Classification-based Mining of Reusable Components on Software Product Lines

M. Arias, A. De Renzis, A. Buccella, A. Flores and A. Cechich, *Member*, *IEEE.*

***Abstract*— Software Product Lines and Component-based systems can be combined to maximize reuse in a predictable and opportunistic manner. When a product line is built for a certain subdomain within a more generic domain, future needs from a closely subdomain may be fulfilled by mining the line's internal components to build a new product line. In this work, we present an approach to classify internal and external (third party) reusable components into a repository, by applying a K-Nearest Neighbors strategy, as a support for building new product lines. Natural language techniques and the WordNet lexical database are also used to process information from software components. We validate the approach with an experiment based in a dataset of external third-party components and reusable components from a product line that we built in the geographic subdomain of marine ecology.**

***Keywords*— Software Product Lines, Software Components, Software Reuse, K-Nearest Neighbors.**

I. Introducción

El REUSO de software se ha convertido en uno de los objetivos principales de los nuevos paradigmas de desarrollo de software. Entre los paradigmas de mayor adopción en la actualidad se encuentran las líneas de producto (SPL – Software Product Line) [1] y los sistemas basados en componentes (CBSD – Component-based Software Development) [2]. Ambos promueven fuertemente el reuso de software, aunque fueron concebidos sobre distintas bases a partir de las cuales se reconocen dos aspectos distintivos [3]. En primer lugar, las SPLs se crean con un objetivo de *predictibilidad* para el reuso de los activos que se diseñan explícitamente a tal efecto, contrario al *oportunismo* de la identificación de componentes reusables en CBSD. En segundo lugar, las líneas de productos son tratadas "*como un todo*", no como múltiples productos que se ven y se mantienen por separado, como ocurre en el desarrollo de sistemas basados en componentes. Estas diferencias no imposibilitan la combinación de ambos paradigmas, para aumentar sus beneficios, minimizar sus desventajas y alcanzar un reuso efectivo. Así surgieron las SPLs basadas en componentes cuyo precursor fue el enfoque KobrA [4] y otros enfoques posteriores tales como [5][6].

M. Arias, GIISCo, Universidad Nacional del Comahue, Neuquén, y CONICET, Argentina, maximiliano.arias@fi.uncoma.edu.ar

A. De Renzis, GIISCo, Universidad Nacional del Comahue, Neuquén, y CONICET, Argentina, alanderenzis@fi.uncoma.edu.ar

A. Buccella, GIISCo, Universidad Nacional del Comahue, Neuquén, y CONICET, Argentina, agustina.buccella@fi.uncoma.edu.ar,

A. Flores, GIISCo, Universidad Nacional del Comahue, Neuquén, y CONICET, Argentina, andres.flores@fi.uncoma.edu.ar,

A. Cechich, GIISCo, Universidad Nacional del Comahue, Neuquén, Argentina, alejandra.cechich@fi.uncoma.edu.ar

Siguiendo un enfoque combinado de SPL basada en componentes, el desarrollo de nuevos sistemas o conjuntos de sistemas debe ser adaptado para considerar las acciones pertinentes sobre dos tipos distinguibles de componentes: *externos* e *internos*. Los componentes *externos* (o de terceras partes) se reusan tal cual fueron diseñados y en general son vistos como cajas negras – usualmente sólo se cuenta con información de sus interfaces. Los componentes *internos* (o *in-house*) han sido creados dentro de la organización y se dispone de información más completa – incluyendo las interfaces y código, entre otros. Además, estos componentes podrían incluir variabilidades debido a que se han construido dentro de una SPL.

En trabajos previos [7][8], hemos desarrollado una SPL basada en componentes dentro del dominio genérico de información geográfica y en particular para el subdominio de ecología marina. Para ello, se aplicaron los estándares definidos por el Open Geospatial Consortium [9] y el Comité Técnico ISO/TC 2118 [10] – en particular se aplicó el estándar ISO 19119 [11] – para mejorar el reuso e interoperabilidad de los sistemas geográficos (GIS – Geographic Information System) que se derivan de la SPL. Además, los requerimientos concretos para esta SPL emanaron de dos institutos de investigación en biología marina de la zona patagónica argentina: IBMPAS [12] y CENPAT [13]. Dado que ha surgido interés de otros institutos de investigación (de subdominios geográficos distintos), se ha considerado el desarrollo de nuevas SPLs con la posibilidad de reusar los componentes internos de la SPL existente, y también identificar componentes externos con los cuales se establezca una fuerte vinculación.

Cualquier organización que contara con desarrollos propios podría reusar ambos tipos de componentes ante un nuevo desarrollo de una SPL, impactando positivamente en la productividad y los costos totales. Es decir, además de nuevos componentes creados para satisfacer las necesidades de una nueva SPL, se pueden reusar componentes externos y también aquellos creados internamente en desarrollos previos de la organización. De aquí, surge la necesidad de mantener un repositorio de componentes reusables como base esencial para una minería de componentes [14], brindando una fácil identificación y selección de componentes que soporten apropiadamente las funcionalidades requeridas para el desarrollo de nuevas SPLs.

