Universidad de las Ciencias Informáticas

Facultad 1

Título: Subsistema de Detección de Roles para el Motor de Categorización Inteligente de Contenido en Correo Electrónico.

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas.

Autores: Yaritza Rodriguez Garcia
Jorge Enrique Miralles Betancourt

Tutor: Karel Antonio Verdecia Ortiz
Co-Tutor: Susel Vázquez Acuña
Tutor Consultante: Ernesto Rodriguez Ortiz

Ciudad de la Habana. 14 de junio de 2012

"Año 54 de la Revolución"

Declaración de autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas (UCI) los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste se firma la presente a los ____ días del mes de _____ del año _____.

_____ _____

Yaritza Rodriguez Garcia Jorge Enrique Miralles Betancourt

_____ _____

Licenciado en Cibernética y Matemática
Karel Antonio Verdecia Ortiz

Ingeniero Informático
Susel Vázquez Acuña

_____

Ingeniero en Ciencias Informáticas
Ernesto Rodriguez Ortiz

“Porque nadie puede saber por ti. Nadie puede crecer por ti. Nadie puede buscar por ti. Nadie puede hacer por ti lo que tú mismo debes hacer. La existencia no admite representantes”.

Jorge Bucay

Agradecimientos

Yaritza Rodriguez Garcia

Agradezco a mi papá Fidel, por todo su amor, por el apoyo, el ejemplo y por saberme guiar siempre por el camino correcto, papi eres lo mejor que me ha pasado en la vida, dios no me pudo enviar un mejor padre. A mi mamá Maritza por haberme traído al mundo, dándome la posibilidad de existir y poder estar hoy aquí. A mi hermanito Yariel por ser el mayor tesoro que tengo en la vida. A mi Papo Migue por haberme apoyado siempre que lo necesité y por su ayuda para poder terminar mi carrera. A mi compañero de tesis Miralles porque supo entenderme y complacerme aunque no lo dejaba vivir. A mis tutores, en especial a Ernesto muchísimas gracias, por el apoyo que nos diste y por ayudarnos siempre que lo necesitamos, sin usted este sueño no fuera realidad. A mis amigas y hermanas Mirelis, Zamay y Zulema que le debo mucho en mi carrera y en la vida, las quiero. A mis compañeros de estudio Daimara, Anigleidis, Yuni y Ailena por toda la ayuda que me han prestado. A mi grupo de primer año, ustedes son lo mejor de la UCI. A todo el que de una forma u otra ha hecho posible la realización de este sueño.

Jorge Enrique Miralles Betancourt

Primero quiero agradecer a mis padres y a mi hermano y a toda mi familia, que siempre me han apoyado y son la razón de que yo esté aquí hoy. A dos personas que no pudieron estar aquí hoy pero que han estado ahí para mí siempre dos personas que considero mi familia Carlos y Miriam. A Mario que estuvo ahí cuando realmente hizo falta. A mi gente, mis amigos del Edificio 16 Jorgito, Janier, las dos Elizabeth, Betty, Alberto por todos los días y las noches que pasé allí, ustedes son el grupo de personas más extraordinario con el que he tenido el placer de compartir, me los llevo conmigo para donde quiera que vaya. A mis amigos y equipo del aula Eddy, Aylena, Mayra, Amanda, Guirola, Alberto por todo lo que hemos pasado juntos. A mi compañera de tesis Yaritza, por halarme las orejas para que trabajara y por soportarme todo este tiempo. Sin ti no lo hubiéramos logrado. A mis tutores por el esfuerzo y el tiempo que nos dedicaron, en especial a Ernesto que nos guió hasta aquí y nos ayudó siempre que lo necesitamos. A todos los que compartieron conmigo en estos cinco años y a los que de una forma u otra colaboraron con la realización de este trabajo.

Dedicatoria

Yaritza Rodriguez Garcia

A mi papá Fidel, mi mamá Maritza, mi hermano Yariel los adoro y los adoraré por siempre.

Jorge Enrique Miralles Betancourt

A mi mamá y a mi papá, no soy nadie sin ustedes.

Resumen

Existen diferentes tipos de redes sociales presentes en Internet; dentro de estas, las redes de correos electrónicos generan un gran volumen de datos, que no resultan de mucha utilidad sin antes pasar por un proceso de análisis. Realizar dicho proceso de forma manual resulta prácticamente imposible, por lo que sería útil una herramienta capaz de extraer la información significativa de estos datos. Con este fin surge el proyecto MOCICE, el cual realizaría de forma automatizada el análisis de los datos generados por la interacción de los usuarios en una red de correo electrónico. Durante la realización del presente trabajo de diploma se desarrolló el Subsistema de Detección de Roles, el cual permite a MOCICE analizar el comportamiento de los usuarios de una red de correo electrónico y a partir de estos identificar los roles que desempeñan. Para realizar el análisis de los datos se implementó el algoritmo Triadic Census descrito por Hans Hummell y Wolfgan Sodeur [1], algoritmo que además de ser rápido, presenta baja complejidad computacional. Durante el período de pruebas se realizaron varios experimentos, los que muestran una mejora significativa en el desempeño del algoritmo sobre los implementados en los sistemas analizados durante la investigación.

Palabras clave: red social, correo electrónico y detección de roles.

Índice general

Introducción **1**

1. Detección de roles **5**

1.1. Conceptos fundamentales 5

1.1.1. Red Social 5

1.1.2. Análisis de Redes Sociales 6

1.1.3. Roles 6

1.2. Trabajos similares 6

1.2.1. InFlow 7

1.2.2. GeoSocialApp 7

1.2.3. UCINET 7

1.2.4. Pajek 8

1.3. Equivalencia 8

1.3.1. Equivalencia estructural 9

1.3.2. Equivalencia automórfica 10

1.3.3. Equivalencia regular 11

1.3.4. Otras equivalencias 12

1.4. Modelado de bloques 13

1.5. Algoritmos para la detección de roles 14

1.5.1. CONCOR 14

1.5.2. Maxsim 15

1.5.3. REGE 15

1.5.4. RART 16

1.5.5. Triad Census 16

1.6. Tecnologías 17

1.6.1. Metodología 17

1.6.2. Lenguajes de programación 19

1.6.3. Entornos de Desarrollo Integrado 19

1.6.4. Herramientas de la Base de Datos 20

1.6.5. Herramientas de modelado 21
1.6.6. Herramientas para la revisión de código 22
1.6.7. Otras herramientas 22
Conclusiones parciales 23

2. Propuesta del Subsistema de Detección de Roles 24
2.1. Modelo de dominio 24
2.2. Descripción de la propuesta de solución 25
2.2.1. Ciclo de ejecución del Subsistema de Detección de Roles 25
2.2.2. Descripción del algoritmo Triad Census 27
2.3. Captura de requisitos 31
2.3.1. Requisitos funcionales 32
2.3.2. Requisitos no funcionales 32
2.4. Historias de Usuario 33
2.4.1. Historias de Usuario (HU) Detectar roles 33
2.4.2. HU Registrar actores según rol 34
2.4.3. HU Establecer comunicación con el Controlador de MOCICE 34
2.4.4. HU Registrar trazas de ejecución del Subsistema de Detección de Roles 34
2.4.5. HU Leer fichero de configuración 35
2.5. Diseño 35
2.5.1. Arquitectura propuesta 36
2.5.2. Patrones de diseño 37
2.5.3. Diagrama de paquetes de clases del diseño 39
2.5.4. Diagrama de clases del diseño 40
2.5.5. Descripción de las clases del diseño 41
2.5.6. Diseño de la Base de Datos 46
Conclusiones parciales 49

3. Implementación y validación del Subsistema de Detección de Roles 50
3.1. Modelo de implementación 50
3.1.1. Plan de Entrega 50
3.1.2. Diagrama de componentes 52
3.1.3. Diagrama de despliegue 53

3.2. Revisión de código 53
3.2.1. Revisión con la herramienta *Rough Auditing Tool for Security* (RATS) 54
3.2.2. Revisión con la herramienta Valgrind 55
3.3. Pruebas de *software* 55
3.3.1. Pruebas de funcionamiento 56
3.3.2. Pruebas de integración 59
3.3.3. Pruebas de rendimiento y de estrés 61
Conclusiones parciales 64

Conclusiones **65**

Recomendaciones **66**

Lista de acrónimos **67**

Referencias bibliográficas **72**

Bibliografía **75**

A. Lista de Reserva del Producto **76**

B. Archivo de configuración **79**

Introducción

Desde su surgimiento, el hombre sintió la necesidad de agruparse con sus semejantes formando sociedades, las cuales representan grupos de personas que comparten un territorio y una cultura definida. La sociología llevó el término un poco más lejos, alegando que sociedad engloba además, la estructura social y las interacciones del grupo. Una estructura social se define como los patrones de comportamiento y las relaciones, relativamente duraderas, dentro de una sociedad [2]. Por lo tanto, una sociedad no es solo un grupo de personas y su cultura, sino que también incluye el comportamiento y las relaciones existentes entre las personas y las instituciones dentro del grupo.

Dentro de una sociedad, según la Teoría de Roles dada por la Sociología [3], la conducta humana está guiada por las expectativas que posee tanto el individuo como el resto de las personas. Estas expectativas corresponden a diferentes roles desempeñados por los individuos en su vida diaria. Por ejemplo, la mayoría de las personas conocen qué esperar de alguien con el rol de secretaria, lo que puede incluir: contestar el teléfono, redactar notas, organizar citas, etc. Estas acciones no serían esperadas del jefe de la secretaria. O sea, un rol consiste en un conjunto de reglas o normas que funcionan como guía del comportamiento. Los roles especifican qué metas se deben perseguir, qué tareas deben ser cumplidas y qué conducta se requiere en un escenario o situación específica.

La Teoría de Roles sostiene que una parte sustancial del comportamiento social diario está formado por un grupo de personas desempeñando un rol específico, como dijera Henry L. Tischler: *"Las personas no interactúan unas con otras como seres anónimos. Se relacionan en contextos y ambientes específicos con propósitos definidos. Sus relaciones involucran comportamientos asociados con estatus definidos y roles particulares. Estos roles y posiciones sociales ayudan a crear patrones de nuestras interacciones sociales y proveen predictibilidad"* [4]. Esta teoría es de hecho predictiva, lo que implica que si se posee información sobre las expectativas de un rol, es posible predecir una porción significativa del comportamiento de la persona que lo desempeñe.

El desarrollo de las tecnologías de la comunicación y la informática posibilitó que las interacciones sociales sobrepasaran las distancias, Internet se convirtió en una base para la creación de nuevas y diversas

redes sociales, más complejas y dinámicas que las existentes hasta el momento. En Internet existen varios tipos de redes sociales, unas de las más comunes son aquellas creadas a partir del uso del correo electrónico, estas redes son muy ricas en cuanto a información se refiere. A partir de estas se puede hacer un análisis detallado de las interacciones sociales y los roles que juegan las personas que pertenecen a dichas redes. Es aquí donde entra a desempeñar un papel fundamental el análisis de redes sociales y dentro de esta el reconocimiento de roles, cuyo objetivo es la detección de las características de los papeles que desempeñan los actores, con el fin de predecir el comportamiento de estos, haciendo uso de la capacidad predictiva de la Teoría de Roles.

Dado el gran volumen de información que se genera en una red de correos electrónicos (20 millones al día en Cuba) resulta casi imposible analizar de forma rápida y sencilla las interacciones sociales que ocurren en la misma, como la detección de violaciones de uso, los temas de los mensajes, la dinámica de los grupos de trabajo presentes en la red y el desempeño de los líderes, etc. Para llevar a cabo un análisis eficiente en redes de correos electrónicos se crea el proyecto Motor de Categorización Inteligente de Contenido para Correo Electrónico (MOCICE) el cual realizaría de forma automatizada la detección de tópicos, comunidades y roles. El presente trabajo de diploma surge debido a que en estos momentos este proyecto no cuenta con el Subsistema de Detección de Roles. Proceso necesario para determinar los comportamientos de los usuarios, a partir de los cuales se pueden identificar aquellos que son propensos a cometer violaciones, facilitando el control de las mismas. Ayuda también a detectar anomalías en los comportamientos de los usuarios de un grupo determinado, que al corregirse podrían significar una mejoría en el desempeño de dicho grupo y puede ser utilizado para medir y comparar desempeños individuales, con el objetivo de determinar quién o quiénes podrían realizar mejor una tarea específica.

Surge entonces como problema a resolver: ¿Cómo detectar los roles de los usuarios en colecciones de mensajes de correo electrónico en MOCICE?

Se enmarcó el objeto de estudio en el Análisis de Redes Sociales.

El campo de acción lo constituye la detección de roles de usuarios en colecciones de correo electrónico.

En el presente trabajo se plantea como objetivo general: Desarrollar un subsistema informático para la detección de roles de usuarios en colecciones de correos electrónicos que pueda ser integrado a MOCICE.

Para dar solución al objetivo general se plantean los siguientes objetivos específicos:

- Realizar sistematización teórica de la detección de roles en colecciones de correos electrónicos.
- Implementar el Subsistema de Detección de Roles de MOCICE.
- Realizar pruebas internas al Subsistema de Detección de Roles de MOCICE.

Para el desarrollo de la investigación se utilizaron los siguientes métodos teóricos y empíricos:

Teóricos

Analítico-Sintético: Ayudó a procesar el marco referencial de la tesis a partir de la sistematización del conocimiento científico relacionado con el Análisis de Redes Sociales. Permitió reconocer las múltiples relaciones y componentes del problema abordado por separado, para luego integrarlas en un todo como se presenta en la realidad. Además, fue la vía a través de la cual se realizó la interpretación de la información recogida, a fin de alcanzar las conclusiones correspondientes.

Análisis Histórico-Lógico: Permitió que se analizara el desarrollo histórico del problema a resolver, así como las publicaciones y *softwares* sobre los mecanismos de detección de roles que brindan soluciones similares a la que se necesita.

Empíricos

Experimento: Se utilizó para la selección de los algoritmos y las herramientas necesarias para la implementación del Subsistema de Detección de Roles.

Observación: Para la formulación del problema a solucionar.

El presente trabajo de diploma está estructurado en tres capítulos, tal como se describe a continuación.

Capítulo 1. Detección de Roles: En este capítulo se realiza un estudio de los sistemas y algoritmos que realizan la detección de roles existentes en la actualidad, así como sus principales características y funcionalidades. Se abordan varios conceptos clave y se explican las razones por las cuales se escogió la

Metodología de Desarrollo y las herramientas empleadas para el desarrollo de la solución propuesta.

