como ayuda para el entendimiento del software.

A.2 CLUSTER POR EDGE BETWEENNESS

La idea de este algoritmo es, a partir de un grafo, revisar las relaciones entre los elementos para identificar comunidades y las relaciones entre las comunidades. En cada iteración, se revisan todas las relaciones del grafo y se les asigna a cada una un valor de Edge Betweenness; para cada par de nodos se identifica el camino más corto que los une, si la relación a revisar está contenida en el camino (en caso de existir) se suma una unidad al valor de Edge Betweenness (ver 33).

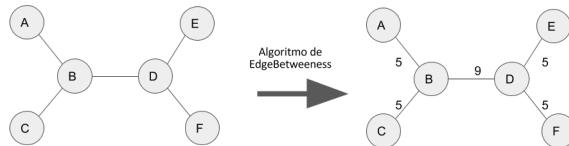


Figura 33.:

El siguiente paso es eliminar la relación que tiene la medida de Edge Betweenness más alta (ver 34).

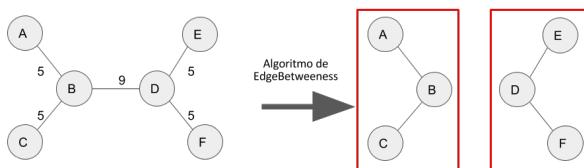


Figura 34.:

Las figuras 33 y 34 ilustran el ejemplo más básico que puede definirse para explicar el algoritmo, sin embargo en casos donde el grafo es densamente conectado y no es fácil distinguir comunidades el algoritmo tiende a requerir mas iteraciones para encontrar una separación, que posiblemente no sea adecuada.

B

LIMITACIONES TÉCNICAS

B.1 SONARQUBE

Toda la implementación se pretende integrar sobre la plataforma de manejo de calidad SonarQube para poder realizarla automáticamente sobre proyectos administrados sobre éste. Esta plataforma permite el desarrollo de plugins Maven para la fácil integración y personalización de métricas, vistas y gráficas. Estos plugins se ejecutan al analizar los proyectos manejados, por tanto, se puede realizar la transformación bajo el proyecto que se requiera.

Para poder integrar la transformación se utilizaron las librerías de cada una de las partes, para esto se incluyeron las librerías necesarias de Epsilon, de ésta manera se pueden ejecutar las transformaciones desde código Java compilado en el plugin. Ya que las librerías de Epsilon no se encuentran en ningún repositorio de manera oficial, se tomaron las librerías disponibles en los plugins de Eclipse IDE y se instalaron localmente para que no generara errores de compilación ni de ejecución.

El desarrollo de estos plugin se basan en extensiones. La clase principal para definir el plugin tiene como nombre SonarPlugin. En ésta clase se implementa el método getExtensions() donde se definen las clases que conforman el plugin. En el siguiente ejemplo se puede apreciar el uso de SonarPlugin:

```
public class IDEMetadataPlugin extends SonarPlugin {
    public List<Class<? extends Extension>> getExtensions() {
        return Arrays.asList(IDEMetadataMetrics.class,
            IDEMetadataSensor.class,
            IDEMetadataDashboardWidget.class);
    }
}
```

Es necesario declarar en el archivo de dependencias pom.xml la dependencia hacia el API del plugin de sonar y exponer la clase que extiende a SonarPlugin:

```

<dependencies>
    <dependency>
        <groupId>org.codehaus.sonar</groupId>
        <artifactId>sonar-plugin-api</artifactId>
        <version>3.7.3</version>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.codehaus.sonar</groupId>
            <artifactId>sonar-packaging-maven-plugin</artifactId>
            <version>1.7</version>
            <extensions>true</extensions>
            <configuration>
                <pluginClass>tools.sonarplugin.
                    simplifieddecisionmetrics.
                    IDEMetadataPlugin</pluginClass>
            </configuration>
        </plugin>
    </plugins>
</build>

```

Las extensiones soportadas pueden ser de tres tipos: Metrics, Sensor o Widget. La clase Metrics sirve para definir métricas específicas que serán guardadas y que otros plugin pueden utilizar. En ésta clase se debe implementar el método getMetrics y se pueden definir métricas con la clase Metric que provee el API de la siguiente forma:

```

public class IDEMetadataMetrics implements Metrics {

    public static final Metric NUMBER_OF_PACKAGES =
        new Metric.Builder(
            "number_of_packages", // metric identifier
            "Number of packages", // metric name
            Metric.ValueType.INT)// metric data type
            .setDescription("The name of packages used on the project")
            .setQualitative(false).setDomain(CoreMetrics.DOMAIN_GENERAL)
            .create();

    public List<Metric> getMetrics() {
        return Arrays.asList(NUMBER_OF_PACKAGES);
    }
}

```

La clase Widget se utiliza para definir visualizaciones dentro de los dashboard de SonarQube para mostrar métricas, gráficas o infor-

mación de interés para la persona que administra cada proyecto. La clase principal para definir Widgets es RubyRailsWidget, que define los Widget por medio de lenguaje Ruby. Se le deben agregar anotaciones para definir categorías, roles y descripciones. Se sugiere utilizarla con la clase AbstractRubyTemplate para que tome por medio de un archivo la configuración de la siguiente manera:

```

@UserRole(UserRole.USER)
@Description("")
@WidgetCategory("")
public class IDEMetadataDashboardWidget extends AbstractRubyTemplate
    implements RubyRailsWidget {

    public String getId() {
        return "simplifieddecisionmetrics";
    }

    public String getTitle() {
        return "Simplified Decision Metrics";
    }

    @Override
    protected String getTemplatePath() {
        return "/simplifieddecisionmetrics/sdm_widget.html.erb";
    }
}