En este sentido, en un trabajo previo [15] se definió un mecanismo para analizar nuevos requerimientos funcionales y determinar una relación con aspectos arquitectónicos de la ISO 19119 (que fue base para la SPL desarrollada), y desde allí establecer una vinculación a componentes reusables de un

repositorio. Por ello, en este trabajo presentamos un *proceso de clasificación de componentes*, que permite organizar de manera esquemática, la información almacenada en un repositorio, facilitando su utilización para la construcción de nuevas SPLs. Para ello se aplicó la estrategia de vecinos más cercanos (KNN – K-Nearest Neighbors) [16] considerando componentes previamente almacenados en el repositorio, lo cual habilita un aprendizaje sobre futuras clasificaciones, permitiendo que la minería de componentes provea resultados más certeros tanto al inferir una categoría predominante, como al identificar un componente reusable apropiado.

Este trabajo se organiza como sigue. La Sección II presenta el *proceso de clasificación de componentes* y en las Secciones III y IV se explican sus dos fases. La Sección V describe una evaluación experimental sobre la base de nuestra SPL para GIS. La Sección VI presenta trabajos relacionados. Luego se detallan las conclusiones y trabajos futuros.

II. Clasificación de componentes

Nuestro *proceso de clasificación de componentes* se basa en la estrategia de clasificación de KNN. Dado un conjunto de componentes preclasificados (C_b) y un nuevo componente (C_a), el mismo es clasificado considerando los k elementos más similares de acuerdo a una función de similitud definida. Para ello, fue necesario determinar qué información contenida en un componente es relevante para establecer su clasificación, cómo extraer la misma y de qué manera representar cada componente (Sección III).

Luego, para aplicar la estrategia de clasificación, fue necesario definir una función de similitud que recibe como entrada dos especificaciones de componentes y retorna un valor numérico que representa la similitud entre ambos (Sección IV). Además la clasificación se realiza en función de un *esquema de categorías* que debe ser definido para organizar la información en el repositorio.

La Fig. 1 muestra el *proceso de clasificación de componentes* que comprende dos fases: *preprocesamiento* y *clasificación*.

La fase de *preprocesamiento* se realiza sobre un nuevo componente a clasificar (C_a), e incluye 4 pasos: (1) extracción de nombres de operaciones, (2) separación en términos de estos nombres, (3) eliminación de stopwords, y (4) construcción del vector de términos.

Luego se procede a la fase de *clasificación* de C_a, que incluye 2 pasos: (5) clasificación KNN en función de un *esquema de categorías* y de los componentes preclasificados (C_b), y (6) selección de la categoría predominante para el componente C_a.

A. Ejemplo

Para ilustrar cada uno de los pasos del *proceso de clasificación de componentes* se definió un ejemplo sencillo, que involucra a dos componentes que proveen dos operaciones cada uno. La Fig. 2 muestra el componente a clasificar (C_a) denominado $MapEditor$, y un componente preclasificado (C_b) denominado $Visualizer$.

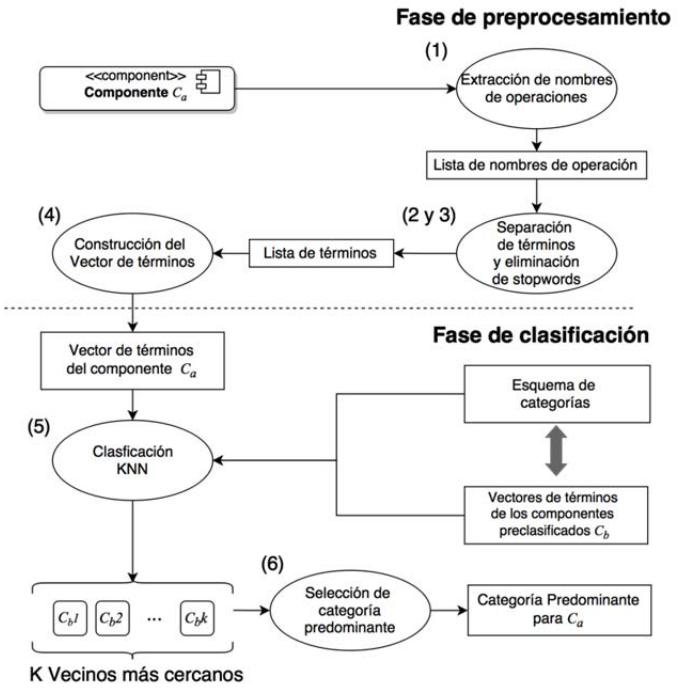

Figura 1. Proceso de clasificación de componentes.

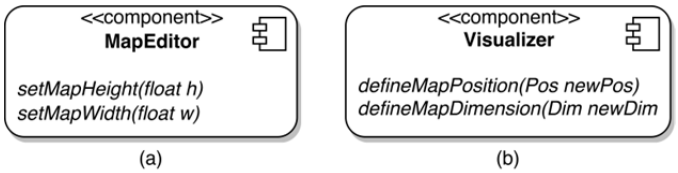

Figura 2. Componentes (a) a clasificar y (b) preclasificado.