Capítulo 2. Características del Subsistema de Detección de Roles: Se presenta la propuesta de solución del subsistema. Se realiza la captura de requisitos funcionales y no funcionales; se priorizan las historias de usuario a implementar así como las tareas asociadas a cada una de ellas y se realizan los diagramas de paquetes de clases del diseño, de clases del diseño y de la Base de Datos (BD).

Capítulo 3. Implementación y Pruebas del Subsistema de Detección de Roles: Luego de la implementación del Subsistema de Detección de Roles, se realizan las pruebas necesarias para validar las funcionalidades implementadas y verificar que den cumplimiento al subsistema desarrollado. Se realiza un plan de pruebas, un grupo de pruebas funcionamiento, de integración, de rendimiento y de estrés.

Finalmente, se presentan las Conclusiones, Recomendaciones, Lista de acrónimos, Referencias bibliográficas, Bibliografía y Anexos.

Capítulo 1

Detección de roles

En este capítulo se presenta una revisión de sistemas y algoritmos que realizan la detección de roles, así como las investigaciones realizadas sobre análisis de redes sociales y reconocimiento de roles. Se fija el marco teórico para poder establecer definiciones formales y presentar aspectos relacionados con la detección de roles. Se dan a conocer las Tecnologías que servirán de apoyo a la implementación del prototipo funcional y la confección de este documento.

1.1. Conceptos fundamentales

A continuación se presentan los conceptos fundamentales relacionados con el proceso de detección de roles, los cuales serán útiles para un mayor entendimiento del problema presente. Estos estarán implícitos en el trabajo de diploma, por lo que es necesario dejar bien claro el significado de cada uno de ellos.

1.1.1. Red Social

Una red social es una estructura social entre actores, ya sean individuos u organizaciones. Indica en qué forma se conectan a través de varias familiaridades sociales que van desde encuentros casuales hasta las relaciones de familia. El estudio de las redes sociales es conocido como Análisis de Redes Sociales (SNA)[1] o teoría de redes sociales. Investigaciones en un amplio número de campos académicos han demostrado que las redes sociales operan en muchos niveles, desde familias hasta naciones y desempeñan un papel crítico en la determinación de cómo los problemas son resueltos, las organizaciones son administradas y el grado en que las personas logran alcanzar sus metas [5].

[1] SNA: Social Network Analysis (por sus siglas en inglés)

1.1.2. Análisis de Redes Sociales

El SNA es un conjunto de técnicas matemáticas utilizadas por sociólogos para analizar interacciones sociales [6]. El concepto de red enfatiza el hecho de que los individuos tengan relaciones con otros individuos, cada uno de los cuales se relaciona con otros y así sucesivamente. La expresión red social se refiere al conjunto de actores y relaciones definidas entre estos. Tales redes permiten la extracción de patrones e implicaciones de las relaciones compartidas entre los actores.

1.1.3. Roles

Incluso si el concepto de rol es una de las ideas más populares en las ciencias sociales, una definición formal es difícil de encontrar. Existen dos corrientes diferentes; la primera se centra en el papel de la persona como individuo y ve los roles como las estrategias de afrontamiento que son adoptadas por las personas. La segunda se centra más en la persona como representante de una posición social y por lo tanto concibe los roles como patrones de comportamiento que son típicos en las personas cuyas posiciones sociales son similares. Se utilizará este último concepto en el presente trabajo, considerando los roles como patrones del comportamiento característico de las personas, tal como lo define Biddle [7]. De este modo, se asume que las personas son miembros de las posiciones sociales, y que dichas posiciones son las principales generadoras de funciones.

1.2. Trabajos similares

En el marco particular de la detección de roles, el estudio de herramientas que llevan a cabo este proceso resulta de gran importancia para tener una base sobre el funcionamiento del mismo. Hoy día el problema de reconocimiento de roles aún está muy descuidado en la literatura. En este sentido, el trabajo presentado en los próximos capítulos es uno de los primeros intentos de abordar el problema en Cuba, ya que hasta el momento no se encontró un sistema o investigación sobre detección de roles. Por otra parte, a nivel internacional se encontraron varios sistemas, los cuales serán estudiados con el fin de obtener procedimientos y algoritmos que podrían ser de utilidad para la propuesta de solución. A continuación se muestra una lista de algunos de estos sistemas y sus características:

1.2.1. InFlow

Creado por Orgnet. Es un *software* interactivo y comercial para el SNA. Integra visualización y análisis en un mismo sistema. Este se integra con Microsoft Office. Ha sido utilizado en creación de equipos de trabajo, diseño de organizaciones, periodismo investigativo, administración de conocimientos y descubrimiento de redes terroristas [8]. Tiene una serie de algoritmos para: centralidad, agrupamiento, Equivalencia Estructural y vecindad.

1.2.2. GeoSocialApp

Es una herramienta de visualización que soporta la exploración de redes sociales integrando métodos computacionales y gráficos, lo que posibilita el descubrimiento de patrones complejos en grandes redes sociales. Está implementado usando el entorno de desarrollo *GeoViz Tollkit* (GVT) y las estructuras de datos para los grafos están basados en la *Java Universal Network/Graph Framework* (JUNG) [9].

1.2.3. UCINET

Este es un *software* para analizar información de redes sociales, que puede leer de una multitud de diferentes formatos de texto, así como archivos de Excel. Puede manejar un máximo de 32 767 nodos aunque muchos de los procedimientos se hacen más lentos entre los 5 000 y 10 000. Los métodos de SNA incluyen medidas de centralidad, identificación de subgrupo, análisis de roles, teoría elemental de grafo y análisis estadístico basado en permutaciones. Adicionalmente el *software* posee fuertes rutinas de análisis de matrices tales como álgebra de matrices y estadísticas multivariadas [10].

UCINET tiene una ventana para el reconocimiento de roles, donde están habilitados algoritmos basados en equivalencia estructural, equivalencia regular y equivalencia automórfica.

1.2.4. Pajek

Es un programa de código abierto para el análisis y visualización de grandes redes sociales (varios miles de vértices). En el idioma eslovaco la palabra Pajek significa araña. El diseño de este *software* comenzó en 1996 por Andrej Nrva y fue implementado en Pascal. Pajek provee herramientas para el análisis de redes tales como redes de colaboración, interacción, de Internet, de citas de difusión y genealógicas. Una de sus principales ventajas es que, además de su propio formato, permite la entrada de datos en otros formatos tales como los de UCINET [11].

A pesar de las características y las ventajas que presentaron los *softwares* estudiados, ninguno de estos será utilizado en su totalidad para dar solución al problema planteado. Debido a que los sistemas InFlow, GeoSocialApp, UCINET y Pajek resultaron ser herramientas privativas, por lo que no pueden ser utilizadas en ningún proyecto de desarrollo. Además, al ser privativos no permiten modificaciones de su código fuente, por lo que no pueden ser reutilizados como parte del subsistema. No permiten la conexión con la Base de Datos (BD) y la integración con el sistema principal MOCICE. De todos estos *softwares* se tendrán en cuenta algunas de sus características como apoyo para la implementación del subsistema, como utilizar métodos de agrupamiento jerárquico para agrupar los actores según los roles y la implementación de más de un algoritmo para detectar roles.

Debido a los inconvenientes presentados por los sistemas, se hace necesario crear un sistema propio, por lo que se estudiaron algoritmos que puedan ser utilizados para realizar la detección de roles, los cuales serán explicados a continuación.

Para un mejor entendimiento de los algoritmos es necesario explicar la Equivalencia y el Modelado de Bloques, que son las bases de estos para realizar la detección de roles.

1.3. Equivalencia

El problema de agrupar actores que ocupan las mismas posiciones o que desempeñan el mismo rol en una red social, forma parte del análisis de redes sociales. En esta sección varios métodos de análisis de roles serán abordados.

Para lograr un mayor entendimiento de los métodos de equivalencia de roles y declarar formalmente las descripciones de estos, es necesario definir ciertos conceptos. En aras de la simplicidad solo se consideraran grafos no dirigidos. Las definiciones para grafos dirigidos y una visión matemática mas completa se puede ver en [12].

Sea $G(V, E)$ un grafo no dirigido, donde V es un conjunto de vértices que representa los actores de la red y E una relación simétrica $E \subseteq V \times V$ y sus elementos las aristas del grafo. Si $v \in V$, entonces $N(v) = \{u \in V; (u, v) \in E\}$ denota la vecindad de v. Otros conceptos de grafos se pueden encontrar en [13].

Si $\sim \subseteq V \times V$ es una relación de equivalencia y $v \in V$, entonces se denota como $[v] = \{u; u \sim v\}$ a la clase de equivalencia de v. El conjunto de clases de equivalencia define una partición de V. Una partición $P = \{C_1, ..., C_k\}$ de V se define como un conjunto no vacío de conjuntos disjuntos $C_i \subseteq V$, llamados clases o bloques, tales que $V = \bigcup_{i=1}^{k} C_i$, lo que significa que cada vértice $v \in V$ está en solo una clase. La relación $r : v \to [v]$ será llamada asignación de rol. Si dos vértices $u, v \in V$ son equivalentes, entonces se dice que u y v ocupan la misma posición o desempeñan el mismo rol.

De forma general, los métodos de asignación de roles tienen como objetivo dividir el conjunto de vértices en diferentes clases a partir de alguna expresión de compatibilidad. A continuación se describen tres de estas expresiones de compatibilidad.

1.3.1. Equivalencia estructural

Esta forma de equivalencia fue presentada por Lorrain & White en 1971 [14] siendo la más simple y a su vez la más restrictiva de todas.

Definición: Sea $G(V, E)$ un grafo y r una asignación de rol en V, entonces r se denomina estructural si para todo $u, v \in V$

$$r(u) = r(v) \Longrightarrow N(u) = N(v).$$

Lo que significa que dos nodos son equivalentes estructuralmente si sus vecindades son iguales, o sea, si tienen las mismas relaciones con todos los demás nodos.

En la figura se presentan algunos ejemplos de equivalencia estructural. En la estrella la asignación de roles detecta al vértice central en un rol y en otro a los demás. El grafo bipartito muestra en dos roles diferentes sus respectivas particiones: los nodos de la derecha en un rol y los de la izquierda en el otro. El grafo completo de la derecha muestra un conjunto de vértices que pertenecen al mismo rol al estar todos conectados entre sí.

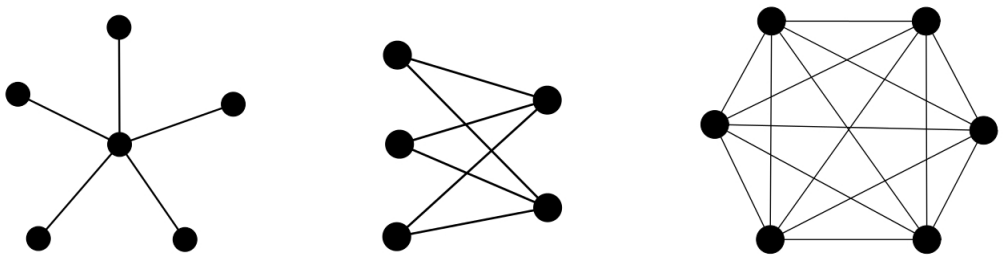

Figura 1.1: Estrella (izquierda), Grafo bipartito (medio) y Grafo completo (derecha) [12].

La equivalencia estructural es teórica y computacionalmente bastante sencilla. Pero resulta demasiado estricta y no cumple con la noción sociológica de rol, por lo que no es recomendable su uso para análisis de redes irregulares como aquellas presentes en el mundo real. No obstante, es la base para varios procedimientos que surgieron posteriormente a partir de generalizaciones o relajaciones de esta equivalencia.

1.3.2. Equivalencia automórfica

Esta equivalencia enuncia que dos nodos son automórficamente equivalentes si ambos son estructuralmente intercambiables y es menos restrictiva que la equivalencia estructural [15, 16].

Definición: Sea $G = (V, E)$ un grafo y $u, v \in V$, entonces se dice que u y v son automórficamente equivalentes si existe un automorfismo φ de G, que cumpla que $\varphi(u) = v$. Las particiones o clases automórficas en las que se agrupan los nodos se denominan orbitas.

Para una mejor explicación se presenta la siguiente Figura:

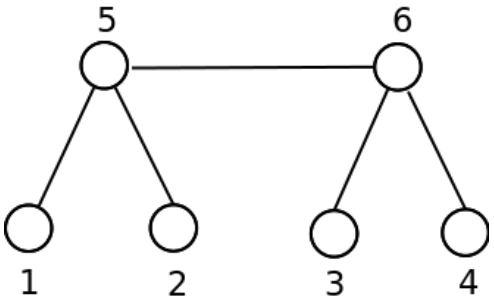

Figura 1.2: Nodos $\{1, 2, 3, 4\}$ y $\{5, 6\}$ son automórficamente equivalentes.

En la Figura 1.2 los vértices 1 y 2 tienen el mismo rol según la equivalencia estructural. Lo mismo ocurre para los vértices 3 y 4 sin embargo 2 y 3 no son estructuralmente equivalentes así como 5 y 6. Si los nodos del grafo no tuvieran ninguna etiqueta no existiría forma de diferenciar los nodos $\{1, 2, 3, 4\}$ y $\{5, 6\}$. Por ejemplo si se quisiera asignar la etiqueta “5” a un nodo se tendrían dos posibilidades: cualquiera de los vértices con tres aristas. Para lograr asignar la etiqueta “5” es necesario conocer más información, por ejemplo la etiqueta de otro vértice. La equivalencia estructural proporciona dicha información, por lo que 5 y 6 no son estructuralmente equivalentes. En cambio en la equivalencia automórfica $\{1, 2, 3, 4\}$ y $\{5, 6\}$ son las orbitas del grafo y representan los roles presentes en este, o sea los nodos 5 y 6 son intercambiables por lo que son automórficamente equivalentes, así como $1, 2, 3$ y 4.

Esta equivalencia aunque menos restrictiva que la estructural no puede determinar roles equivalentes en redes irregulares como las redes del mundo real, debido a que por regla general, en estas no hay automorfismos relevantes y los que hay son triviales (contienen pocos nodos), además este proceso es computacionalmente costoso y complejo.

1.3.3. Equivalencia regular

La idea de equivalencia regular es una relajación de la equivalencia estructural, surge en una publicación escrita por Sailer [17]. En la misma el autor propone que los actores desempeñan el mismo rol si están conectados a otros actores en un mismo rol, a diferencia de la equivalencia estructural donde deben estar conectados con los mismos actores. La definición formal de la equivalencia regular fue dada por White y Reitz [18]. Borgatti y Everett [16] dieron otra definición de equivalencia regular basada en función de coloreado: los vértices de un color determinado son regularmente equivalentes si sus vecindades tienen

los mismos colores. Si r es una asignación de rol en V y $U \subseteq V$, entonces $r(U) = \{r(u); u \in U\}$ es el conjunto de clases (o colores) que tienen los miembros de U.