```

La clase Sensor es la que ejecuta el método analyse con la información del proyecto. De ésta forma se puede tomar la información necesaria dentro del proyecto y hacer las transformaciones necesarias:

```

public class IDEMetadataSensor implements Sensor {

    private ModuleFileSystem fileSystem;

    public IDEMetadataSensor(ModuleFileSystem fileSystem) {
        this.fileSystem = fileSystem;
    }

    public boolean shouldExecuteOnProject(Project projectInfo) {
        return true;
    }

    public void analyse(Project projectInfo, SensorContext sensorContext) {
        //TODO Transformaciones necesarias
    }
}

```

B.2 ANÁLISIS APLICACIÓN ICFES

Al intentar integrar Modisco dentro del plugin, el framework no lo permitió ya que se necesita obligatoriamente una instancia de Eclipse IDE para poder realizar la extracción y análisis del código. Ésto impide que esta transformación se pueda integrar como plugin Maven dentro de SonarQube. Aún así, se empaquetaron meta-modelos y modelos quemados para comprobar que las transformaciones posteriores a Modisco se puedan ejecutar desde SonarQube con resultados satisfactorios.

Una posible solución a éste inconveniente es independizarlo, desarrollando una implementación tipo servidor, que descargue el proyecto con la información suministrada, que genere el archivo dependiendo del código y lo retorne al plugin de SonarQube que se esté ejecutando.

B.2 ANÁLISIS APLICACIÓN ICFES

Se realizó un intento para analizar la aplicación del ICFES, sin embargo MoDisco generó un problema en la generación del modelo conforme a KDM y no fue posible continuar con el proceso de análisis.

C

INSTALACIÓN Y EJECUCIÓN

C.1 REPOSITORIO

La implementación cuenta con 3 proyectos, actualmente están como repositorios en GitHub, uno contiene las transformaciones modelo a modelo, el segundo es el plugin encargado de la ejecución de los algoritmos de clustering, y el tercero es el proyecto que visualiza las métricas en SIRIUS.

- Repositorio de la Transformación:
https://github.com/rolandoamarillo/miso4301_201520.git
- Repositorio Plugin Clúster:
<https://github.com/danesco0507/pluginetl.git>
- Repositorio Visualización SIRIUS:
<https://github.com/dianadcs/VisualizacionSirius.git>

C.2 CLONACIÓN DEL REPOSITORIO

Para descargar los repositorios puede hacer uso de herramientas como SourceTree, (<https://www.sourcetreeapp.com/>).

Desde la pantalla principal de sourceTree adicione el repositorio desde la opción nuevo repositorio/clonar, donde le solicitará indicar la url del repositorio y la ruta de destino, ver figuras: 35 y 36.



Figura 35.: Clonar Repositorio

C.3 PRE REQUISITOS

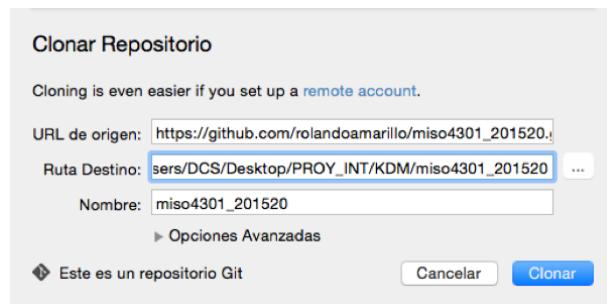


Figura 36.: Clonar Repositorio

Se realiza el mismo proceso para PluginEtl y el proyecto de SIRIUS, si el proceso se completó exitosamente se visualizarán en la pantalla principal de SourceTree los repositorios descargados (ver figura 37), de esta forma ya podrán ser importados a eclipse.



Figura 37.: Repositorios en SourceTree

C.3 PRE REQUISITOS

Para ejecutar la transformación completa y obtener la visualización en SIRIUS se requiere de eclipse Modeling Tools, la versión usada es Eclipse Luna, ésta versión la puede descargar de: <https://eclipse.org/downloads/packages/release/Luna/SR2>.

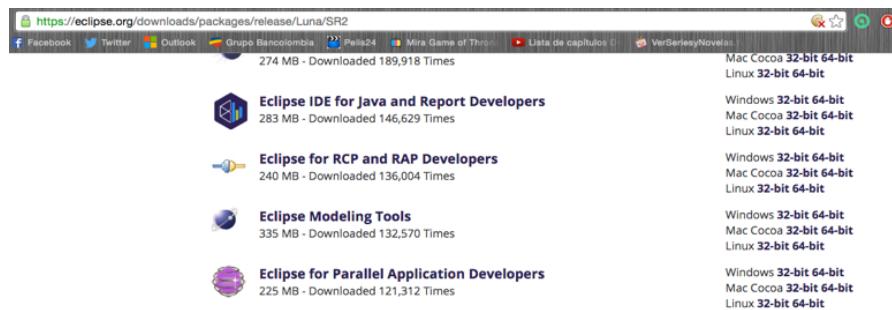


Figura 38.: Eclipse Luna

C.3 PRE REQUISITOS

Adicionalmente debe tener en cuenta que debe instalar los componentes necesarios, en eclipse diríjase al menú help/Install Modelling Component (ver figura 39) y seleccione: Acceleo, Graphical Modeling Framework Tooling,Papyrus (incubation), OCL Tools, MoDisco (Incubation) y SIRIUS. Adicionalmente debe instalar EcoreTools, para esto valla a Help/Install New Software (ver figura 40).