III. Fase de Preprocesamiento

La estrategia de clasificación KNN se basa en el análisis de vectores de términos, como representación de la información que proveen los componentes. Para esto se analizó la descripción autocontenida en un componente software, para identificar cuál es el tipo de información más adecuado para representar los componentes y construir el vector de términos.

A. Extracción de nombres de operaciones:

Un componente software expone sus capacidades funcionales mediante interfaces públicas, que encapsulan una colección de operaciones [17][18]. De esta manera, la información relevante puede ser obtenida analizando los nombres de las operaciones contenidas en las interfaces. La interfaz ofrecida por un componente puede estar separada de su implementación, como por ejemplo un archivo WSDL para un Servicio Web; o puede estar empaquetada dentro del mismo, como por ejemplo un componente Java ensamblado en un archivo JAR [18].

El proceso de extracción de nombres de operaciones es independiente del tipo de componente, y el resultado de la extracción es el mismo, como así también el tratamiento posterior sobre los nombres de operaciones.

En el ejemplo de la Fig. 2, para los componentes $MapEditor$ y $Visualizer$, el proceso de extracción de nombres de operaciones resulta en las listas de nombres de operaciones, L_1

y L_2, respectivamente:

$L_1 = [setMapHeight, setMapWidth]$

$L_2 = [defineMapPosition, defineMapDimension]$

B. Separación de términos:

Las listas de nombres de operaciones deben ser procesadas para identificar cada uno de los términos (palabras) que componen a un nombre de operación. Los nombres de operaciones suelen respetar alguna convención en cuanto a su morfología, que facilitan la separación de términos. En este trabajo se adoptaron las reglas propuestas en la Tabla I.

TABLA I. REGLAS DE SEPARACIÓN DE TÉRMINOS.

Notación	Regla	Original	Resultado
Java Beans	Separar al cambiar a mayúscula	*getPointLatitude*	*get point latitude*
Símbolos Especiales	Separar cuando ocurre "_" o "-"	*create_histogram*	*create histogram*

Para separar en términos un nombre de operación, se analizan todos los términos potenciales, reconociendo secuencias de mayúsculas/minúsculas en los mismos. Para confirmar la separación de términos correcta se utiliza la base de datos léxica WordNet [19]. Esto además permite contemplar casos donde no se han respetado las convenciones de nombrado. Como resultado de la separación de términos se obtiene para cada componente una lista con todos los términos extraídos de cada una de sus operaciones [20].

En el ejemplo, para la lista de operaciones L_1 y L_2 generada en el paso anterior, el proceso de separación de términos generaría las siguientes listas de términos:

$SepTerm(L_1) = [set, map, height, set, map, width]$

$SepTerm(L_2) =$
$[define, map, position, define, map, dimension]$

C. Eliminación de stopwords:

Stopwords es el calificativo que reciben las palabras poco significativas – tales como artículos, pronombres y preposiciones – que son filtradas antes o después del procesamiento de datos (texto) en lenguaje natural [21][22]. La lista de stopwords está compuesta por las palabras/términos que son irrelevantes para determinar si un conjunto de términos es similar a otro, y ciertas palabras de cada dominio específico. En general las palabras de la lista de stopwords no ayudan a determinar la semántica de la operación (carecen de un peso semántico suficiente). En particular para este trabajo fueron agregados, a la lista de stopwords, términos pertenecientes a las convenciones de nombrado para nombres de operaciones. Concretamente en esta fase, son eliminados los stopwords de las listas de términos.

En el ejemplo, para la lista de términos separados $SepTerm(L_1)$, podemos observar que el término set se considera un stopword, pues pertenece a la convención de nombrado de JavaBeans [23], y es eliminado de dicha lista. Para $SepTerm(L_2)$ no se encuentra ningún stopword y no se modifica.

D. Construcción del vector de términos:

Sobre las listas obtenidas en las fases anteriores se realiza la construcción de un vector de términos para cada componente. Cada elemento de un vector se compone de dos atributos: el término y la cantidad de ocurrencias del mismo en el componente. En el vector de términos se incluyen sólo de aquellas palabras que denotan sustantivos, dado que por lo general, los verbos pertenecientes a nombres de operaciones son similares en todos los componentes sin importar la categoría a la que pertenezcan. Como resultado se obtiene para cada componente un vector de términos, que contiene sólo sustantivos junto con la cantidad de ocurrencias de los mismos, y está ordenado por esa cantidad de ocurrencias.

En el ejemplo, analizando las listas libres de stopwords, la construcción del vector de términos produce el vector V_1 para el componente $MapEditor$, y el vector V_2 para el componente $Visualizer$ – y en este caso se elimina el término $define$ (pues es un verbo):

$V_1 = [(map, 2), (width, 1), (height, 1)]$

$V_2 = [(map, 1), (position, 1), (dimension, 1)]$

IV. Fase de Clasificación

Luego de completar la fase de preprocesamiento se realiza la clasificación de los componentes (C_a) según un *esquema de categorías* establecido.