Definición: Sea $G = (V, E)$ un grafo. Una asignación de rol r en V es regular si para todo $u, v \in V$

$$r(u) = r(v) \Longrightarrow r(N(u)) = r(N(v)).$$

En la siguiente imagen se muestra un ejemplo de equivalencia regular. Los vértices coloreados de un mismo color pertenecen a la misma clase de equivalencia regular. Nótese que los vértices equivalentes pueden tener relaciones con otros roles con grados diferentes, por ejemplo los vértices grises.

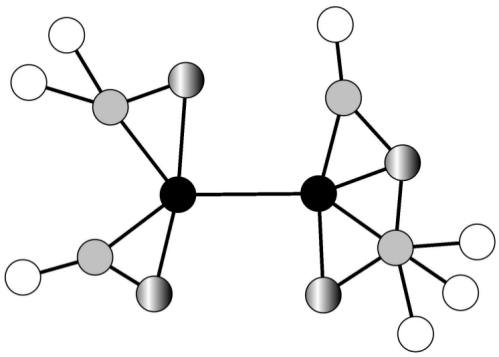

Figura 1.3: Grafo con equivalencias regulares no triviales indicadas por el coloreado de los vértices [19].

La equivalencia regular es computacionalmente compleja y no maneja adecuadamente las relaciones simétricas. Además, esta equivalencia necesita una partición inicial de los nodos y está diseñada para trabajar de manera local y no para representar la estructura global de una red, especialmente si esta es de gran tamaño.

1.3.4. Otras equivalencias

Existen otras formas de equivalencia como la equivalencia perfecta, equivalencia exacta y equivalencia estocástica, que en general son casos especiales de las mencionadas anteriormente. El artículo [12] ofrece una descripción de estas equivalencias.

1.4. Modelado de bloques

El modelado de bloques o “blockmodeling” es una técnica de agrupamiento que tiene como objetivo agrupar los actores de una red, de acuerdo con el grado en que sus relaciones son equivalentes en correspondencia con alguna de las definiciones de equivalencia vistas anteriormente.

Un bloque consiste en un grupo de actores que tienen patrones de relaciones similares y un modelo de bloques es la representación de las relaciones entre los bloques de una red.

Formalmente un modelo de bloques es una permutación de la matriz de adyacencia de un grafo formando una versión reducida de este, de tal forma que en cada bloque solo existen relaciones similares. Una definición matemática formal del modelado de bloques puede ser encontrada en [20].

Con el objetivo de ilustrar el modelado de bloques se presenta el siguiente ejemplo: la Figura 1.4 muestra un grafo, su matriz de adyacencia y las posiciones obtenidas de un análisis de equivalencia estructural. La Figura 1.5 representa el modelo de bloques correspondiente y el grafo formado por las posiciones y las relaciones entre estas.

E	1	2	3	4	5	6	7	8	9
1	-	0	0	1	0	0	0	0	1
2	0	-	0	0	1	0	1	0	0
3	0	1	-	0	1	1	1	1	0
4	1	0	0	-	0	0	0	0	1
5	0	1	0	0	-	0	1	0	0
6	0	1	1	0	1	-	1	1	0
7	0	1	0	0	1	0	-	0	0
8	0	1	1	0	1	1	1	-	0
9	1	0	0	1	0	0	0	0	-

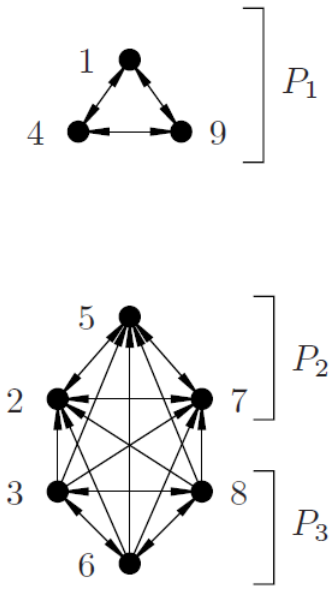

Figura 1.4: Matriz de adyacencia de un grafo y sus respectivas particiones a partir de la equivalencia estructural [20].

E	3	6	8	1	4	9	2	5	7
3	-	1	1	0	0	0	1	1	1
6	1	-	1	0	0	0	1	1	1
8	1	1	-	0	0	0	1	1	1
1	0	0	0	-	1	1	0	0	0
4	0	0	0	1	-	1	0	0	0
9	0	0	0	1	1	-	0	0	0
2	0	0	0	0	0	0	-	1	1
5	0	0	0	0	0	0	1	-	1
7	0	0	0	0	0	0	1	1	-

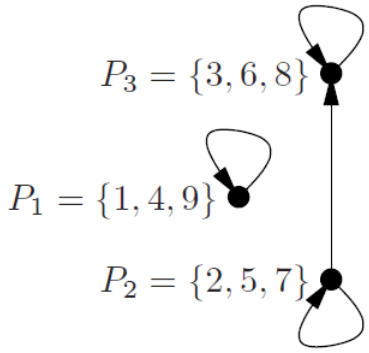

Figura 1.5: Matriz de adyacencia permutada correspondiente al grafo de la Figura 1.4. Los bloques en la matriz están separados por líneas a partir de la equivalencia regular [20].

1.5. Algoritmos para la detección de roles

Durante la investigación realizada, los siguientes algoritmos se tomaron como posibles candidatos a implementar en el Subsistema de Detección de Roles.

1.5.1. CONCOR

Es un algoritmo tradicional de modelado de bloques, fue presentado por Breiger, Boorman y Arabie [21] en 1975 para sugerir un modelo de bloques de una red utilizando como medida de similitud la equivalencia estructural. Se basa en el análisis de una matriz de correlación[2], que se determina a partir de los valores de los coeficientes de correlación de Pearson[3] [22, 23] existente entre dos nodos. El proceso de análisis ocurre iterativamente sobre la matriz de correlación, dividiéndola en dos bloques que son analizados siguiendo el mismo procedimiento hasta que en cada bloque solo queden dos nodos o hasta que el usuario decida. El algoritmo tiene una complejidad temporal de $O(n^3)$ por cada iteración que realice.

[2]La matriz formada por los valores de correlación entre todos los nodos del grafo se denomina matriz de correlaciones del grafo.

[3]El coeficiente de correlación de Pearson es utilizado como medida para determinar en qué grado dos variables están relacionadas linealmente; siendo 1 el valor de una relación linear perfecta y 0 la no existencia de relación linear.

Debido a la alta complejidad temporal y al uso de la equivalencia estructural (que no contempla la idea de que dos actores desempeñan el mismo rol si tienen relaciones similares con otros actores con los mismos roles) no es recomendable el uso de este algoritmo para detectar roles en redes sociales como las redes de correo electrónico.

Otras críticas más profundas sobre el funcionamiento del algoritmo pueden encontrarse en [24, 25, 26].

1.5.2. Maxsim

Este algoritmo fue presentado por Everett y Borgatti [27] en 1988. Detecta las similitudes estructurarles de los nodos de una red basándose en la equivalencia automórfica. El algoritmo calcula a partir de la matriz de adyacencia del grafo, una matriz de distancias entre los nodos a partir de la cual se determina cuales nodos pertenecen a las mismas clases de equivalencia. La medida de distancia utilizada por el algoritmo es la distancia euclidiana entre los nodos y el agrupamiento de los mismos ocurre mediante un método de agrupamiento jerárquico. Su complejidad temporal es de $O(n^3)$. Este método al utilizar equivalencia automórfica padece de las mismas restricciones de esta.

Este algoritmo no es recomendable para ser utilizado en redes irregulares y/o de baja densidad debido a que las clases de los vértices automorficamente equivalentes resultan demasiado pequeñas y carentes de significado.

1.5.3. REGE

Este algoritmo fue creado por los esfuerzos conjuntos de Lee Sailer, Jhon P Boyd, Douglas R. White y Karl Reitz. Fue presentado por White [28] en 1984. Fue implementado para determinar roles en una red mediante la equivalencia regular. Tienen una complejidad temporal de $O(n^3)$. Es un algoritmo iterativo que proporciona una medida del grado de equivalencia e_{ij} de todos los pares de nodos i y j. Inicialmente $e_{ij} = 1$ (partición inicial) para todos los pares de nodos y en cada iteración se optimiza ese valor a partir del grado de equivalencia entre los nodos adyacentes a i y los adyacentes a j. En la primera iteración, e_{ij} se obtiene calculando la medida en que las aristas de i se corresponden con las aristas de j, en las siguientes iteraciones se calcula a partir de los valores de la iteración anterior.

Debido a que este algoritmo posee una alta complejidad temporal no se recomienda su utilización para analizar grandes redes sociales. Además, al analizar redes sociales reales con este algoritmo se pueden obtener resultados triviales donde la mayoría de los nodos pertenecen a una misma clase de equivalencia.

1.5.4. RART

Este algoritmo fue presentado por A. McCallum, X. Wang y A. Corrada-Emmanuel [29] en el año 2007. Es un modelo matemático que obtiene a partir de una red social de correos electrónicos los temas de los mensajes y los roles de los usuarios a partir de estos. Utiliza distribuciones probabilísticas para cada par remitente-receptor sobre las palabras de los mensajes para determinar los temas. Para determinar los roles de cada actor se utiliza otra distribución probabilística sobre los temas de los mensajes y se analiza la estructura de la red. Este modelo parece tener una muy buena eficiencia debido a que no se limita solamente a analizar la estructura de la red sino que toma en cuenta los atributos de las relaciones.

Aunque en una primera instancia este sería el mejor modelo de los seleccionados para realizar la detección de roles en el subsistema a implementar existen algunas razones a tener en cuenta:

- La documentación existente del modelo está incompleta pues este cuenta con tres variantes y solo se ha publicado la descripción de una de ellas.
- La complejidad matemática del modelo hace difícil la comprensión de su funcionamiento y la traducción de este a un lenguaje de programación.
- La estimación del tiempo necesario para interpretar el funcionamiento del modelo excede el tiempo de desarrollo del subsistema.

1.5.5. Triad Census

Este algoritmo fue publicado en 1987 por Hans Hummel y Wolfgang Sodeur [1]. Ofrece una solución práctica para determinar equivalencia de roles. Tiene una complejidad temporal de $O(n \log n)$. El funcionamiento del algoritmo se basa en la obtención de los patrones de las relaciones existentes entre tres

nodos, a partir de estos patrones se calculan las similitudes entre los nodos y se determinan los roles a los que pertenecen. Al tratar los roles de los actores como un conjunto de relaciones con otros roles y no con un conjunto de actores y comparar los individuos en función de patrones de tríadas Hummell y Sodeur generalizaron la equivalencia más allá de tener relaciones idénticas con otros individuos.

Al culminar el proceso de investigación se decide implementar este algoritmo por las siguientes ventajas que presenta:

- Plantea un modelo intuitivo y computacionalmente simple haciéndolo muy útil para la implementación en microcomputadores.
- No contiene iteraciones, lo que evita fallas en la convergencia y a diferencia de la equivalencia regular maneja perfectamente tanto las relaciones asimétricas como las simétricas.
- Permite analizar grandes redes del mundo real lo que posibilita su implementación en sistemas de detección de roles.

1.6. Tecnologías

Realizar una selección correcta de las tecnologías con las que se construirá la nueva aplicación informática, tiene un papel primordial en la calidad y seguridad final del proyecto, además permiten agilizar el proceso de construcción del subsistema. Para la selección de las mismas se tienen en cuenta los conocimientos del equipo de trabajo y las necesidades del trabajo a realizar. A continuación se describen las tecnologías utilizadas para la realización del Subsistema de Detección de Roles.

1.6.1. Metodología

No existe una metodología de *software* universal. Cada equipo de desarrollo escoge la metodología según las características de su proyecto, por lo que es importante determinar el alcance del proyecto antes de escoger la metodología que se va a emplear en el desarrollo del mismo. Para la realización de este proyecto se escogió ScrumXP (SXP) que es una metodología compuesta por la metodología

Programación eXtrema (XP) y SCRUM que es un método adaptativo de gestión de proyectos que se basa en principios ágiles.

SCRUM es una forma de gestionar un equipo de manera que trabaje de forma eficiente y de tener siempre medidos los progresos, de forma que se sepa por qué etapa se encuentra el proceso. XP es una metodología encaminada al desarrollo; consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo al usuario final, pues es uno de los requisitos para llegar el éxito del proyecto.

SXP ofrece una estrategia tecnológica, a partir de la introducción de procedimientos ágiles que permitan actualizar los procesos de *software* para el mejoramiento de la actividad productiva fomentando el desarrollo de la creatividad, aumentando el nivel de preocupación y responsabilidad de los miembros del equipo, ayudando al líder del proyecto a tener un mejor control del mismo. Está especialmente indicada para proyectos de pequeños equipos de trabajo, rápido cambio de requisitos o requisitos imprecisos, muy cambiantes, donde existe un alto riesgo técnico y se orienta a una entrega rápida de resultados y una alta flexibilidad. Ayuda a que trabajen todos juntos, en la misma dirección, con un objetivo claro, permitiendo además seguir de forma clara el avance de las tareas a realizar, de forma que los jefes pueden ver día a día cómo progresa el trabajo [30].

La metodología SXP está dividida en cuatro fases [31] :

- **Planificación-Definición:** Tiene como propósito establecer la visión, fijar expectativas, y asegurar financiamiento. En esta fase se escribe la Plantilla de Concepción del Sistema, se crea la Lista de Reserva del Producto (LRP), se confeccionan las plantillas de Historias de Usuarios, Lista de Riesgos, Modelo de Diseño y la Plantilla de Arquitectura de *Software*.
- **Desarrollo:** Esta fase tiene como objetivo implementar un sistema listo para entregar en una serie de iteraciones de 60 días. Se realizan también un número de plantillas como Glosario de Términos, Tareas de Ingeniería, Plan de Pruebas, Caso de Prueba de Aceptación.
- **Entrega:** En esta fase se hace entrega del producto. Se confecciona el Manual de Usuario, Identidad y de Desarrollo.
- **Mantenimiento:** Se realiza el soporte para el cliente. Se elabora la Plantilla de Gestión de Cambios.