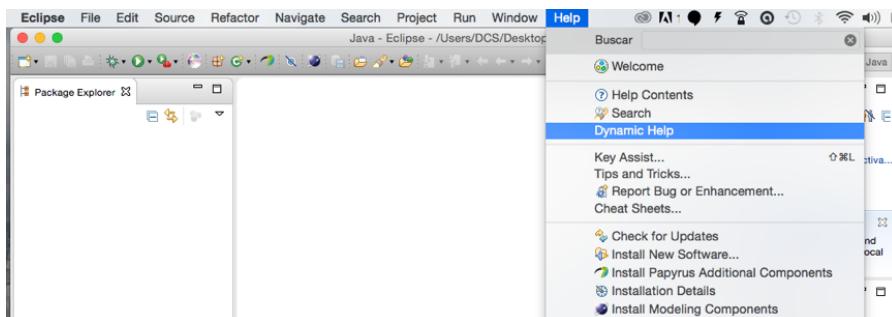


Figura 39.: Instalación Modelling Component

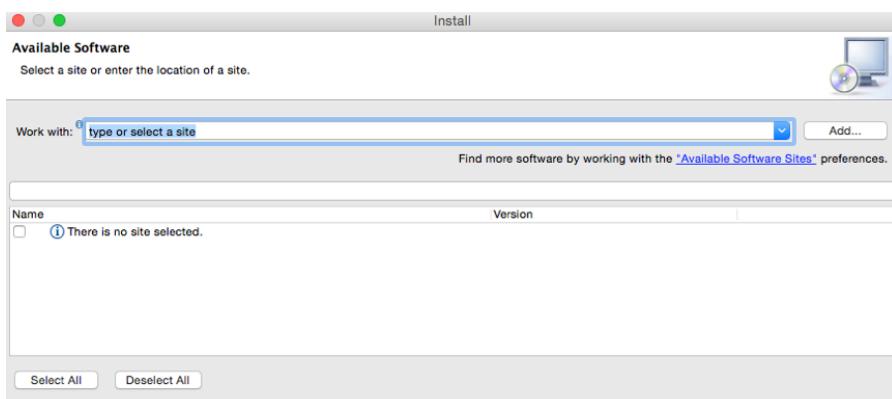


Figura 40.: Instalación EcoreTools

En la ventana de instalación de nuevo software, elegir la opción Add, escribir en Name “EcoreTools” y en Location la siguiente dirección: <http://download.eclipse.org/ecoretools/updates/nightly/2.0.x/luna>, finalice la instalación y podrá continuar con la ejecución de las transformaciones.

Finalmente se debe tener en cuenta que la ejecución del proyecto se realizó con la versión 1.2.0.201408 de epsilon y la versión 2.0.4.201 de SIRIUS.

C.4 EJECUCIÓN DE LA TRANSFORMACIÓN EN ECLIPSE

C.4 EJECUCIÓN DE LA TRANSFORMACIÓN EN ECLIPSE

En primer lugar se debe seleccionar una aplicación JEE y abrirla en eclipse, para este ejemplo seleccionamos carrito de compras, tenga en cuenta que si la aplicación es de tipo maven, es necesario instalar los componentes necesarios.

Para generar el modelo KDM, de clic derecho sobre el proyecto, y diríjase a la opción Discovery/Discoverers/Discover KDM Source (si la opción no aparece valide la instalación de MoDisco).

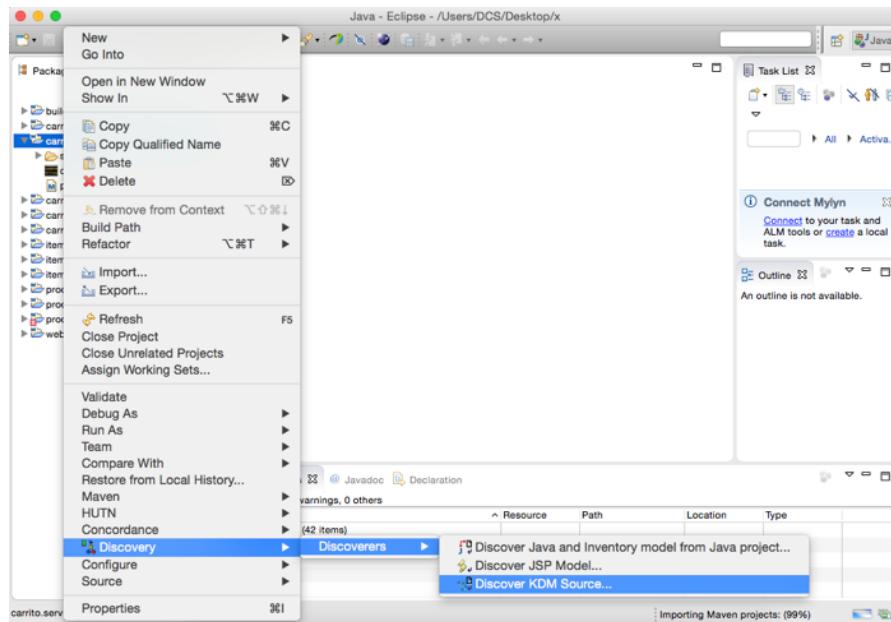


Figura 41.: Generación Modelo KDM con MoDisco

Seleccione la opción serializeTarget y cambie el valor a true y de clic en OK (ver figura 42), al finalizar el xmi se debió generar en la raíz del proyecto, como se observa en la figura 43.