A. Clasificación KNN:

Se decidió utilizar la estrategia de clasificación KNN, en la cual se toma el vector de términos del componente a clasificar (C_a), y se lo compara contra todos los vectores de los componentes preclasificados (C_b), que se recuperan del repositorio. Considerando que los vectores de términos representan la funcionalidad que proveen los componentes (Sección III-A), la similitud de los vectores de términos implica una potencial equivalencia con respecto a un mismo contexto o dominio de aplicación, lo cual determina una correspondencia de los componentes a una misma categoría del repositorio. Para ello se definió una *función de similitud*, en la cual se utiliza WordNet como base semántica subyacente para el cálculo de similitud entre términos.

1) Integración de WordNet:

WordNet presenta una estructura de árbol con raíz en el nodo $\{entity\}$, donde los términos son agrupados en synsets (conjuntos de sinónimos) que representan el mismo concepto. Diversas relaciones conectan diferentes synsets, tales como hiperonimia/hiponimia, holonimia/meronimia y antonimia. Para calcular relaciones entre términos se necesita acceder a la información que proporciona WordNet, y para esto utilizamos una librería Java de amplio uso denominada JWNL [24].

2) Similitud entre componentes:

La *función de similitud* de componentes se forma de dos subfórmulas: una de ellas establece la similitud entre dos términos y la otra define la relevancia de cada término. Se corresponde con una fórmula de distancia clásica para KNN para clasificación de textos, pero considerando la estructura taxonómica de conceptos (en forma de árbol) que provee WordNet; y además teniendo en cuenta la reducción de ruido por términos irrelevantes, asignando un peso conveniente.

En la Fórmula 1 se define la función de similitud entre

términos, que hace uso de WordNet.

$$termSim(t_i, t_j) = \frac{MaxTreeDistance - length(t_i, t_j)}{MaxTreeDistance} \quad (1)$$

donde $MaxTreeDistance$ es el doble de la profundidad máxima del árbol léxico de WordNet – la distancia máxima entre dos términos en un árbol de profundidad fija. La función $length(t_i, t_j)$ es el camino más corto entre el término t_i y el término t_j en dicho árbol.

En el ejemplo, si se toma el término map del vector V_1, y se realizan las comparaciones con los términos map y $dimension$ del vector V_2, los valores de distancia son: $length(map, map) = 0$ y $length(map, dimension) = 7$. Considerando que actualmente la distancia máxima entre dos términos incluidos en el árbol léxico de WordNet es 32, obtenemos:

$$termSim(map, map) = \frac{32 - 0}{32} = 1$$

$$termSim(map, dimension) = \frac{32 - 7}{32} = 0{,}78$$

El rango de valores para $termSim$ es [0,1]; donde 1 representa un par de términos idénticos, mientras pares de términos distintos generan valores de similitud tendientes a 0.

En la Fórmula 2 se define la relevancia para un término t dentro de un componente C_a. Esto representa el peso de t dentro del conjunto total de términos en el componente C_a de acuerdo a su cantidad de ocurrencias.

$$rel(t \in C_a) = \frac{\#t}{\#C_a} \quad (2)$$

donde $\#t$ es la cantidad de ocurrencias del término t en C_a y $\#C_a$ es la cantidad de términos en C_a.

Si analizamos la relevancia del término map dentro de los vectores de términos V_1 y V_2, obtenemos:

$$rel(map \in V_1) = \frac{\#(map)}{\#V_1} = \frac{2}{4} = 0{,}5$$

$$rel(map \in V_2) = \frac{\#(map)}{\#V_2} = \frac{1}{3} = 0{,}33$$

El rango de valores para rel es (0,1]; donde 1 representa la mayor relevancia.

La Fórmula 3 define la similitud entre un par de componentes haciendo uso de las Fórmulas 1 y 2.

$$Similarity(C_a, C_b) = \sum_{i=0}^{N} \left[Max\big(termSim(C_a[i], C_b)\big) * \frac{rel(C_a[i]) + rel(C_b[max])}{2} \right] \quad (3)$$

donde N es la cantidad de términos de C_a, $C_a[i]$ es el i-ésimo término de C_a y $Max\big(termSim(C_a[i], C_b)\big)$ es el máximo valor de similitud de términos obtenido al comparar el término $C_a[i]$ contra todos los términos de C_b. $C_b[max]$ es el término de C_b con el que se obtuvo ese valor máximo de similitud.

Para el ejemplo propuesto y tomando en cuenta los valores calculados anteriormente, la similitud entre los componentes $MapEditor$ y $Visualizer$ se calcula aplicando la Fórmula 3:

$$Similarity(MapEditor, Visualizer) = \sum_{i=0}^{3} \left[Max\big(termSim(V_1[i], V_2)\big) * \frac{rel(V_1[i]) + rel(V_2[max])}{2} \right]$$

Calculando cada una de las partes de la sumatoria, obtenemos las siguientes expresiones:

$$termSim(map, map) * \frac{rel(map) + rel(map)}{2}$$

Pues el valor máximo de similitud para map del vector V_1 se presentó con el término map del vector V_2.

$$termSim(width, dimension) * \frac{rel(width) + rel(dimension)}{2}$$

Pues el valor máximo de similitud para $width$ del vector V_1 se presentó con el término $dimension$ del vector V_2.

$$termSim(height, position) * \frac{rel(height) + rel(position)}{2}$$

Pues el valor máximo de similitud para $height$ del vector V_1 se presentó con el término $position$ del vector V_2.