1.6.2. Lenguajes de programación

Para el desarrollo de aplicaciones en sistemas Unix/Linux existen diferentes lenguajes de programación, tales como: Java, C++, Python, C, etc. La elección de uno de ellos debe sustentarse principalmente en las particularidades de la solución que se proponga y los conocimientos de los desarrolladores de los mismos. Se seleccionó C++ dadas las siguientes características [32]:

- Es un lenguaje de programación orientado a objetos.
- Permite la programación estructurada.
- Es un lenguaje compilado, lo que le hace muy rápido.
- Es uno de los lenguajes más empleados en la actualidad, por lo que posee un gran número de usuarios y existe una gran cantidad de libros, cursos, páginas web, entre otros dedicados a él.
- Es un lenguaje que permite el manejo de todo tipo de estructuras de datos (arreglos, pilas, colas, textos, objetos, etc.) por lo que es capaz de resolver todo tipo de problemas.
- Existe una gran cantidad de compiladores, depuradores y librerías para este lenguaje.

Además, se decidió utilizar C++ pues todos los sistemas con los que interactúa el subsistema están implementados en este lenguaje, lo que facilita la integración con los mismos.

1.6.3. Entornos de Desarrollo Integrado

Las ventajas principales de un Entorno de Desarrollo Integrado (IDE)[4] radican en la automatización de algunas tareas asociadas al desarrollo de aplicaciones y al soporte para algunas funcionalidades como el auto completamiento de código y la depuración, contribuyendo a evitar pérdidas de tiempo. La selección de un IDE para el desarrollo de esta aplicación se centró en herramientas libres. Aunque existen varios candidatos entre los que están Eclipse, Notepad++, CodeBlocks, NetBeans. Se escogió este último de acuerdo con las siguientes características [33]:

[4] IDE: Integrated Development Environment (por sus siglas en inglés)

- Por la experiencia de los desarrolladores en el trabajo con el NetBeans.
- Editor de código sensible al contenido. Con soporte para completamiento del código, coloreado de etiquetas, autotabulación y uso de abreviaturas para varios lenguajes de programación.
- Soporta el control de versiones y el desarrollo colaborativo.
- Tiene un Editor de código fuente avanzado.
- Soporta el desarrollo de aplicaciones en otros lenguajes como Java, C, C++, Lenguaje de Marcado Extensible (XML)[5] y Preprocesador de Hipertexto (PHP).

1.6.4. Herramientas de la Base de Datos

Existen diferentes tipos de gestores de bases de datos, de los cuales se tomaron en consideración, como posibles opciones para la creación de la base de datos de la aplicación; PostgreSQL, MySQL, Drizzle, SQLite, Apache Derby. Luego de ser analizados detalladamente se escogió PostgreSQL por las siguientes características [34]:

- Es considerado como uno de los gestores de BD de *software* libre más potentes, cumpliendo así con las especificaciones del cliente de realizar el subsistema utilizando herramientas libres.
- Sistema de gestión de BD objeto relacional.
- Publicado bajo la licencia Distribución de Software Berkeley (BSD)[6].
- Posee documentación muy bien organizada, pública y libre.
- Puede operar sobre distintas plataformas, incluyendo Linux, Windows, Unix, Solaris y MacOS X.
- Buen sistema de seguridad mediante la gestión de usuarios, grupos de usuarios y contraseñas.
- Buena escalabilidad ya que es capaz de ajustarse al número de Unidad Central de Procesamiento (CPU) y a la cantidad de memoria disponible de forma óptima, soportando una mayor cantidad de peticiones simultáneas a la base de datos de forma correcta.

[5] XML: eXtensible Markup Language (por sus siglas en inglés)
[6] BSD: Berkeley *Software* Distribution (por sus siglas en inglés)

Se eligió pgAdmin para la administración del gestor de bases de datos PostgreSQL, por las características que se presentan a continuación [35]:

- Es la aplicación gráfica más completa y popular con licencia Open Source.
- Está diseñado para responder a las necesidades de todos los usuarios, desde escribir consultas Lenguaje de Consulta Estructurado (SQL) simples hasta desarrollar bases de datos complejas.
- Su interfaz gráfica soporta todas las características de PostgreSQL y facilita enormemente la administración.
- La aplicación incluye un editor SQL con resaltado de sintaxis, un editor de código de la parte del servidor, un agente para lanzar scripts programados, soporte para el motor de replicación Slony-I.
- La conexión al servidor puede hacerse mediante conexión TCP/IP o Unix Domain Sockets y puede encriptarse mediante Capa de Conexión Segura (SSL)[7] para mayor seguridad.

1.6.5. Herramientas de modelado

Existen diferentes tipos de herramientas de modelado, entre las cuales se encuentran Rational Rose, DIA, UMBRELLO, BoUML, ArgoUML, ArgoUML, Visual Paradigm. Se escogió esta última ya que [36]:

- Tiene disponibilidad en múltiples plataformas y en múltiples versiones. Esta característica es muy importante pues Visual Paradigm está disponible para varios sistemas operativos como Windows, Linux, Unix.
- Es una herramienta que apoya el ciclo de vida completo del desarrollo de *software*.
- Proporciona una plataforma de modelado colaborativo para el trabajo en equipo, los miembros pueden ver y editar el mismo proyecto, o el mismo esquema, incluso de forma simultánea.
- Ayuda a una rápida construcción de aplicaciones de calidad, mejores y a un menor costo.
- Genera código fuente a partir de los modelos de clases.

[7] SSL: Secure Socket Layer (por sus siglas en inglés)

1.6.6. Herramientas para la revisión de código

RATS es una herramienta que permite realizar revisiones de código fuente escrito en los lenguajes C, C++, Perl, Python y PHP. Tiene como objetivo detectar errores de programación y vulnerabilidades en el código. Esta herramienta solo efectúa una revisión superficial del código, siendo muy probable que no encuentre todos los errores o que genere falsos positivos. Por ello, a pesar de disponer la herramienta de apoyo, se deberá realizar una revisión manual del código fuente [37].

Valgrind es una herramienta libre que ayuda en la depuración de problemas de memoria y rendimiento de programas [38]. Está compuesta por otras herramientas, la más usada es Memcheck que introduce código de instrumentación en el programa a depurar, lo que le permite realizar un seguimiento del uso de la memoria y detectar los siguientes problemas:

- Uso de memoria no inicializada.
- Lectura/escritura de memoria que ha sido previamente liberada.
- Lectura/escritura fuera de los límites de bloques de memoria dinámica.
- Fugas de memoria.

1.6.7. Otras herramientas

Las herramientas que se presentan a continuación fueron de gran ayuda en la confección del documento de tesis y la generación de la documentación del código, las mismas son:

TEX: Es un lenguaje de composición con capacidad para macros escrito por Donald E. Knuth. TEX toma una secuencia de comandos de tipografía, escritos en un guión en un archivo ASCII y los ejecuta. Muchos de los “trucos” del proceso de impresión fueron modelados por Knuth como algoritmos de computación e incorporados a TEX, de ahí su excelente apariencia impresa [39].

LATEX: Es un sistema de preparación de documentos diseñado por Leslie Lamport en 1985, desarrollado a partir de un lenguaje de tipografía llamado TEX [39]. Es muy utilizado para la composición de artículos

académicos, tesis y libros técnicos, dado que la calidad tipográfica de los documentos realizados con LaTeX es comparable a la de una editorial científica de primera línea.

Kile: Es un IDE de desarrollo para LaTeX. Posee comandos avanzados de edición, una interfaz intuitiva que posibilita el desarrollo fluido del documento.

Doxygen: Es una herramienta que permite generar de forma automática documentación para C++, C, Java, Objective-C, Python, IDL (versiones Corba y Microsoft), PHP y C#. Puede generar la documentación en formato *Hyper Text Markup Language* (HTML), LaTeX , RTF y XML [40].

Conclusiones parciales

Aunque gran parte de los sistemas investigados en la sistematización teórica presentaron algunas de las características que debía poseer el Subsistema de Detección de Roles, no contaban con todos los requisitos solicitados por el cliente; como integrarse con MOCICE y guardar los resultados obtenidos en una base de datos. Esto evidencia la necesidad de crear un subsistema propio que sea capaz de realizar la detección de roles y cumplir con las especificaciones mencionadas.

De los algoritmos de detección de roles investigados se decidió utilizar el Triad Census, que resultó ser el más factible para la implementación del subsistema.

Se seleccionó SXP como metodología para guiar el proceso de desarrollo de *software*, resultando ser la más adecuada para el equipo de desarrollo, por su flexibilidad, adaptabilidad y previsibilidad.

Capítulo 2

Propuesta del Subsistema de Detección de Roles

En este capítulo se presenta la propuesta de solución del subsistema, sus funcionalidades, características y la arquitectura seleccionada, iniciando así el desarrollo de la solución guiada por la metodología de desarrollo SXP. También se realiza la captura de requisitos, actividad de vital importancia para la implementación del Subsistema de Detección de Roles y se realiza la concepción de las Historias de Usuario de cada requisito funcional.

2.1. Modelo de dominio

Un modelo del dominio captura los objetos más importantes en el contexto del sistema. Los objetos del dominio representan conceptos o eventos del entorno de trabajo del sistema [41, 42]. Para lograr un mayor entendimiento del proceso de detección de roles se realizó el Modelo de Dominio mostrado en la Figura 2.1, el cual está compuesto por los principales conceptos asociados a la situación problemática.

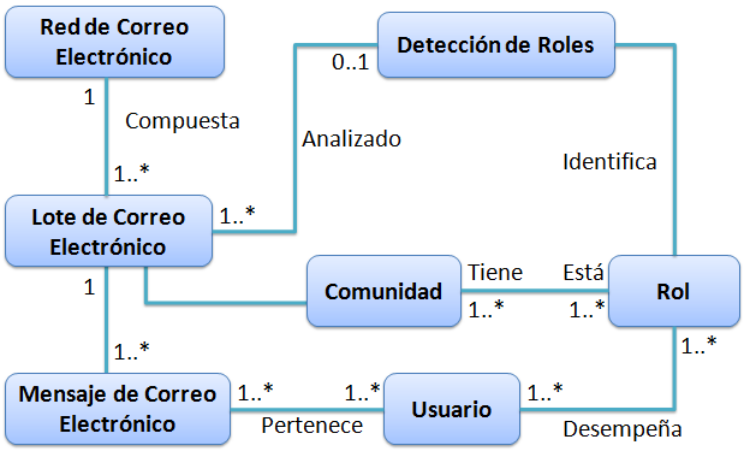

Figura 2.1: Modelo de Dominio.

2.2. Descripción de la propuesta de solución

Después de todo lo expuesto en el capítulo anterior, se propone un Subsistema de Detección de Roles en redes de correo electrónico, que realizará sus funciones mediante la implementación de varios algoritmos. Para su total funcionamiento el subsistema se integrará con el sistema principal (MOCICE), el cual mediante su Controlador indicará qué lote de correo electrónico y que comunidad de interés se desea analizar. Presentará una arquitectura flexible que permita la adición de nuevos algoritmos para la detección de roles, ya que en otro momento podría necesitarse un algoritmo diferente y no será necesario modificar todo el subsistema. Este funcionará de forma distribuida, ya que podrá estar ejecutándose en varias máquinas a la vez. Y además contará con un sistema para llevar un registro de todos los eventos que ocurran en el subsistema y mediante un archivo de configuración se cargarán los datos necesarios para el proceso de detección de roles. La Figura 2.2 muestra la solución propuesta.

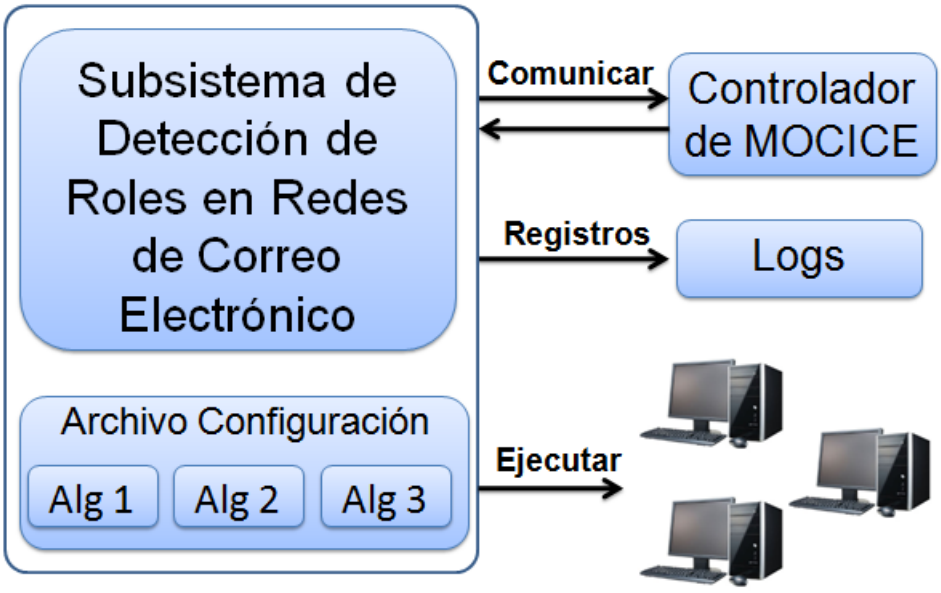

Figura 2.2: Propuesta de Solución.

2.2.1. Ciclo de ejecución del Subsistema de Detección de Roles

Al iniciar, el subsistema lee el archivo de configuración donde se encuentran los parámetros necesarios para establecer la comunicación con el Controlador de MOCICE, como la dirección Protocolo de Internet

(IP) del controlador y el puerto de comunicación.

Una vez obtenidos los parámetros se establece la comunicación con el Controlador de MOCICE y se le envía un mensaje informándole de que el subsistema está listo para realizar la detección de roles.

Como respuesta el Controlador de MOCICE envía al subsistema una comunidad perteneciente a un lote determinado, a la cual se le desea realizar la detección de roles.

Se lee del archivo de configuración los parámetros necesarios para conectarse a la BD, como la dirección, el puerto, el nombre, el usuario y la contraseña.

Con los parámetros anteriores se establece la conexión con la BD y se obtienen los datos de la comunidad que pertenece al lote enviado.

Se lee el archivo de configuración para obtener los parámetros del algoritmo de detección de roles como el nombre (en este caso el algoritmo propuesto por los autores: Triad Census) y la dirección donde se encuentra.

Se carga el algoritmo que se especificó en el archivo de configuración, el cual realiza el proceso de detección de roles que se divide en tres etapas:

Etapa de Pre-procesamiento de los datos de la comunidad:
Con la información obtenida de la BD se crea un grafo que, dependiendo del algoritmo de detección de roles utilizado, puede ser dirigido o no donde los nodos corresponden a los usuarios de la red de correos y las aristas a los mensajes de correo que se envían los usuarios.