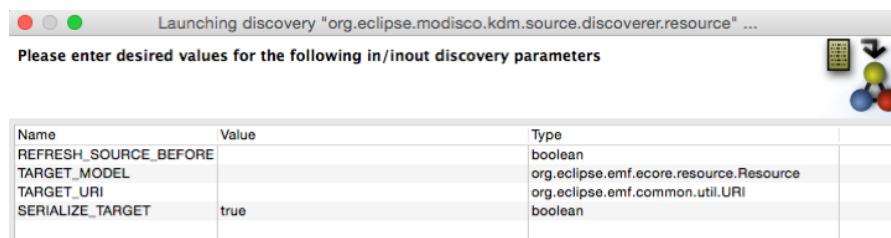


Figura 42.: Configuración sereilizaTarget

C.4 EJECUCIÓN DE LA TRANSFORMACIÓN EN ECLIPSE



Figura 43.: Generación Modelo KDM con MoDisco

El archivo xmi generado debe ser copiado en la ruta models en el proyecto miso4301-201520, para este caso fue renombrado como carrito.logic.kdm.xmi. (ver figura 44).

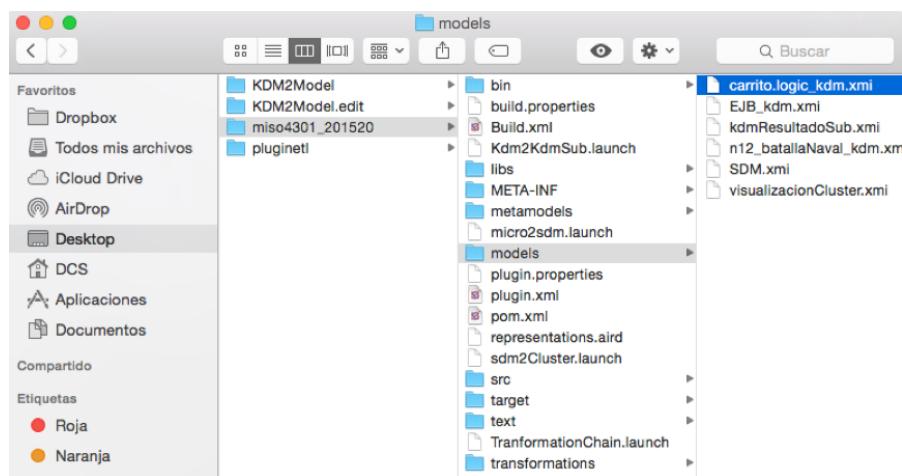


Figura 44.: Copia de modelo KDM

Teniendo el modelo conforme a KDM, se procede a importar en eclipse los proyectos del repositorio. Inicialmente debe importarse únicamente los proyectos pluginetl (Clusterer) y miso4301201520, como se muestra en la figura 45



Figura 45.: Importar Proyectos

Posteriormente se debe generar el «Code model» del metamodelo subkdm, para esto abra el archivo subKDM.genmodel que se en-

C.4 EJECUCIÓN DE LA TRANSFORMACIÓN EN ECLIPSE

cuenta en la carpeta metamodels (ver figura 46), de clic derecho y seleccione Generate Code Model (ver figura 47)