Finalmente, el valor de similitud entre ambos componentes será:

$$\left(1 * \frac{0{,}5 + 0{,}33}{2}\right) + \left(0{,}96 * \frac{0{,}25 + 0{,}33}{2}\right) + \left(0{,}84 * \frac{0{,}25 + 0{,}33}{2}\right) = 0{,}41 + 0{,}27 + 0{,}24 = 0{,}92$$

El rango de valores para *Similarity* es [0,1]; donde 1 representa el mayor valor de similitud, indicando la potencial correspondencia de ambos componentes a una categoría del repositorio.

B. Selección de la Categoría Predominante:

Para definir la categoría del componente a clasificar (C_a) se considera el *esquema de categorías* y los k componentes (C_b) más similares de acuerdo a los valores de similitud obtenidos. El componente C_a será clasificado con la categoría predominante en los k componentes más similares. Por ejemplo, si $k = 3$, se consideran los tres vecinos más similares y se observa la categoría de cada uno de ellos. Si dos de los tres vecinos más similares pertenecen a una cierta categoría A, entonces se determina que el componente a clasificar pertenece a esa categoría A.

1) Criterio de Desempate

Cuando no se observa una clara predominancia de una categoría por su cantidad de ocurrencias, se considera que el vecino más cercano será aquel componente preclasificado (C_b) que obtuvo el mayor valor de similitud. Entonces, su categoría se utiliza para la clasificación del componente C_a.

V. EXPERIMENTACIÓN EN SISTEMAS GEOGRÁFICOS

Para validar la aplicación del *proceso de clasificación de componentes*, presentamos un caso de estudio en el cual se ha construido un repositorio de componentes para el desarrollo de SPLs en GIS. Se considera el conjunto de componentes internos que forman nuestra SPL [7][8] para el subdominio de ecología marina. La arquitectura de referencia de la SPL adoptó la estructura multicapa del estándar ISO 19119, que se muestra en la Fig. 3. y cuenta con las siguientes capas:

- Interacción Humana (IH): se encarga de la visualización y de la interacción con el usuario.
- Comunicación (CO): se encarga de definir lenguajes y servicios para conectar las otras dos capas. Esta capa agrupa las siguientes subcapas:
  - Procesamiento del usuario: se encarga del análisis y

procesamiento de dominio geográfico.
- Procesamiento compartido: engloba todos aquellos servicios que realizan procesamiento de dominio general y no específicamente geográfico.

- Gestión de la Información (GI): se encarga del almacenamiento y procesamiento básico.

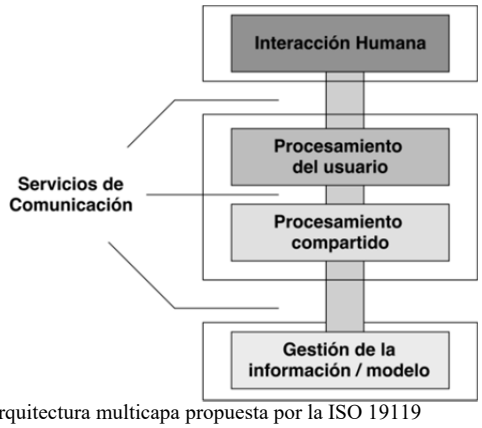

Figura 3. Arquitectura multicapa propuesta por la ISO 19119

Así, para la creación de nuevas SPLs se desea contar con un repositorio que permita reusar los componentes que sean necesarios. El repositorio mantiene un conjunto de metadatos que almacenan datos específicos de cada componente: el nombre del componente, sus interfaces, sus puntos variantes y/o variantes disponibles, y la categoría dentro de la arquitectura para la cual ese componente aplica.

Para mostrar el desempeño del *proceso de clasificación de componentes* hemos considerado componentes internos y externos, donde el mayor desafío surge con los componentes externos. Para los componentes internos de la SPL existente, se dispone del código fuente y una semántica precisa de sus interfaces y variabilidad definida, con lo cual su clasificación no resultó compleja y sus metadatos se completaron rápidamente.

El *proceso de clasificación de componentes* fue entonces aplicado sobre un conjunto de componentes geográficos externos, para ubicarlos en la categoría correcta dentro del repositorio y completar los metadatos propuestos.

A. Configuración del Experimento

1) Instanciación del esquema de categorías:

Las capas de la arquitectura que se propone en la ISO 19119 (Fig. 3), representan categorías primarias de una taxonomía de servicios geográficos. El estándar define categorías para la clasificación de servicios (funcionalidades) de manera que éstos puedan ser mapeados a la arquitectura propuesta. Las capas IH, CO y GI, conforman las categorías de mayor interés. Por lo tanto, en este caso de estudio, el *esquema de categorías* del proceso en la Fig. 1, se instancia con la arquitectura de referencia de la SPL y que se corresponde con el estándar ISO 19119 para GIS (Fig. 3).

2) Componentes preclasificados:

Los componentes internos de la SPL existente fueron preclasificados, para completar los metadatos definidos para el repositorio. Se identificaron 12 componentes geográficos internos, que fueron preclasificados según las capas de la arquitectura (Fig. 3). Se clasificaron: 6 componentes para la categoría IH, 3 componentes para la categoría CO, y 3 componentes para la categoría GI. Además, para estos componentes se obtuvieron y almacenaron todos los vectores de términos correspondientes.