Etapa de Clasificación:
De forma general en esta etapa se procesa el grafo y se descubren las regularidades existentes, determinando las similitudes entre los nodos. Específicamente el Triad Census durante esta etapa procesa el grafo y analiza para todos los nodos todas sus aristas, con el objetivo de obtener las tríadas en las que están involucrados. Crea para cada nodo el vector de frecuencias de tríadas el cual representa el rol de cada actor.

Etapa de Agrupamiento:

En general en esta etapa dadas las similitudes obtenidas en la fase de clasificación se crean grupos con los nodos similares, los cuales son etiquetados como roles. El Triad Census dados los vectores de frecuencias de tríadas de cada nodo calcula las similitudes entre los nodos utilizando como medida de similitud la distancia euclidiana, como se describe en el algoritmo (ver Ecuación 2.1). Posteriormente se crean grupos, haciendo uso de un método de agrupamiento jerárquico, con los nodos similares, a los cuales se les etiqueta como rol. Además, se crea un árbol con la jerarquía de los roles el cual representa el resultado del algoritmo.

Posteriormente se almacenan los resultados obtenidos por el algoritmo en la BD guardándose los roles detectados y los usuarios que pertenecen a cada rol. Como la detección de roles se realiza de forma no supervisada no es posible asignarle a los roles nombres, por lo que los roles son denotados con identificadores. Los usuarios que tienen comportamientos similares, estarán asociados al mismo identificador que representa el rol. Finalmente, se informa al Controlador de MOCICE que el análisis ha finalizado y que el subsistema está listo nuevamente.

2.2.2. Descripción del algoritmo Triad Census

En aras de una mejor compresión sobre el proceso de detección de roles y como este es realizado por el algoritmo seleccionado se describe a continuación el funcionamiento del mismo.

La ejecución de este algoritmo se realiza durante la segunda y tercera etapa del proceso de detección explicado anteriormente.

El modelo consta de tres pasos:

- En el primer paso el algoritmo expresa cada rol de los individuos como un conjunto de tríadas de actores, o sea, como un conjunto de patrones que describen la orientación de cada actor con respecto a los demás.

  Cada nodo puede estar presente en una de las 36 relaciones de la Figura 2.3

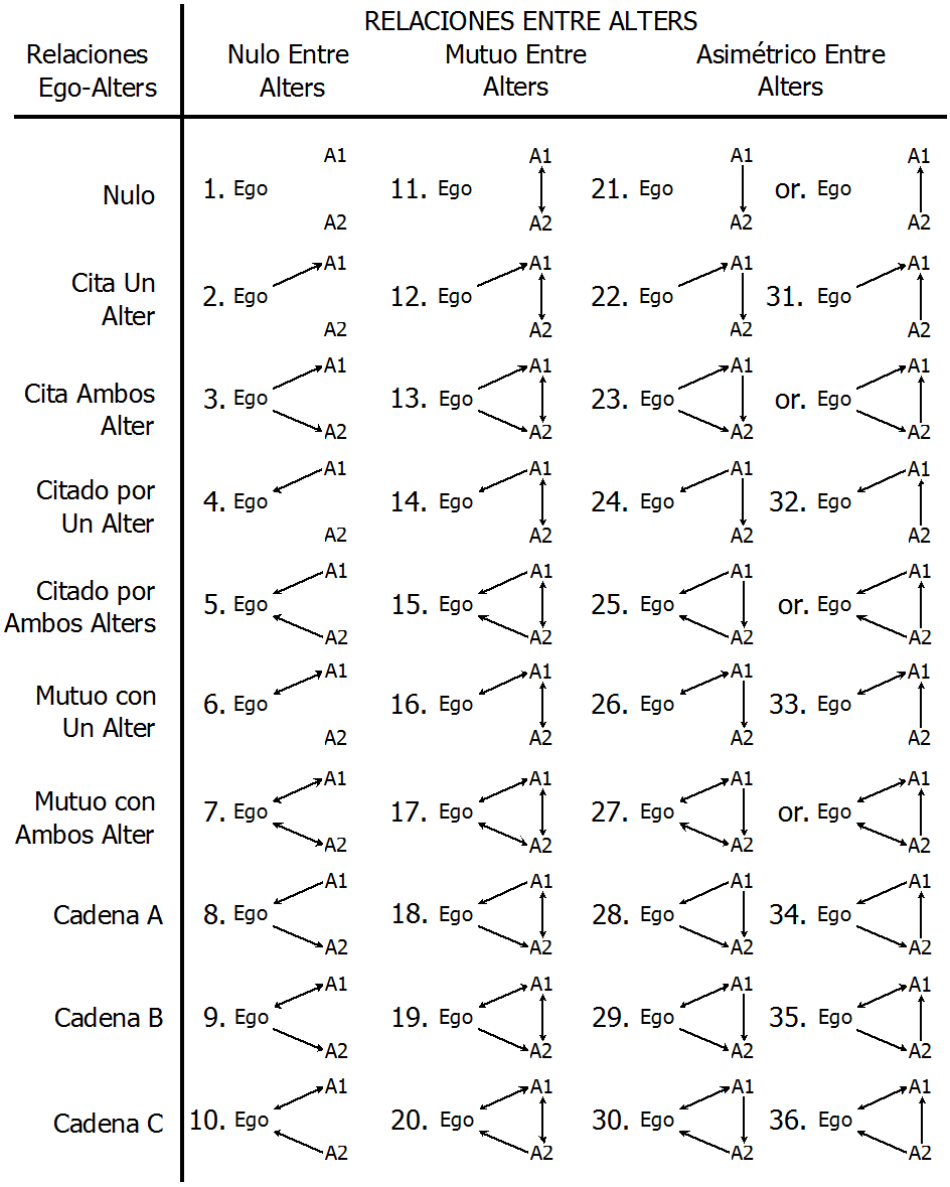

Figura 2.3: Tipos de tríadas que definen roles en una red.

Las tres columnas distinguen las relaciones de las tríadas de acuerdo con las relaciones entre los alters. En la primera columna no existen relaciones entre estos. En la segunda cada alter dirige una relación hacia el otro. En la última solo uno de los alters dirige la relación hacia el otro. Hay dos tipos distintos de tríadas en la tercera columna, esta distinción ocurre cuando el ego tiene diferentes relaciones con los alter, haciendo importante saber cuál alter mantiene la relación con el ego. En la primera fila por ejemplo no existe distinción en que alter cita al otro, el ego no tiene relación con ninguno de ellos por lo que las relaciones entre los alter no afectan las del ego. Por otra parte, en la segunda fila hay dos tríadas diferentes en la tercera columna, existe una relación desde el ego al

primer alter y no tiene contacto con el segundo. Si existe una relación asimétrica del segundo alter al primero, entonces el ego es uno de los dos individuos que dirigen relaciones hacia el primer alter, si existe una relación desde el primer alter al segundo entonces el ego forma parte de una relación asimétrica en cadena desde el ego al primer alter y de este al segundo.

La idea principal de la Figura 2.3 es que las identidades de los alters están definidas por los patrones de sus relaciones con el ego y entre ellos. Cada uno de los patrones representa un componente de un rol. La Figura 2.3 es un inventario de esos componentes, y la frecuencia relativa con que cada actor desempeña el papel de ego en cada tríada dentro de la red, define cual es el rol del actor. Entonces dos individuos poseen el mismo rol en la red si están involucrados en los mismos componentes de un rol.

Específicamente en una red de n actores, cada actor estará involucrado en $\frac{(n-1)(n-2)}{2}$ tríadas, cada una de estas tríadas será uno de los 36 patrones de la Figura 2.3, donde t_{jq} es la frecuencia con que un actor j está involucrado en un trío q. El conjunto de patrones para un actor es entonces un arreglo de las 36 frecuencias.

Para demostrar el proceso se ofrece el ejemplo de la (Figura 2.4).

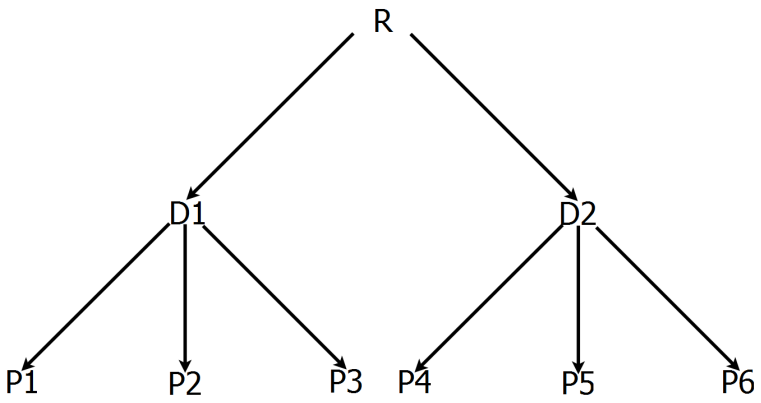

Figura 2.4: Red de dos facultades de una universidad.

El grafo o red de este ejemplo muestra la estructura de dos facultades cada una (en aras de simplificar el ejemplo) con 3 profesores (P1-P6), los decanos de cada facultad (D1,D2) y el rector (R). Cada individuo en esta red está involucrado en 28 relaciones triádicas $\frac{7*8}{2}$.

Para ilustrar esto se tomará como referencia a uno de los decanos de una de las facultades y se presentarán las relaciones que mantiene el mismo.

La posición de un decano está involucrada en siete de los 36 tipos de tríadas.

El decano está involucrado en las siguientes tríadas:

Tres tríadas de tipo 1 con los profesores de la otra facultad: no supervisa a los profesores y estos no se supervisan entre sí.

Doce tríadas de tipo 2: creadas por la supervisión de sus profesores y la falta de supervisión hacia el decano y los profesores de la otra facultad.

Tres tríadas de tipo 3: por las relaciones con los profesores de su facultad: los supervisa y ellos no se supervisan entre sí.

Tres tríadas de tipo 8 creadas por la supervisión indirecta del rector que lo supervisa y él supervisa a los profesores.

Tres tríadas de tipo 21 con los profesores y el decano de la otra facultad al no supervisar ni al otro decano ni a los profesores y estos últimos son supervisados por su respectivo decano.

Una tríada de tipo 24 los dos decanos son supervisados por el rector y no se supervisan entre ellos.

En resumen el rol que desempeña el decano está descrito por las siguientes frecuencias de tríadas:
3 12 3 3 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0

Haciendo este análisis a los demás actores de la red se obtienen los siguientes patrones:

El rol del otro decano está representado por:

3 12 3 3 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0

El rol del rector está representado por:

15 6 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Y los roles de los seis profesores están representados por:

17 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 2 0 0 0 0 0 0 0 0 1 0 0 0 0

17 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 2 0 0 0 0 0 0 0 0 1 0 0 0 0

17 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 2 0 0 0 0 0 0 0 0 1 0 0 0 0

17 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 2 0 0 0 0 0 0 0 0 1 0 0 0 0

17 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 2 0 0 0 0 0 0 0 0 1 0 0 0 0

17 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 2 0 0 0 0 0 0 0 0 1 0 0 0 0

La equivalencia de roles entre los actores es obvia, los decanos están involucrados en los mismos patrones. Sus relaciones con individuos específicos es muy diferente por lo que no son equivalentes

estructuralmente. Sin embargo mantienen relaciones idénticas con los otros roles presentes en la red. De la misma forma los profesores están en el mismo rol.

- En el segundo paso se calculan las distancias euclidianas entre cada tríada para determinar la equivalencia de roles.

  Para medir las diferencias entre los patrones de dos actores Hummell y Sodeur proponen el uso de la distancia euclidiana, la cual resulta útil no solo por su simplicidad sino que también es una medida utilizada para calcular disimilitudes entre nodos en otros algoritmos [20].

  Para calcular la distancia existente entre los roles del actor j y el actor i se utiliza la fórmula siguiente

$$d_{ij} = \left[\sum_{q} (t_{jq} - t_{iq})^2 \right]^{1/2} \tag{2.1}$$

  donde q varía desde 1 a 36 a través de los patrones de tríadas de la Figura 2.3. Cuando esta distancia es cero los individuos i y j tienen roles equivalentes bajo un criterio fuerte. A medida que la distancia aumenta de valor la equivalencia entre i y j disminuye.

- Por ultimo en el tercer paso se procede a agrupar los actores de acuerdo a las distancias calculadas para determinar los conjuntos de actores con roles similares.

  El proceso de agrupamiento ocurre utilizando un método de Agrupamiento Jerárquico. Este método toma inicialmente todos los nodos como grupos de un nodo y los va uniendo de dos en dos, de acuerdo a una expresión de similitud, hasta que solo quede un grupo que contiene todos los nodos. Primeramente el algoritmo de agrupamiento calcula una matriz de distancias con todas las distancias entre los nodos. Luego se crea un nuevo grupo al unir los dos nodos más cercanos de acuerdo a las distancias calculadas en la matriz. Iterativamente se van uniendo todos los nodos hasta que solo queda un nodo que contiene a todos los demás. A medida que ocurre el proceso se va creando un árbol con los nodos que se unieron como hijos y el nodo resultante como padre, donde la raíz del árbol sería el nodo que contiene todos los nodos y las hojas los nodos individuales.

2.3. Captura de requisitos

Para el desarrollo del subsistema se realizó la captura de requisitos, estos no son más que las capacidades, condiciones o cualidades que el subsistema debe cumplir y tener. Para obtener dichos requisitos

se tomó como punto de partida las entrevistas realizadas al cliente, mediante las cuales se obtuvo una descripción de lo que debía hacer el subsistema, generándose como resultado la Lista de Reserva del Producto (LRP), (Véase el Cuadro A.1 del Anexo A).

2.3.1. Requisitos funcionales

- RF1 Detectar roles.
- RF2 Registrar actores según rol.
- RF3 Establecer comunicación con el Controlador de MOCICE.
- RF4 Registrar trazas de ejecución del Subsistema de Detección de Roles.
- RF5 Leer fichero de configuración.

2.3.2. Requisitos no funcionales

Requisitos no funcionales de restricciones en el diseño y la implementación:

- RNF1 Usar Protocolo de Control de Transmisión/Protocolo de Internet (TCP/IP) para la comunicación.
- RNF2 Utilizar XML para el fichero de configuración.
- RNF3 Utilizar como lenguaje de programación C++.
- RNF4 Utilizar NetBeans como IDE.
- RNF5 Utilizar RapidSVN para el control de versiones.
- RNF6 Utilizar Rats y Valgrind para la revisión automática del código.

Requisitos no funcionales de *software*:

- RNF7 Usar Ubuntu 10.04 como sistema operativo.

- RNF8 El subsistema debe utilizar PostgresSQL como gestor de BD.
- RNF9 La documentación del código se realizará con Doxygen y se exportará en formato html y pdf.