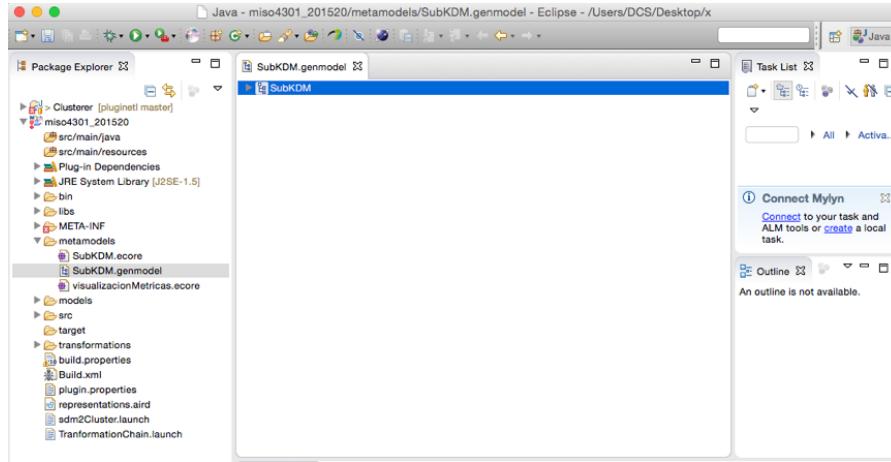


Figura 46.: Visualización archivo GemModel

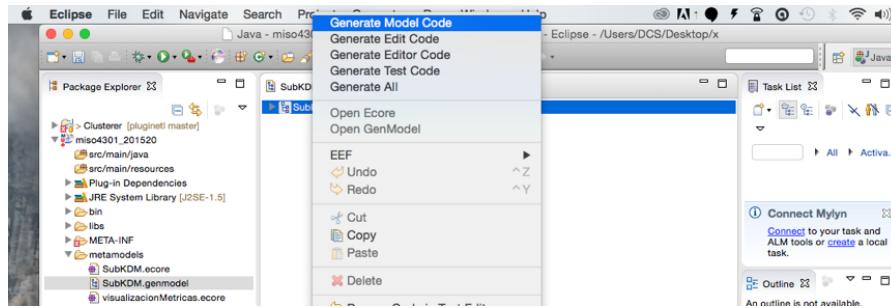


Figura 47.: Generacion Code Model

Realice el mismo proceso para generar el «Edit Code». Si por alguna razón tiene que actualizar el genmodel o tiene inconvenientes en la generación, valide que la configuración de las propiedades esté correcta, ver figuras 48 y 49. Al finalizar se deben crear dos nuevos proyectos: KDM2Model y KDM2Model edit.

Por último la herramienta ya está lista para ser ejecutada. Para ejecutar la cadena completa es necesario lanzar una segunda instancia de eclipse (es posible que para la configuración de la segunda instancia necesite asignar más memoria). Para esto de clic derecho sobre kdm2Model y ejecútelo como aplicación eclipse, ver figura 51

C.4 EJECUCIÓN DE LA TRANSFORMACIÓN EN ECLIPSE

Edit	
Color Providers	<input checked="" type="checkbox"/> false
Creation Commands	<input checked="" type="checkbox"/> true
Creation Icons	<input checked="" type="checkbox"/> true
Edit Directory	<input type="checkbox"/> /KDM2Model.edit/src
Edit Plug-in Class	<input type="checkbox"/> subkdm.kdmObjects.provider.SubKDMEditPlugin
Edit Plug-in ID	<input type="checkbox"/> KDM2Model.edit
Edit Plug-in Variables	<input type="checkbox"/>
Font Providers	<input checked="" type="checkbox"/> false
Optimized Has Children	<input checked="" type="checkbox"/> false
Provider Root Extends Class	<input type="checkbox"/>
Style Providers	<input checked="" type="checkbox"/> false
Table Providers	<input checked="" type="checkbox"/> false

Figura 48.: Configuración propiedades generación editModel

Model	
Array Accessors	<input checked="" type="checkbox"/> false
Binary Compatible Reflective Methods	<input checked="" type="checkbox"/> false
Class Name Pattern	<input type="checkbox"/>
Containment Proxies	<input checked="" type="checkbox"/>
Feature Delegation	<input type="checkbox"/> None
Generate Schema	<input checked="" type="checkbox"/> false
Interface Name Pattern	<input type="checkbox"/>
Minimal Reflective Methods	<input checked="" type="checkbox"/> true
Model Directory	<input type="checkbox"/> /KDM2Model/src
Model Plug-in Class	<input type="checkbox"/>
Model Plug-in ID	<input type="checkbox"/> KDM2Model
Model Plug-in Variables	<input type="checkbox"/>
Operation Reflection	<input checked="" type="checkbox"/> true
Suppress Containment	<input checked="" type="checkbox"/> false
Suppress EMF Metadata	<input checked="" type="checkbox"/> false
Suppress EMF Model Tags	<input checked="" type="checkbox"/> false
Suppress GenModel Annotations	<input checked="" type="checkbox"/> true
Suppress Interfaces	<input checked="" type="checkbox"/> false
Suppress Notifications	<input checked="" type="checkbox"/> false

Figura 49.: Configuración propiedades generación codeModel

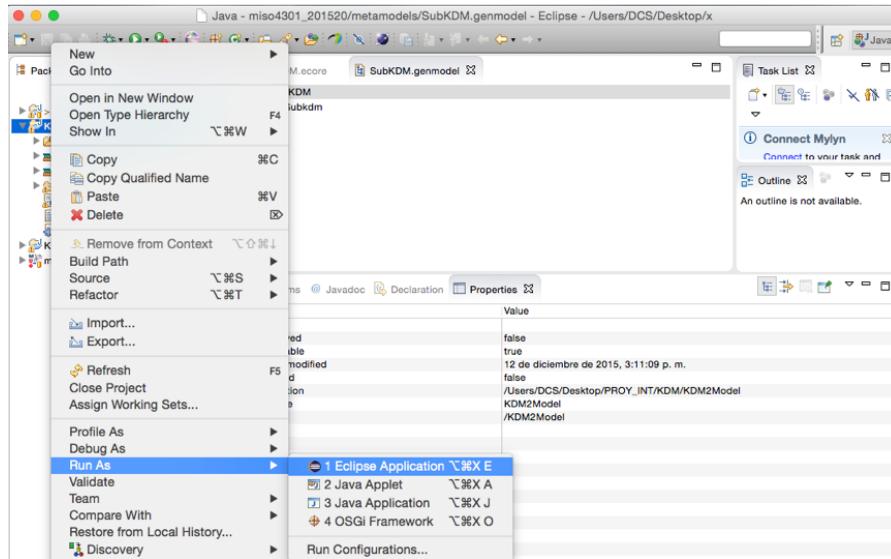


Figura 51.: Ejecución Segunda Instancia Eclipse

Estando en la segunda instancia importe el proyecto de las transformaciones "miso4301201520", y registre únicamente el.ecore "visua-

C.4 EJECUCIÓN DE LA TRANSFORMACIÓN EN ECLIPSE

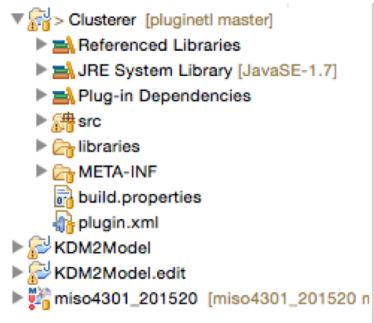


Figura 50.: Proyectos CodeModel

lizacionMetricas”, el.ecore correspondiente a subkdm es importante que NO se registre.