3) Componentes a clasificar:

Como componentes para clasificar automáticamente se consideró un conjunto de 70 componentes extraídos de la herramienta GeoTools [25]. GeoTools es una librería Java de código abierto que provee métodos estándar para manipular datos geoespaciales, por ejemplo para implementar GIS. Además, la herramienta GeoTools fue desarrollada en conformidad con el estándar ISO 19119.

Para contrastar los resultados de la clasificación automática, se realizó paralelamente una clasificación manual de este conjunto de 70 componentes externos. La clasificación fue realizada por desarrolladores expertos pertenecientes al Grupo de Investigación en Ingeniería de Software del Comahue (GIISCO) [26].

4) Configuración del proceso de clasificación:

La implementación realizada para el *proceso de clasificación de componentes*, posee dos parámetros configurables que modifican su comportamiento. Por un lado, se debe configurar el *umbral de términos* relevantes de cada vector, es decir la cantidad de términos (ordenados por cantidad de ocurrencias) que se consideran para el proceso de clasificación. Esto se debe a que se observó que en general los componentes poseen algunos términos con el mínimo de ocurrencias y que no son representativos de la funcionalidad provista por los componentes – tal como, tipos de datos, formatos de archivos, conectividad de red, abreviaturas sin significado deducible. Ignorando estos términos se logra mejorar la eficiencia del proceso, sin afectar la calidad de los resultados de clasificación. Por otro lado, al aplicar la estrategia de clasificación KNN, se debe establecer el valor de k, que representa la cantidad de vecinos más similares a ser considerados para inferir la categoría predominante del componente a clasificar.

Para este caso de estudio, se realizó la siguiente configuración para estos valores. Por un lado, se utilizó un valor de *umbral de términos* $= 12$, que surgió de analizar los vectores de términos extraídos de los componentes preclasificados. Se observó que en promedio, cada componente posee entre 10 y 15 términos con una cantidad significativa de ocurrencias. Por otro lado, se consideró un valor de $k = 3$, lo cual está sujeto a la cantidad de componentes preclasificados. En este caso de estudio, se cuenta con un total de 12 componentes preclasificados, por lo tanto un valor mayor para k podría generar resultados que abarquen la mayoría de los componentes, sobrepasando las categorías afines.

B. Clasificación automática de componentes

Para cada uno de los componentes extraídos de la herramienta GeoTools, se realizó el *preprocesamiento* para obtener sus vectores de términos, los cuales fueron comparados contra los vectores almacenados para los

componentes preclasificados. Luego, fue registrada la categoría predominante para cada uno de los 70 componentes.

Los resultados obtenidos fueron comparados con las categorías correctas de cada componente según lo establecido por los expertos y las guías establecidas por la ISO 19199.

1) Resultados

La Tabla II muestra una matriz de confusión que representa los resultados obtenidos para los 70 componentes clasificados. Considerando aciertos/fallos en las categorías seleccionadas con respecto a las establecidas por los expertos, se observaron:

TABLA II. MATRIZ DE CONFUSIÓN DE LA CLASIFICACIÓN.

	IH	CO	GI	Total
IH	36	2	0	38
CO	4	15	6	25
GI	3	1	3	7

- 54 aciertos (77,1 %), que figuran en gris en la Tabla II.
- 16 fallos (22,9 %), según la siguiente distribución:
  - Categoría IH: 2 fallos (2,9 %).
  - Categoría CO: 10 fallos (14,3 %).
  - Categoría GI: 4 fallos (5,7 %).

2) Discusión:

Analizando los resultados obtenidos se pueden destacar dos aspectos principales. En primer lugar, una cantidad aceptable de componentes (77,1 %) fue bien clasificado, lo cual podría mejorar al contar con un conjunto más grande de componentes preclasificados para cada categoría. Por otro lado, la mayoría de los fallos de clasificación corresponden a componentes pertenecientes a la categoría Comunicación (CO). Esto se debe a que resulta complicado disponer de componentes suficientemente representativos para esta categoría. La misma engloba componentes de reglas de negocio del usuario, comunicación de las otras capas, procesamiento geográfico, entre otras. La diversidad de estos componentes se encuentra fuertemente relacionada con el dominio particular de cada aplicación, objetivos del sistema y aspectos técnicos de implementación (conectividad, lectura de formatos de archivo específicos, plataforma de desarrollo, etc).

3) Tiempo de Ejecución:

Para el experimento se utilizó el siguiente equipo: PC AMD FX-6300 Six-Core, 16GB RAM, HDD SATA2 y S.O. Linux 3.13.0 X64.

El proceso automatizado de clasificación (incluyendo las fases de preprocesamiento y clasificación) de los 70 componentes no superó los 3 segundos de tiempo de ejecución. En contraste, la validación manual del experimento realizada por los expertos alcanzó un tiempo aproximado de 16 horas (sumando el tiempo invertido por cada experto).