Requisitos no funcionales de escalabilidad:

- RNF10 El subsistema debe permitir adicionar diferentes algoritmos para la detección de roles.

2.4. Historias de Usuario

Las Historias de Usuario (HU) son técnicas utilizadas para especificar los requisitos del *software*. Las mismas son escritas por los clientes como tareas que el sistema debe hacer. Su construcción depende principalmente de la habilidad que se tenga para definirlas. Sirven también para estimar tiempos de desarrollo y son la base para las pruebas funcionales [30].

2.4.1. HU Detectar roles

	Historia de Usuario
Número: ROL-01	**Nombre Historia de Usuario:** Detectar roles
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Yaritza Rodriguez Garcia, Jorge Enrique Miralles	**Iteración Asignada:** Sprint 1
Prioridad en Negocio: Muy Alta	**Puntos Estimados:** 3
Riesgo en Desarrollo: Alto	**Puntos Reales:** 3
Descripción: El subsistema se encarga de analizar la colección de correos electrónicos, con el objetivo de encontrar similitudes entre los usuarios y los agrupa según el rol en la BD.	

2.4.2. HU Registrar actores según rol

Historia de Usuario	
Número: ROL-02	**Nombre Historia de Usuario:** Registrar actores según rol.
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Jorge Enrique Miralles	**Iteración Asignada:** Sprint 2
Prioridad en Negocio: Alta	**Puntos Estimados:** 2
Riesgo en Desarrollo: Medio	**Puntos Reales:** 2
Descripción: El subsistema guarda los datos que se generan del análisis de las colecciones de correos electrónicos, para que puedan ser accedidos por el usuario posteriormente.	

2.4.3. HU Establecer comunicación con el Controlador de MOCICE

Historia de Usuario	
Número: ROL-03	**Nombre Historia de Usuario:** Establecer comunicación con el Controlador de MOCICE.
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Yaritza Rodriguez Garcia, Jorge Enrique Miralles	**Iteración Asignada:** Sprint 2
Prioridad en Negocio: Alta	**Puntos Estimados:** 2
Riesgo en Desarrollo: Medio	**Puntos Reales:** 2
Descripción: El subsistema será capaz de comunicarse con el sistema principal, el cual tiene integrado los demás subsistemas. Mediante un protocolo de comunicación el Controlador se encarga de decir al subsistema que realice la detección de roles a una colección determinada que se encuentra almacenada en la BD.	

2.4.4. HU Registrar trazas de ejecución del Subsistema de Detección de Roles

Historia de Usuario
Continúa en la próxima página

Número: ROL-04	**Nombre Historia de Usuario:** Registrar trazas de ejecución del Subsistema de Detección de Roles.
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Yaritza Rodriguez Garcia	**Iteración Asignada:** Sprint 3
Prioridad en Negocio: Media	**Puntos Estimados:** 2
Riesgo en Desarrollo: Medio	**Puntos Reales:** 2
Descripción: Se registran todos los eventos que se realizan en el subsistema, como por ejemplo cargar la BD, ejecutar alguno de los algoritmos que realizan la detección de roles, los errores que puedan ocurrir en el subsistema, etc. El subsistema debe guardar estos registros, con el objetivo de ser analizados posteriormente.	

2.4.5. HU Leer fichero de configuración

	Historia de Usuario
Número: ROL-05	**Nombre Historia de Usuario:** Leer fichero de configuración.
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Jorge Enrique Miralles	**Iteración Asignada:** Sprint 3
Prioridad en Negocio: Media	**Puntos Estimados:** 2
Riesgo en Desarrollo: Medio	**Puntos Reales:** 2
Descripción: El subsistema carga la configuración que necesita para ejecutarse, utilizando un fichero de configuración que contiene todos los parámetros (como la dirección de la BD, dirección del controlador, el método de detección de roles que se va a ejecutar, etc.) con el objetivo de que los datos de los parámetros no permanezcan estáticos y puedan cambiar según las necesidades (Véase Anexo B).	

2.5. Diseño

Durante el Diseño se crean las clases y funcionalidades que dan cumplimiento a los requisitos tanto funcionales como no funcionales. La fase inicial de elaboración se centra en la creación de una arquitectura inicial para el sistema, proporcionando un punto de inicio para el trabajo de análisis principal. Posterior-

mente se enfoca en la creación de entidades e interfases, así como en determinar las dependencias del subsistema.

2.5.1. Arquitectura propuesta

La arquitectura de un *software* puede entenderse como aquella estructura del programa que cohesiona las funcionalidades más críticas y relevantes, necesarias para el sistema, y que sirve de soporte al resto de funcionalidades finales, necesarias para el usuario [43].

La arquitectura del subsistema está basada en el estilo de arquitectura Llamada y Retorno, este enfatiza la modificabilidad y la escalabilidad. Dentro de esta familia de estilos se utiliza específicamente la arquitectura Basada en Componentes.

Se decidió el uso de esta arquitectura ya que permite que los componentes tomen formas especiales de bibliotecas que admiten registro tardío, lo que significa que pueden ser cargadas en tiempo de ejecución [44]. En este estilo las interfases están separadas de las implementaciones, y dichas interfases y sus interacciones son el centro de incumbencias en el diseño arquitectónico. Los componentes permiten que sus propiedades puedan ser descubiertas y utilizadas en tiempo de ejecución. En cuanto a las restricciones, puede admitirse que una interfaz sea implementada por múltiples componentes. Usualmente, los estados de un componente no son accesibles desde el exterior [45]. Que los componentes sean locales o distribuidos es transparente en la tecnología actual.

Teniendo en cuenta que la arquitectura seleccionada permite la adición de componentes sin necesidad de recompilar el código original, se decidió implementar como componentes aquellas funcionalidades que pudieran cambiar de comportamiento como: la comunicación con el Controlador de MOCICE debido a que el protocolo de comunicación podría cambiar, el algoritmo de detección de roles, pues una de las características que tiene el subsistema es poder añadir diferentes algoritmos para la detección de roles, lo que permite que para cada tipo de red se utilice el algoritmo más conveniente y la conexión con la base de datos si se cambiara el gestor utilizado.

Finalmente, la arquitectura quedó estructurada de la forma que muestra la figura:

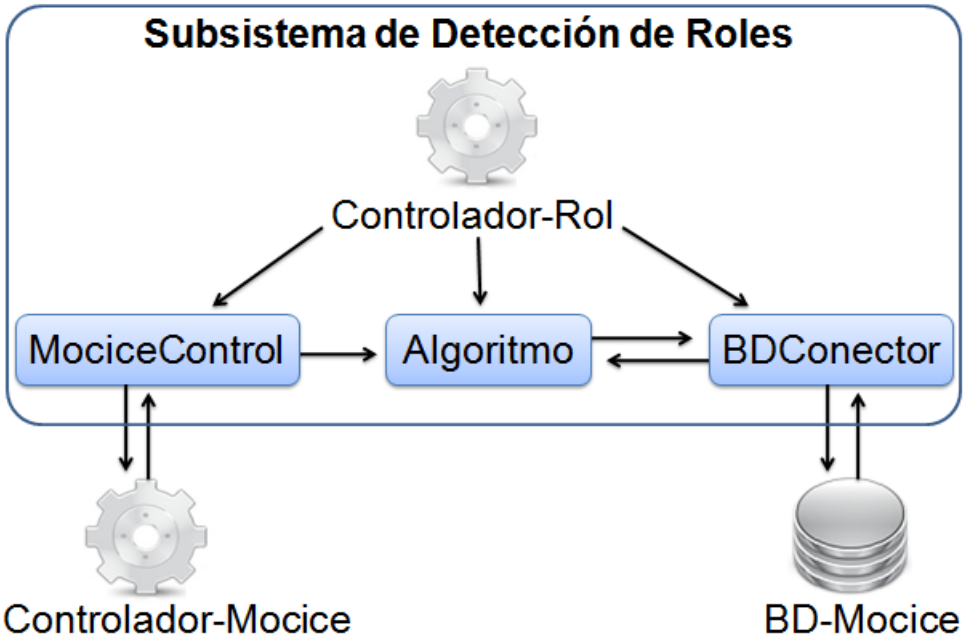

Figura 2.5: Arquitectura Propuesta.

Donde el Controlador-Rol es el núcleo del subsistema y el encargado del funcionamiento del mismo, tiene como tareas principales la lectura del archivo de configuración y el registro de los componentes que en este caso son bibliotecas dinámicas. Además, contiene las interfaces que describen el comportamiento de cada uno de los componentes. El componente MocíceControl se creó para controlar la comunicación con el Controlador de MOCICE, sus responsabilidades son registrar el subsistema en el Controlador y recibir las comunidades y lotes de correo electrónico. El componente Algoritmo se implementó para realizar el análisis de las redes de correo, en él ocurre todo el proceso de detección de roles. El componente BDConector se diseñó para controlar la conexión con la base de datos, tiene como funciones extraer los nodos (usuarios que envían o reciben correos) y aristas (relación de envío de correos entre usuarios) para formar el grafo e insertar los roles obtenidos del proceso de detección de roles y las relaciones de estos con los usuarios.

2.5.2. Patrones de diseño

Un patrón es una pareja de problema/solución con un nombre y que es aplicable a otros contextos, con una sugerencia sobre la manera de usarlo en situaciones nuevas [46]. Es decir, que tiene un nivel de

abstracción menor, estando más próximo a la implementación final. Su uso no se refleja en la estructura global del sistema.

Patrones Generales de Software para Asignación de Responsabilidades (GRASP)[1]

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos expresados en forma de patrones [47]. En la realización del diseño se utilizan tres patrones de este tipo, el experto, el creador y alta cohesión.

El **patrón experto** plantea que una funcionalidad debe pertenecer a la clase que cuenta con la información necesaria para cumplirla. Este se utilizó en la mayoría de las clases principalmente en ComponentManager y Graph. Este patrón permite conservar el encapsulamiento y la definición de clases más sencillas.

El **patrón creador** orienta la asignación de responsabilidades vinculadas con la creación de objetos. Este se utilizó en las clases ComponentManager y TriadCensus. Tiene como beneficio que posibilita la reutilización de código.

El **patrón alta cohesión** proporciona una medida de cuán relacionadas están las funcionalidades de una clase, garantizando que estas no realicen un gran trabajo. Este se empleó en la clase IAlgorithm. Facilita la comprensión del diseño y se simplifica el mantenimiento.

Patrones de diseño *Gang of Four* (GoF) [48]

Los patrones GoF (Gang-of-Four) cuyas siglas significan literalmente Banda de los Cuatro, fueron definidos en el libro Design Patterns escrito de 1991 a 1994 por los autores John Vlissides, Ralph Johnson, Erich Gamma y Richard Helm.

Los patrones GoF se clasifican según su propósito (qué hace el patrón) en:

- Creacional: creación de objetos.
- Estructural: composición de clases y objetos.

[1] GRASP: General Responsability Asignment *Software* Patterns (por sus siglas en inglés)

- De comportamiento: interacción entre objetos.

Según ámbito (clases u objetos) en:

- Clases: relaciones estáticas entre clases.
- Objetos: relaciones dinámicas entre objetos.

De los patrones de **Creación** se utilizó:

Singleton: En la clase ComponentManager que se encarga de gestionar los componentes. Este es utilizado ya que la clase antes mencionada necesita tener una sola instancia, para impedir que ocurran conflictos cuando se registran los componentes.

2.5.3. Diagrama de paquetes de clases del diseño

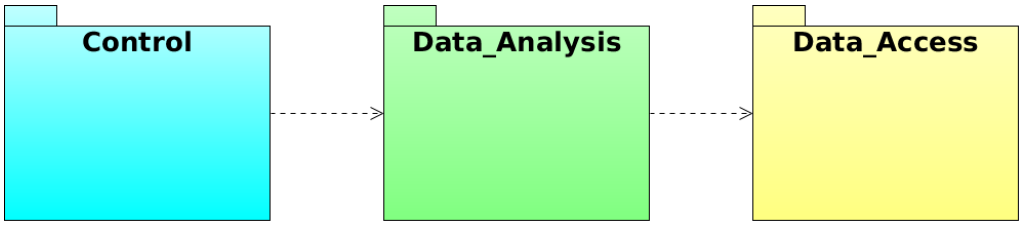

Figura 2.6: Diagrama de Paquetes de Clases del Diseño.

El siguiente diagrama está compuesto por los paquetes *Control*, *Data_Analisys* y *Data_Access*. El paquete *Control* contiene las clases que se encargan de controlar los algoritmos que serán ejecutados y que se encuentran dentro del paquete *Data_Analisys*, además contiene las clases que controlan la conexión con MOCICE y la configuración del sistema. El paquete *Data_Analisys* contiene un conjunto de algoritmos en forma de bibliotecas que se ejecutan sobre los datos a analizar, los cuales se reciben a través del paquete de *Data_Access*. Y por último el paquete *Data_Access* que contiene el conjunto de clases controladoras de BD.