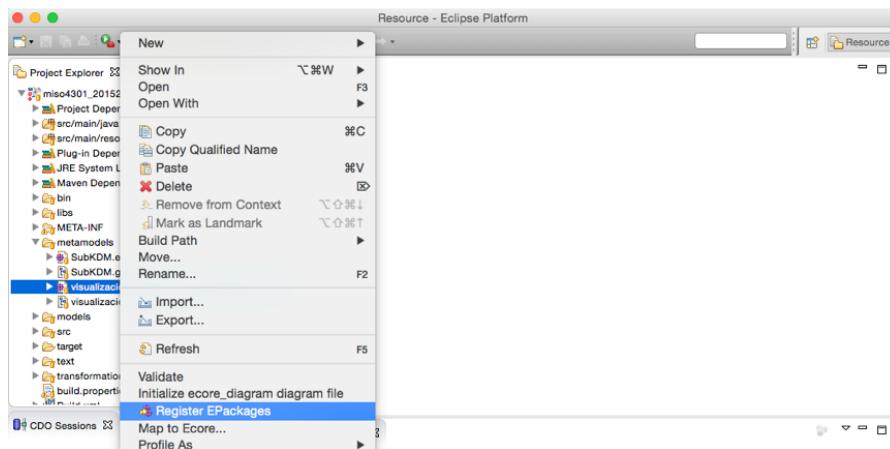


Figura 52.: Registrar Metamodelo Visualización

En la raíz del proyecto encontrará un archivo Build.xml, en éste está la ejecución de la cadena de transformación, en la parte superior se definen los xmi que se usaran (ver figura 53), y en la parte inferior se encuentra el llamado a cada transformación, valide que no estén comentareadas, de lo contrario pueden presentarse errores en la ejecución (ver figura 54)

```
<property name="Kdm" value="./models/carrito.logic_kdm.xmi"></property>
<property name="SubKdm" value="./models/gen/kdmResultadoSub.xmi"></property>
<property name="Cluster" value="./models/gen/representacion.xmi"></property>
<property name="Visualizacion" value="./models/gen/visualizacion.xmi"></property>
```

Figura 53.: Definición modelos Build.xml

C.4 EJECUCIÓN DE LA TRANSFORMACIÓN EN ECLIPSE

```
<!-- Main -->

<target name="main">
    <antcall target="Kdm2Kdm" inheritall="true"></antcall>
    <antcall target="micro2sdm"></antcall>
    <antcall target="sdm2Representacion"></antcall>
    <antcall target="sdm2Visualizacion"></antcall>
</target>
```

Figura 54.: Definición cadenas de transformación Build.xml

Para ejecutarlo seleccione el archivo TranfromationChain que se encuentra en la raíz del proyecto y de clic derecho y ejecutar TranfromationChain.

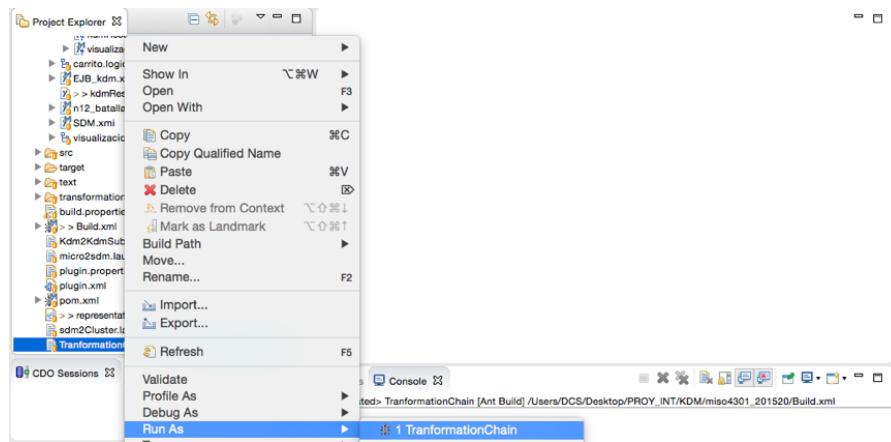


Figura 55.: Ejecución cadena de transformación

En la consola se podrá visualizar que transformación se esta ejecutando

```
TransformationChain [Ant Build] /Users/DCS/Desktop/PROY_INT/KDM/miso4301_201520/Build.xml
Buildfile: /Users/DCS/Desktop/PROY_INT/KDM/miso4301_201520/Build.xml

main:
loadModelsT1:

Kdm2Kdm:
[epsilon.etl - kdm2kdm.etl] kdm2kdm Transformation
```

Figura 56.: OutPut consola Ejecución

Cuando se inicie la ejecución de la segunda transformación, se solicitará al usuario un valor numérico que representará el peso que