VI. Trabajos Relacionados

En [27] se describe un diseño preliminar de un sistema de repositorios cuyo objetivo es taxonomizar y clasificar componentes a partir de GIS de código abierto. El sistema estaría compuesto por cinco componentes: clasificador, identificador, constructor, manejador de metadatos y comparador. Para la clasificación de los GIS, se basará en el código de los mismos para organizarlos según las categorías indicadas por el usuario. Aquellos sistemas que no se correspondan con ninguna de las categorías serían clasificados como indefinidos, permitiendo que el usuario indique una categoría. Para el reuso de componentes el usuario ingresaría una plantilla de requerimientos y el sistema identificaría código reusable que cumpla con las necesidades propuestas. Si la correspondencia no fuera exacta o abarcara varios sistemas, el sistema de repositorios debería ser capaz de ensamblar las secciones código y formar un nuevo componente.

Por otro lado, técnicas muy reconocidas en el área de minería de datos han sido utilizadas en el contexto de CBSD. Por ejemplo, en [28] se propone una estrategia para mejorar el reuso efectivo de software, identificando acoplamiento de componentes mediante clustering [29]. El enfoque plantea el uso de métricas conocidas, como cohesión y acoplamiento, para determinar las categorías de reusabilidad. La agrupación por categorías permite que los desarrolladores puedan identificar aquellos componentes con el menor costo de integración. Los autores sólo presentan un marco teórico y proponen una futura implementación del mismo.

Finalmente, en [30] se propone agrupar componentes de software y formar un subconjunto de librerías para un repositorio disponible. Estos grupos ayudarían a elegir un componente requerido de forma rápida y eficiente. Se definió una función de similitud que permite agrupar componentes de software y estimar el costo de un nuevo proyecto. La función de similitud toma un conjunto de características predefinidas y evalúa su presencia y distribución en cada uno de los componentes. El enfoque está definido sólo a nivel teórico.

VII. Conclusiones

En este trabajo hemos presentado un *proceso de clasificación de componentes* que permite organizar en forma esquemática la información de un repositorio, como soporte a la minería de componentes software en la construcción de una SPL. Se ha aplicado una estrategia de clasificación KNN para inferir una categoría predominante para un nuevo componente a clasificar. Además se aplicaron técnicas de procesamiento de lenguaje natural junto con WordNet, para extraer y procesar la información relevante de los componentes reusables para construir la representación de vectores de términos. Se evaluó el rendimiento del enfoque mediante un experimento que toma como base una SPL basada en componentes que hemos desarrollado previamente en el subdominio geográfico de ecología marina.

Actualmente consideramos extender la información de los metadatos del repositorio, determinando los servicios específicos que proveen los componentes, en función de la taxonomía de servicios GIS que define el estándar ISO 19119. Además, en un trabajo previo [8] se realizó una extensión de la taxonomía, agregando servicios específicos del subdominio oceanográfico y su subdominio de ecología marina. Tal información de servicios podría utilizarse para enriquecer los vectores de términos en componentes preclasificados, para incrementar la precisión en la clasificación de componentes reusables y permitiendo que su posterior recuperación pueda ser por categoría y también por servicio GIS.

AGRADECIMIENTOS

Este trabajo se enmarca en el proyecto UNCo–Reuse(04-F001) de la Universidad Nacional del Comahue.