2.5.4. Diagrama de clases del diseño

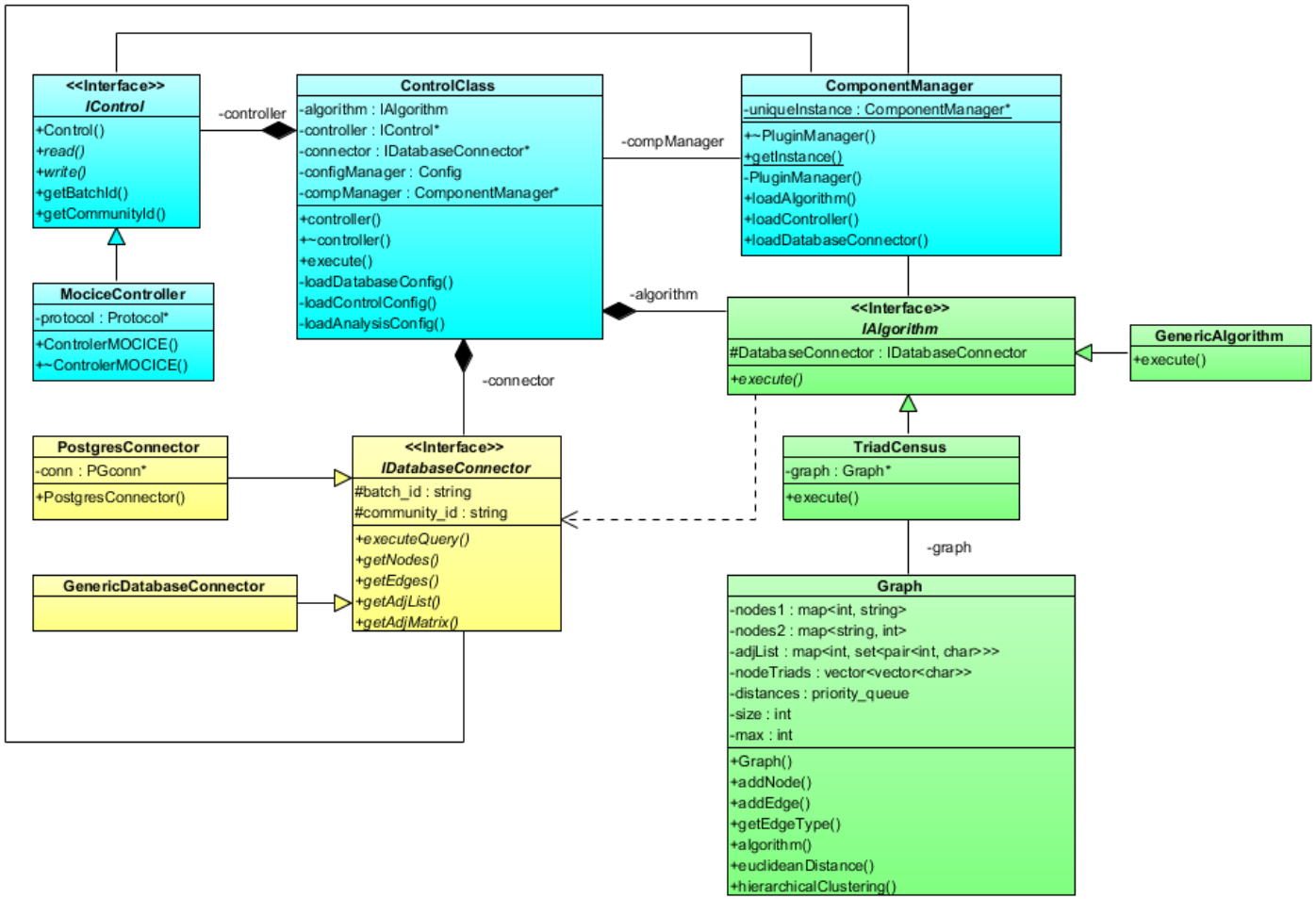

Figura 2.7: Diagrama de Clases.

2.5.5. Descripción de las clases del diseño

Nombre: IAlgorithm	
Tipo de clase: interfaz	
Atributo	**Tipo**
-DatabaseConnector	IDatabaseConnector*
Para cada funcionalidad:	
Nombre:	execute()
Descripción:	Ejecuta el algoritmo implementado en la biblioteca.

Nombre: IDatabaseConnector	
Tipo de clase: interfaz	
Atributo	**Tipo**
-batch_id	string
-community_id	string
Para cada funcionalidad:	
Nombre:	executeQuery()
Descripción:	Ejecuta una consulta predefinida.
Nombre:	getNodes()
Descripción:	Obtiene la lista de nodos (usuarios) almacenados en la BD.
Nombre:	getEdges()
Descripción:	Obtiene la lista de aristas (relaciones entre usuarios) almacenadas en la BD.
Nombre:	getAdjList()
Descripción:	Obtiene una lista de adyacencia como representación de la red social.
Nombre:	getAdjMatrix()
Descripción:	Obtiene una matriz de adyacencia como representación de la red social.

Nombre: ComponentManager
Continúa en la próxima página

Tipo de clase: entidad	
Atributo	**Tipo**
-uniqueInstance	ComponentManager*
Para cada funcionalidad:	
Nombre:	ComponentManager()
Descripción:	Constructor privado de la clase
Nombre:	ComponentManager()
Descripción:	Destructor de la clase.
Nombre:	getInstance()
Descripción:	Devuelve la única instancia de la clase.
Nombre:	loadAlgorithm(std::string path)
Descripción:	Carga el algoritmo solicitado
Nombre:	loadController(std::string path)
Descripción:	Carga el controlador solicitado
Nombre:	loadDatabaseConnector(std::string path)
Descripción:	Carga el conector de bases de datos solicitado

Nombre: ControlClass	
Tipo de clase: controladora	
Atributo	**Tipo**
-configManager	configManager *
-compManager	ComponentManager *
-algorithm	IAlgorithm*
-controller	IControl*
-connector	IDatabaseConnector*
Para cada funcionalidad:	
Nombre:	ControlClass()
Descripción:	Constructor de la clase.
Nombre:	execute()
Descripción:	Ejecuta el subsistema.

Continúa en la próxima página

Nombre:	loadDatabaseConfig()
Descripción:	Carga la configuración de la BD.
Nombre:	loadControlConfig()
Descripción:	Carga la configuración del Controlador de MOCICE.
Nombre:	loadAnalysisConfig()
Descripción:	Carga la configuración del algoritmo de detección de roles.

Nombre: IControl	
Tipo de clase: interfaz	
Atributo	**Tipo**
Para cada funcionalidad:	
Nombre:	IControl()
Descripción:	Constructor de la clase.
Nombre:	read()
Descripción:	Lee del bufer del protocolo, los datos que envía el Controlador de MOCICE.
Nombre:	write()
Descripción:	Escribe en el bufer del protocolo, la respuesta que se le envía a MOCICE.
Nombre:	getBatchId()
Descripción:	Devuelve el identificador del lote enviado por el Controlador de MOCICE.
Nombre:	getCommunityId()
Descripción:	Devuelve el identificador de la comunidad enviado por el Controlador de MOCICE.

Nombre: MociceController	
Tipo de clase: entidad	
-protocol	Protocol*
Para cada funcionalidad:	

Continúa en la próxima página

Nombre:	MociceController()
Descripción:	Constructor de la clase.
Nombre:	MociceController()
Descripción:	Destructor de la clase.
Nombre:	read()
Descripción:	Lee del bufer del protocolo, los datos que envía el Controlador de MOCICE.
Nombre:	write()
Descripción:	Escribe en el bufer del protocolo, la respuesta que se le envía a MOCICE.
Nombre:	getBatchId()
Descripción:	Devuelve el identificador del lote enviado por el Controlador de MOCICE.
Nombre:	getCommunityId()
Descripción:	Devuelve el identificador de la comunidad enviado por el Controlador de MOCICE.

Nombre: PostgresConnector	
Tipo de clase: entidad	
Atributo	**Tipo**
-conn	PGCon*
Para cada funcionalidad:	
Nombre:	PostgresConnector()
Descripción:	Constructor

Nombre: Triad Census	
Tipo de clase: entidad	
Atributo	**Tipo**
-graph	Graph*
Para cada funcionalidad:	

Continúa en la próxima página

Nombre:	Triad Census()
Descripción:	Constructor
Nombre:	Triad Census()
Descripción:	Destructor
Nombre:	execute()
Descripción:	Ejecuta el algoritmo implementado en la biblioteca.

Nombre: Graph	
Tipo de clase: entidad	
Atributo	**Tipo**
-nodes1	map<int, string>
-nodes2	map<string, int>
-adjList	map<int, set<pair<int,char>>>
-nodeTriads	vector<vector<int>>
-distances	priority_queue<double, pair<int,int>>
Para cada funcionalidad:	
Nombre:	Graph()
Descripción:	Constructor.
Nombre:	Graph()
Descripción:	Destructor.
Nombre:	addNode()
Descripción:	Adiciona un nodo al grafo.
Nombre:	addEdge()
Descripción:	Adiciona una arista al grafo.
Nombre:	algorithm()
Descripción:	Calcula las tríadas para todos los nodos.
Nombre:	euclideanDistance()
Descripción:	Calcula las distancias entre los nodos.
Nombre:	hierarchicalClustering()
Descripción:	Agrupa los nodos siguiendo un agrupamiento jerárquico.

2.5.6. Diseño de la Base de Datos

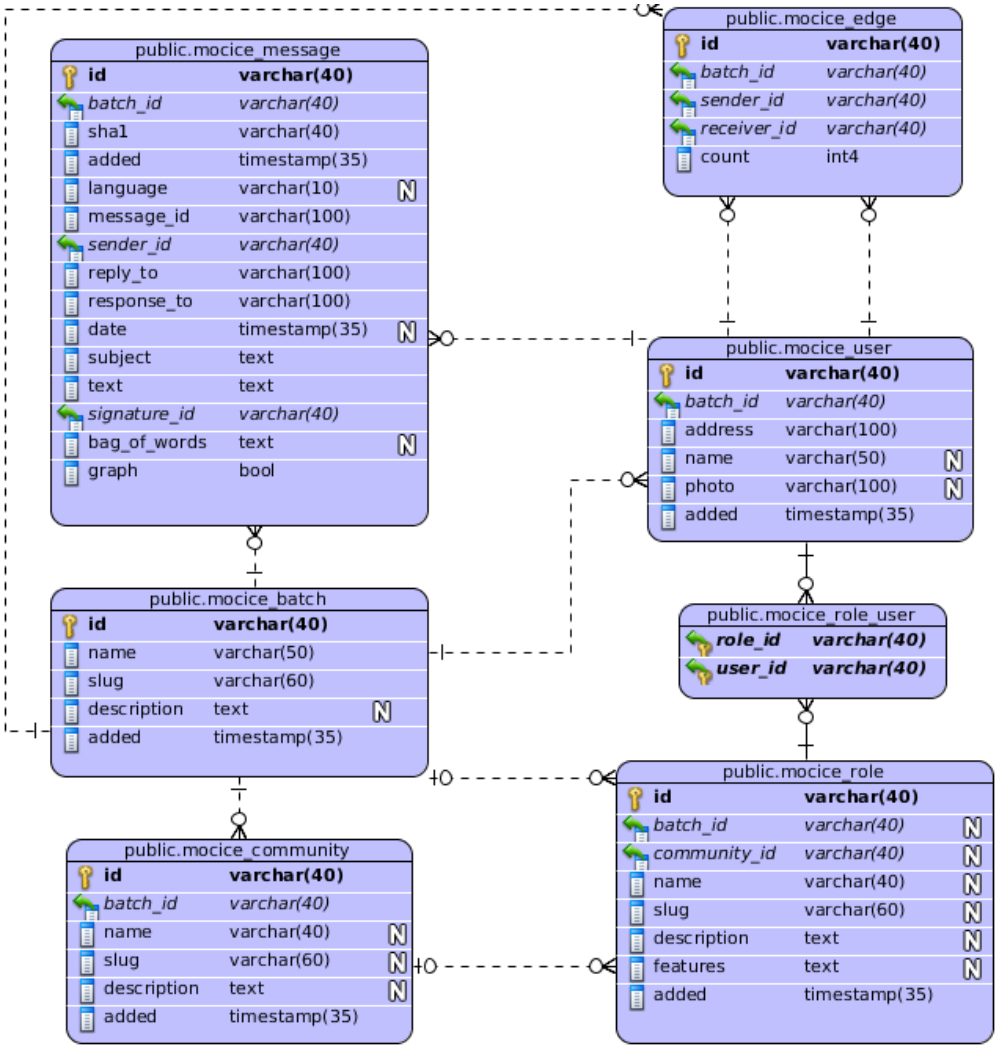

Figura 2.8: Diagrama Entidad Relación de la BD.

La imagen mostrada anteriormente representa las tablas con las que se relaciona el Subsistema de Detección de Roles directamente. La tabla *message* guarda los mensajes de correo electrónico y las características de estos. La tabla *batch* agrupa los correos electrónicos por lote. La tabla *community* se encarga de asociar los correos electrónicos de un lote a comunidades específicas. La relación existente entre la tabla *role* y *user* que se refleja en la tabla *role_user* asocia a cada usuario un rol específico. La tabla *user* almacena un registro de todos los usuarios que envían o reciben correos electrónicos y la tabla *role* guarda los roles presentes en la comunidad. La tabla *edge* guarda las relaciones de envío de correo electrónico entre los usuarios.

A continuación se realiza la descripción de las tablas de la BD que utiliza el Subsistema de Detección de Roles.

Nombre: role		
Descripción: Guarda los roles presentes en un lote de correo electrónico.		
Atributo	**Tipo**	**Descripción**
id	varchar(40)	Identificador de la tabla.
batch_id	varchar(40)	Identificador del lote de correos electrónicos.
community_id	varchar(40)	Identificador de la comunidad.
name	varchar(40)	Nombre del rol.
description	text	Breve descripción del rol.
features	text	Características que presenta el rol.
added	timestamp(35)	Tiempo en que se adicionó a la BD

Nombre: role_user		
Descripción: Tabla que se crea mediante la relación existente entre usuario y rol.		
Atributo	**Tipo**	**Descripción**
role_id	varchar(40)	Es la llave de la tabla role que viaja a la nueva tabla que se crea producto a la relación (llave foránea).
user_id	varchar(40)	Es la llave de la tabla user que viaja a la nueva tabla que se crea producto a la relación (llave foránea).

Nombre: user		
Descripción: Contiene todos los usuarios que envían o reciben mensajes.		
Atributo	**Tipo**	**Descripción**
id	varchar(40)	Identificador del user
batch_id	varchar(40)	Identificador del lote de correos electrónicos.
address	varchar(100)	Dirección de correo electrónico.
name	varchar(50)	Nombre del usuario
photo	varchar(100)	Foto del usuario
added	varchar(35)	Tiempo en que se añadió a la BD

Nombre: message		
Descripción: Almacena los documentos de correos electrónicos.		
Atributo	**Tipo**	**Descripción**
id	varchar(40)	Identificador del mensaje de correo electrónico.
batch_id	varchar(40)	Identificador del lote de correos electrónicos.
sha1	varchar(40)	Cantidad de documentos que tiene el lote.
added	timestamp(35)	Tiempo en que se añadió a la BD.
language	varchar(10)	Idioma en que esta escrito el correo.
message_id	varchar(100)	Identificador con que se envía el correo.
sender_id	varchar(40)	Identificador del usuario que envía el mensaje.
reply_to	varchar(100)	Campo del correo (*responder a:*).
date	timestamp(35)	Fecha de envío del mensaje.
subject	text	Asunto del mensaje.
text	text	Contenido del mensaje.
signature_id	varchar(40)	Identificador de la firma del mensaje.
bag_of_words	text	Palabras presentes en el contenido del mensaje
graph	bool	Cantidad de documentos que tiene el lote.

Nombre: batch		
Descripción: Lotes que almacenan el conjunto de correos electrónicos.		
Atributo	**Tipo**	**Descripción**
id	varchar(40)	Identificador del lote de correos electrónicos.
name	varchar(50)	Nombre del lote de correos electrónicos.
description	text	Descripción del lote de correos electrónicos.
added	timestamp(35)	Tiempo en que se añadió a la BD.