C.4 EJECUCIÓN DE LA TRANSFORMACIÓN EN ECLIPSE

se dará a cada tipo de relación , actualmente se solicita el valor para relaciones de uso y contenencia (ver figura 57)

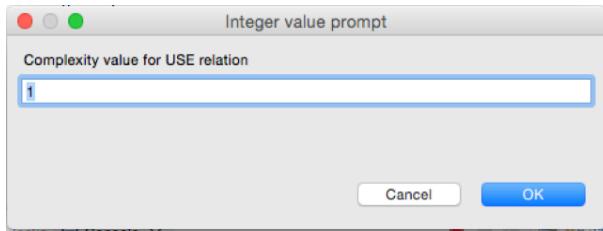


Figura 57.: Solicitud pesos de relaciones

Al finalizar la ejecución el modelo conforme a sub kdm para este caso “kdmResultadoSub.xmi” (ver figura 58) y dos modelos conforme a visualizacionMetricas “Visualizacioncluster.xmi” (ver figura 59) y Visualizacion.xmi (ver figura 60)

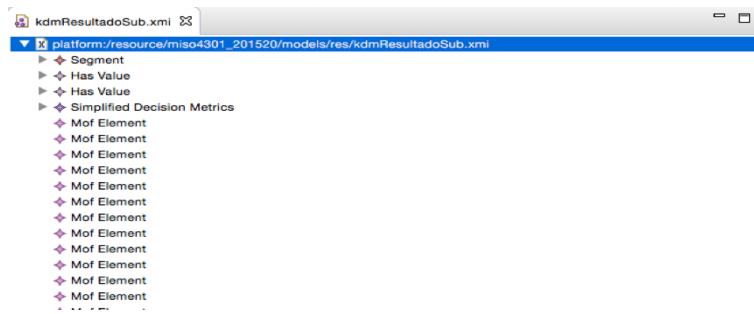


Figura 58.: Modelo conforme a subkdm

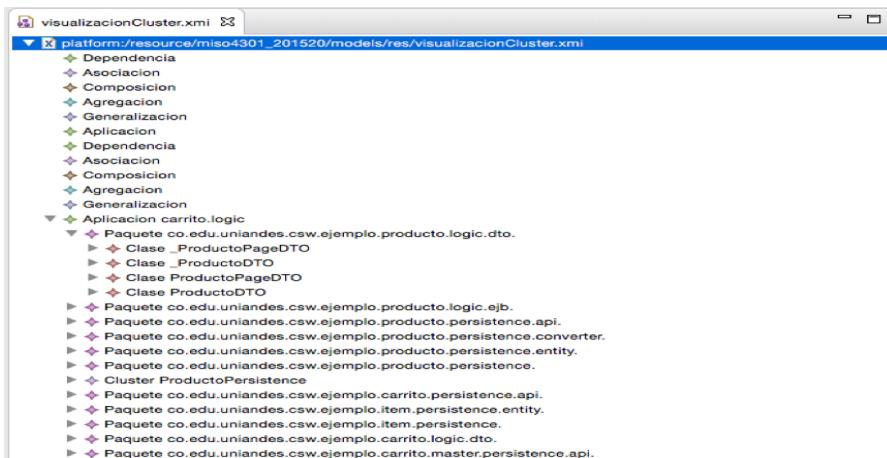


Figura 59.: Modelo conforme a VisualizaciónMetricas - Representación

C.5 VISUALIZACIÓN EN SIRIUS

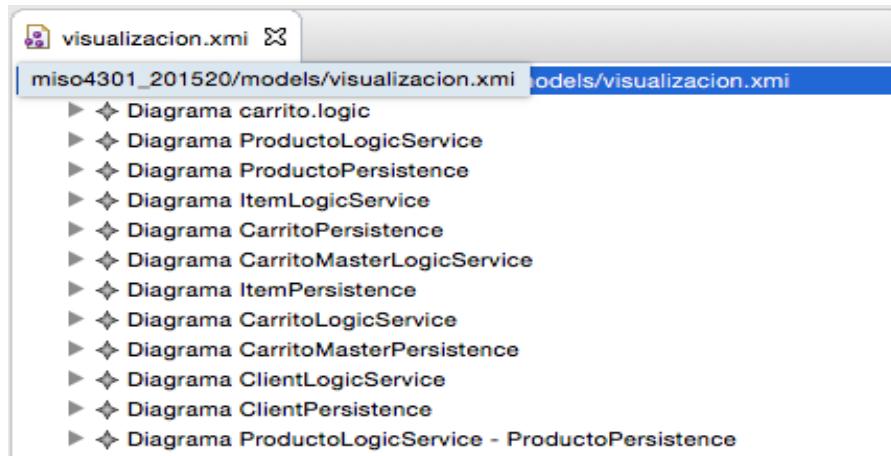


Figura 60.: Modelo conforme a VisualizaciónMetricas - Diagramación

C.5 VISUALIZACIÓN EN SIRIUS

De manera similar que en la generacion del metamodelo subkdm, se debe generar el «Code Model», «Edit Code» y «Editor Code» del metamodelo visualizacionMetricas; para esto se debe abrir el archivo visualizacionMetricas.genmodel que se encuentra en la carpeta metamodels, dar clic derecho y seleccionar "Generate Code Model", "Generate Edit Code" o "Generate Editor Code"(ver figura 61)