REFERENCIAS

[1] K. Pohl, G. Bockle, and F. Linden. Software product line engineering: foundations, principles, and techniques. Springer, 2005.
[2] G. Heineman, W. Councill. Component-based Software Engineering: Putting the Pieces Together. Addison-Wesley, 2001.
[3] S. Jansen, S. Brinkkemper, I. Hunink, and C. Demir. Pragmatic and Opportunistic Reuse in Innovative Start-Up Companies. IEEE Software, 25(6):42-49, 2008.
[4] C. Atkinson, J. Bayer, and D. Muthig. Component-based product line development: the KobrA approach. 1st Conference on Software Product Lines. Kluwer Academic Publishers, pp. 289-309, 2000.
[5] M. Körner, S. Herold, and A. Rausch. Composition of applications based on software product lines using architecture fragments and component sets. WICSA Companion Volume, Nº 13, ACM Press, 2014.
[6] M. Caporuscio, H. Muccini, P. Pelliccione, E. Di Nisio. Rapid System Development Via Product Line Architecture Implementation. Chapter on Rapid Integration of Software Engineering Techniques. Springer LNCS 3943, pp. 18-33, 2006.
[7] A. Buccella, A. Cechich, M. Arias, M. Pol'la, S. Doldan, and E. Morsan. Towards systematic software reuse of gis: Insights from a case study. Computers & Geosciences, 54(0):9–20, 2013.
[8] A. Buccella, A. Cechich, M. Pol'la, M. Arias, S. Doldan, and E. Morsan. Marine ecology service reuse through taxonomy-oriented SPL development. Computers & Geosciences, 73(0):108–121, 2014.
[9] OGC (Open Geospatial Consortium) [Online]. Available: http://www.opengeospatial.org/
[10] ISO/TC 211 - Geographic information/Geomatics [Online]. Available: http://www.isotc211.org
[11] ISO 19119:2005 - Geographic information/Services [Online]. Available: http://www.iso.org/iso/catalogue_detail.htm?csnumber=39890
[12] IBMPAS: Research Institute of Marine Biology and Fisheries Almirante Storni [Online]. Available: http://www.ibmpas.org/
[13] National Patagonian Center CENPAT-CONICET [Online]. Available: http://www.cenpat.edu.ar/
[14] V. Subedha and S. Sridhar. A Systematic Review of Reusability Assessment Model and Related Approach for Reusable Component Mining. Journal of Computer Applications, 5(2):55-59, 2012.
[15] M. Arias, A. De Renzis, A. Buccella, A. Cechich, and A. Flores. Búsqueda de servicios para asistir en el desarrollo de una Línea de Productos de Software. ASSE, 16th Argentine Symposioum on Software Engineering, during 44th JAIIO. pp. 145-158, Rosario, Argentina.
[16] D. Gomez, F. Prieto and M. Guzman. Nearest Neighbors by Adaptive Simulated Annealing. IEEE Latin America Transactions, 13(7):2398-2404, 2015.
[17] C. Szyperski. Component Software: Beyond Object-Oriented Programming. 2nd ed. Addison-Wesley Professional, 2002.
[18] L. Kung-Kiu, and W. Zheng. Software Component Models. IEEE Transactions on Software Engineering, 33(10):709–724, 2007.
[19] G. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. Miller. Introduction to WordNet: An On-line Lexical Database. International Journal of Lexicography, 3(4):235-244, 1990.
[20] A. De Renzis, M. Garriga, A. Flores, A. Zunino, and A. Cechich. Semantic-structural Assessment Scheme for Integrability in Service-oriented Applications. CLEI, XL Latin American Computing Conference, IEEE Xplore, pp. 1-11, 2014.
[21] A. Casamayor, D. Godoy, and M. Campo. Mining Textual Requirements to Assist Architectural Software Design: A State of the Art Review. Artificial Intelligence Review, 38(3):173-191, Springer. 2012.
[22] R. Rocha. Adaptive Technology Applied to Natural Language Processing. IEEE Latin America Transactions, 5(7):544-551, 2007.
[23] JavaBeans Conventions (Java in a Nutshell) [Online]. Available: http://docstore.mik.ua/orelly/java-ent/jnut/ch06_02.htm
[24] Java WordNet Library API: JWNL v1.1 [Online]. Available: https://web.stanford.edu/class/cs276a/projects/docs/jwnl/javadoc/
[25] GeoTools: The Open Source Java GIS Toolkit [Online]. Available: http://geotools.org/
[26] GIISCo: Research Group on Software Engineering of Comahue [Online]. Available: http://giisco.uncoma.edu.ar
[27] A. Naseer, B. Alkazemi, and H. Aldoobi. A Repository System for Open-source GIS Software Components: A General Purpose Educational Environment. 5th International Conference on Computer Science Education: Innovation and Technology, 2014.
[28] J. Basha, and C. Mohan: A Strategy to identify components using clustering approach for component Reusability, WSE,CS & IT 06, pp.397-406, 2012.
[29] A. Reyes, A. Garcia, Y. Mué. System for Processing and Analysis of Information Using Clustering Technique. IEEE Latin America Transactions, 12(2):364-371, 2014.
[30] C. Srinivasa, V. Radhakrishnab, and C. GuruRao. Clustering and Classification of Software Component for Efficient Component Retrieval and Building Component Reuse Libraries. 2nd ITQM, Procedia Computer Science, 31:1044–1050, 2014.

Maximiliano Arias received a BS degree in Computer Science from the National University of Comahue, Argentina (2014). He is currently a fellow of CONICET to pursue a PhD degree in computer science from UNICEN, Argentina. He is member of GIISCo and Teacher Assistant at Informatics Faculty, National University of Comahue. His research interests include software engineering, GIS, SPL and component-based systems.

Alan De Renzis received a BS degree in Computer Science from the National University of Comahue, Argentina (2014). He is currently a fellow of CONICET to pursue a PhD degree in computer science from UNICEN, Argentina. He is member of GIISCo and Teacher Assistant at Informatics Faculty, National University of Comahue. His research interests include software engineering, SOC, component-based systems and software testing.

Agustina Buccella received MS and PhD degrees in Computer Science from the National University of South, Argentina (2005, 2009). She is Research Assistant at CONICET, Argentina. She is also member of GIISCo and Adjunct Professor at the Informatics Faculty, National University of Comahue. Her research interests include software engineering, GIS, SPL and component-based systems.

Andrés Flores received MS degree in Computer Science from the National University of South, Argentina (2005); and PhD degree in Informatics from the University of Castilla-La Mancha, Spain (2009). He is Research Assistant at CONICET, Argentina. He is also member of GIISCo and Adjunct Professor at the Informatics Faculty, National University of Comahue. His research interests include software engineering, SOC, component-based systems and software testing.

Alejandra Cechich received M.S. degree in Computer Science from National University of South, Argentina (2001); and Ph.D. degree in Informatics from University of Castilla-La Mancha, Spain (2005). She is head of GIISCo and Professor at Informatics Faculty, National University of Comahue. Her research interests are Software Engineering, Software Quality, Software Architecture and component-based systems.