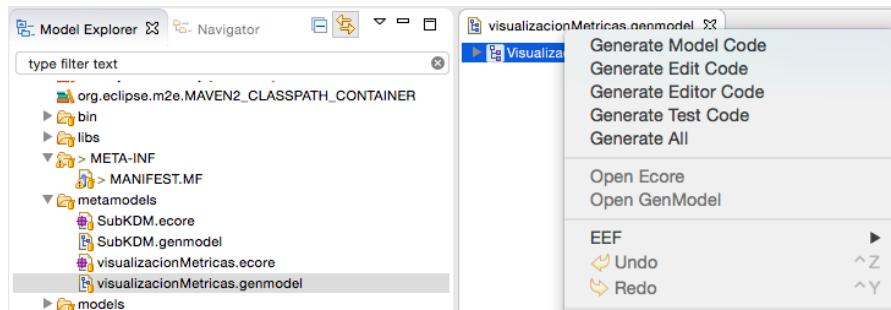


Figura 61.: Generacion Code Model

Al finalizar el proceso de generación se crean tres proyectos: VisualizacionMetricas3, VisualizacionMetricas3.edit y VisualizacionMetricas3.editor.

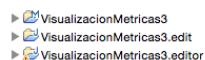


Figura 62.: Proyectos Generados

Se lanza una nueva instancia de eclipse sobre VisualizacionMetricas3 dando clic derecho y ejecutarlo como aplicación eclipse. En esta nueva instancia se importa el proyecto «VisualizacionMetricas3.instances» el cual contiene el *Viewpoint Specification* definido en el archivo visualizacionMetricas3.odesign

En este proyecto se copia la última transformacion (conforme al metamodelo visualizacionMetricas3) con la extención ".visualizacionmetricas3", y desde el Model Explorer podrá navegar por los elementos de modelo, (este comportamiento en eclipse se presenta, dado que se lanzó una nueva instancia sobre el proyecto VisualizacionMetricas3 generado).

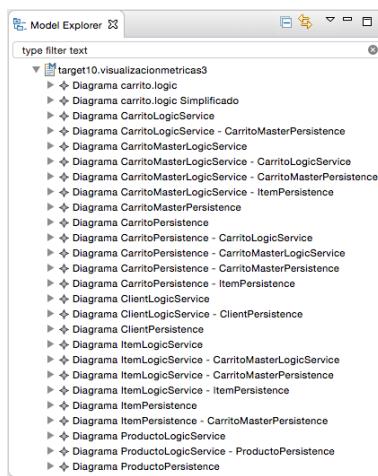


Figura 63.: Modelo abierto desde nueva instancia Eclipse

De clic derecho sobre el proyecto y seleccione "Select viewPoints"(es importante que se encuentra en la vista de Model Explorer de Sirius), posteriormente seleccione el viewPont "métricas". Sobre cualquier elemento del modelo, se puede dar clic derecho donde aparece los tipos de representaciones que Sirius tenga configurado para ese tipo de modelo (ver [visualizacion]).

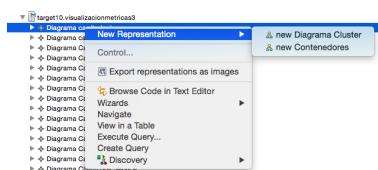


Figura 64.: Generando diagramas desde el Modelo con Sirius

Finalmente se tiene los diagramas generados

BIBLIOGRAFÍA

- [SW11] Robert Sedgewick y Kevin Wayne. *Algorithms, 4th Edition.* Addison-Wesley Professional, 2011.
- [Bax] Ira D. Baxter. *A Tool for Automating Software Quality Enhancement.* URL: <http://www.semanticdesigns.com/Products/DMS/WhyDMSForSoftwareQuality.pdf>.
- [CK] S.R. Chidamber y C.F. Kemerer. *A metrics suite for object oriented design.* URL: [http://ieeexplore.ieee.org/iel1/32/7320/00295895](http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=295895&url=http://ieeexplore.ieee.org/iel1/32/7320/00295895).
- [Foua] The Eclipse Foundation. *Epsilon.* URL: <http://www.eclipse.org/epsilon/>.
- [Foub] The Eclipse Foundation. *MoDisco.* URL: <https://eclipse.org/MoDisco/>.
- [Gar+] Kelly Garcés y col. «Visualization to Assist Structural Understanding of Oracle Forms Applications towards Modernization». En: () .
- [KDM] KDMAalytics. *KDM.* URL: <http://www.omg.org/technology/kdm/>.
- [Mer] Mauricio Verano Merino. «Migrating JEE applications to a microservice-based architecture deployed on the cloud».
- [OFW] Joshua O'Madadhain, Danyel Fisher y Scott White. *Java Universal Network/Graph Framework.* URL: <http://jung.sourceforge.net/index.html>.
- [OMG] OMG. *Structured Metrics Metamodel (SMM).* URL: <http://www.omg.org/spec/SMM/1.0/>.
- [Sán] Ana María García Sánchez. «Evaluación de métricas de calidad del software sobre un programa Java». URL: http://eprints.ucm.es/11487/1/Proyecto_Fin_de_M%C3%A1ster.pdf.
- [San] Edgar David Sandoval. «A tool to assist Structural understanding of Java JEE applications towards modernization».
- [Tec] Moose Technology. *Moose.* URL: <http://www.moosetechnology.org>.