Nombre: community		
Descripción: Contienen los lotes a los que pertenecen los documentos.		
Atributo	**Tipo**	**Descripción**
id	varchar(40)	Identificador de la comunidad.
batch_id	varchar(40)	Identificador del lote de correos electrónicos

name	varchar(40)	Nombre de la comunidad.
description	text	Descripción de la comunidad.
added	timestamp(35)	Tiempo en que se añadió a la BD.

Nombre: edge		
Descripción: Contienen los lotes a los que pertenecen los documentos.		
Atributo	**Tipo**	**Descripción**
id	varchar(40)	Identificador de la arista.
batch_id	varchar(40)	Identificador del lote de correos electrónicos.
sender_id	varchar(40)	Identificador del usuario que envía el mensaje.
recelver_id	varchar(40)	Identificador del usuario que recibe el mensaje.
count	int4	Cantidad de mensajes enviados entre dos usuarios.

Conclusiones parciales

El adecuado levantamiento de requisitos y su posterior traducción a historias de usuarios permite que la propuesta de solución para el Subsistema de Detección de Roles de MOCICE se ajuste a las necesidades del cliente.

Se decidió utilizar una arquitectura basada en componentes ya que brinda grandes ventajas a la solución, al poder añadir nuevos componentes como: algoritmos, conectores de bases de datos y subsistemas controladores, sin necesidad de realizar grandes cambios en la implementación.

El uso de los patrones de diseño definidos posibilita que las clases diseñadas estén estructuradas con bajo nivel de complejidad y alto grado de abstracción.

Capítulo 3

Implementación y validación del Subsistema de Detección de Roles

En este capítulo se estructura y muestra en detalle el modelo de implementación; se revisa el código de la aplicación en busca de vulnerabilidades y se llevan a cabo distintas pruebas como pueden ser pruebas de funcionamiento, de integración, de rendimiento y de estrés que garantizan la calidad y seguridad de la aplicación.

3.1. Modelo de implementación

Para iniciar la implementación se comienza con el resultado del diseño y se implementa el sistema en término de componentes: ficheros de código fuente, *scripts*, ficheros de código binario, ejecutables y similares. La mayor parte de la arquitectura del sistema es capturada durante el diseño, siendo el objetivo general de la implementación desarrollar la arquitectura y el sistema como un todo. El modelo de implementación describe cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje de programación utilizado, y cómo depende un componente de otro [30].

3.1.1. Plan de Entrega

Se recogen las iteraciones a realizar con sus características, además del orden de las historias de usuario con la planificación estimada para su implementación.

Release	**Descripción de la iteración**	**Orden de la HU a implementar**	**Duración total**
Iteración 1	En esta iteración se desarrollará la HU con prioridad muy alta. Con esta el subsistema será capaz de realizar la detección de roles.	ROL-01	3 semanas
Iteración 2	En esta iteración se desarrollarán las HU que tengan prioridad alta. Con estas el subsistema será capaz de establecer comunicación con MOCICE y guardar en la BD los actores por rol.	ROL-02, ROL-03	4 semanas
Iteración 3	En esta iteración se desarrollarán las HU que tengan prioridad media. Con estas el subsistema registrará las trazas del subsistema y verificará fichero de configuración.	ROL-04, ROL-05	4 semanas

Cuadro 3.1: Plan de Entrega.

3.1.2. Diagrama de componentes

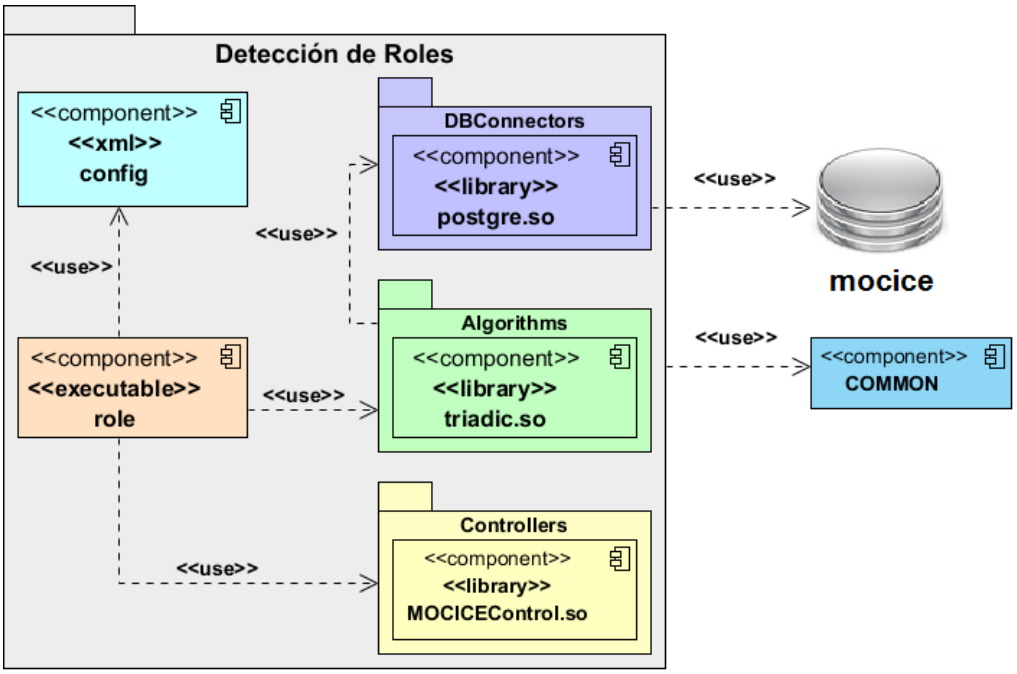

Figura 3.1: Diagrama de Componentes.

En el diagrama de componentes el paquete detección de roles, utiliza la biblioteca MOCIC-COMMON para la integración con MOCICE a través de la cual se establece la comunicación con el Controlador de MOCICE. Dentro de este paquete están los siguientes componentes:

- El componente ejecutable, que se encarga de leer las configuraciones necesarias del archivo XML para la ejecución del proceso de análisis y controla el funcionamiento del resto de los componentes.
- El archivo de configuración XML, que guarda todas las configuraciones que necesita el subsistema.
- Los componentes del paquete Controllers, que se encargan de establecer la comunicación con el Controlador de MOCICE, recibir la información de los datos a analizar y enviar un mensaje una vez terminado el análisis.
- Los componentes del paquete DBConnectors, que se encargan de facilitarle la información a los algoritmos, así como guardar en la BD la información que estos envían.

- Los componentes del paquete Algorithms, que son los encargados de realizar la detección de roles, los que reciben y envían información a través del conector de la BD.
- La BD mocice, que contiene los lotes de correo electrónico y los resultados del proceso de detección de roles.

3.1.3. Diagrama de despliegue

Figura 3.2: Diagrama de Despliegue.

La Figura 3.2 muestra el diagrama de despliegue del Subsistema de Detección de Roles, el cual se ejecuta en un nodo independiente al Controlador de MOCICE. El controlador es el encargado de indicarle al subsistema qué lote de correo electrónico se desea analizar, comunicándose mediante el protocolo TCP/IP, logrando así la integración con el sistema completo. Además, con el protocolo TCP/IP se establece comunicación con la BD, lugar donde están almacenados los lotes de correo electrónico que se analizarán y donde serán guardados los resultados luego de realizado el proceso de detección de roles.

3.2. Revisión de código

Una revisión de código consiste en el análisis estático del código fuente de una aplicación, con el objetivo de detectar y eliminar vulnerabilidades comunes en todo proceso de desarrollo de *software*. Esto significa que se buscan fallas en cuanto a estándares o buenas prácticas utilizados en la codificación. De esta forma, se mejora la seguridad del *software* desarrollado desde su misma implementación.

Para efectuar esta actividad pueden usarse herramientas como RATS y Valgrind que automaticen el

proceso de revisión de código, las cuales realizan en cuestión de segundos lo que una persona haría en varios días o semanas, en dependencia del tamaño del proyecto a revisar.

3.2.1. Revisión con la herramienta RATS

En la primera revisión realizada con RATS, se analizaron en total 821 líneas de código de las cuales se detectaron tres líneas con vulnerabilidades de gravedad alta, una con gravedad media y dos con gravedad baja.

Principales problemas encontrados:

- Uso de la función sprintf() la cual no garantiza que el espacio del parámetro destino sea suficiente para almacenar la información del parámetro cadena, siendo posible un desbordamiento de buffer. No verifica que las cadenas terminen en NULL lo que puede provocar un desbordamiento de buffer.
- Uso de la función unmask() que puede crear archivos con privilegios indebidos, proporcionando una vulnerabilidad de seguridad al subsistema.
- Uso de la función syslog() a la cual si se le da como parámetro una cadena de caracteres con un tamaño mayor a 1024 bytes, provoca problemas en la escritura del registro de trazas.

De acuerdo con los problemas encontrados se tomaron las siguientes medidas:

- Uso de la clase string del lenguaje C++ para manejar las cadenas. Esta clase permite realizar validaciones para evitar desbordamientos de buffer.
- Verificación de la longitud de las cadenas que se pasan por parámetro al syslog.

En la segunda revisión realizada con RATS, luego de la implementación de las medidas tomadas, se analizaron en total 960 líneas de código, donde solo se detectó una vulnerabilidad, de gravedad alta. Hay que señalar, que dicha vulnerabilidad surge debido al uso de la función unmask(). Esta no ha sido eliminada debido a que es necesario su uso para dar permisos a la ejecución del subsistema. No obstante,

se ha controlado el ambiente en el que se ejecuta para evitar comportamientos indeseados por parte del *software*.

3.2.2. Revisión con la herramienta Valgrind

En la primera revisión realizada con la herramienta Memcheck de Valgrind se detectaron cinco problemas que causaban fugas de memoria con un total de 684367 bytes perdidos directa e indirectamente.

Principales problemas encontrados:

- Creación de objetos cuya reserva de memoria no era liberada una vez terminada la ejecución.
- Declaración de estructuras de datos que no eran liberadas después de ser usadas.

De acuerdo con los problemas de fuga de memoria encontrados se tomaron las siguientes medidas:

- Optimizar el uso de las estructuras de datos, liberándolas después de ser usadas o reutilizándolas en otra funcionalidad.
- Creación de destructores más eficaces que permitieron eliminar toda la memoria reservada por un objeto de su clase.

En la segunda revisión realizada con la herramienta Valgrind se detectó 1 problema que causaba una fuga de memoria de 160 bytes perdidos directa e indirectamente. No es posible solucionar esta fuga de memoria pues está presente en la biblioteca *pq*, la cual se encarga de manejar las conexiones a BD de PostgreSQL.

3.3. Pruebas de *software*

Durante las pruebas se verifica el resultado de la implementación probando cada construcción, incluyendo tanto construcciones internas, como intermedias, así como las versiones finales del sistema a

ser entregadas a terceros [30]. A continuación se muestran las pruebas realizadas a la aplicación, para garantizar su correcto funcionamiento y seguridad, como parte del proceso de aseguramiento de la calidad.

3.3.1. Pruebas de funcionamiento

Las pruebas de funcionamiento se realizan para verificar que las funcionalidades del sistema se ejecuten correctamente. Para ello se realizan tantas pruebas como sean necesarias por HU.

Diseño de los Casos de Pruebas

Casos de Prueba de Funcionamiento	
Código de caso de prueba: ROL-01-1	**Nombre de Historia de Usuario:** Detectar Roles.
Nombre de la persona que realiza la prueba: Jorge Enrique Miralles Betancourt	
Descripción de la Prueba: Comprobar que el subsistema al recibir un lote del controlador realice el proceso de detección de roles.	
Condiciones de ejecución: Existencia del ejecutable “role” en la dirección /etc/init.d/role.	
Entradas / **Pasos de ejecución:** Se procede a ejecutar el *script* del demonio de la aplicación que se encuentra en /etc/init.d/role. Se recibe del Controlador un identificador del lote de correo al cual se le identifican los roles.	
Resultados esperados: Se obtiene una lista con los usuarios según el rol desempeñado.	
Evaluación de la Prueba: Prueba Satisfactoria	

Casos de Prueba de Funcionamiento	
Código de caso de prueba: ROL-02-1	**Nombre de Historia de Usuario:** Registrar actores según rol.
Nombre de la persona que realiza la prueba: Yaritza Rodriguez Garcia	
Descripción de la Prueba: Comprobar que el subsistema almacene los usuarios y roles en la BD.	
Continúa en la próxima página	

Condiciones de ejecución: Tiene que haberse ejecutado la HU ROL-01 sin haber dado ningún error, tiene que estar activa la conexión con la BD.
Entradas / **Pasos de ejecución:** Se recibe del algoritmo de detección de roles una lista con los usuarios por roles y se almacena en la BD.
Resultados esperados: Se registran en la BD los actores según el rol que desempeñan.
Evaluación de la Prueba: Prueba Satisfactoria.

Casos de Prueba de Funcionamiento	
Código de caso de prueba: ROL-04-1	**Nombre de Historia de Usuario:** Registrar trazas de ejecución del Subsistema de Detección de Roles.
Nombre de la persona que realiza la prueba: Jorge Enrique Miralles Betancourt	
Descripción de la Prueba: Comprobar que el subsistema registra las trazas del proceso de detección de roles.	
Condiciones de ejecución: Existencia del ejecutable “role” en la dirección /etc/init.d/role y del archivo de configuración.	
Entradas / **Pasos de ejecución:** Se procede a ejecutar el *script* del demonio de la aplicación que se encuentra en /etc/init.d/role y luego se ejecuta el Subsistema de Detección de Roles para un lote de correo electrónico.	
Resultados esperados: Se guardó en el fichero que se encuentra en la dirección /var/log/role.log la información del inicio y fin del proceso de detección, así como cualquier suceso que ocurra durante el mismo.	
Evaluación de la Prueba: Prueba Satisfactoria.	

Casos de Prueba de Funcionamiento	
Código de caso de prueba: ROL-05-1	**Nombre de Historia de Usuario:** Leer fichero de configuración.
Nombre de la persona que realiza la prueba: Yaritza Rodriguez Garcia	
Descripción de la Prueba: Comprobar que el subsistema obtenga correctamente la configuración del controlador.	
Continúa en la próxima página	

Condiciones de ejecución: Existencia del archivo de configuración
Entradas / **Pasos de ejecución:** Ejecución de la funcionalidad loadControlConfig().
Resultados esperados: Se lee la configuración del controlador y se crea el objeto de tipo Control.
Evaluación de la Prueba: Prueba Satisfactoria.

Casos de Prueba de Funcionamiento	
Código de caso de prueba: ROL-05-2	**Nombre de Historia de Usuario:** Leer fichero de configuración.
Nombre de la persona que realiza la prueba: Jorge Enrique Miralles Betancourt	
Descripción de la Prueba: Comprobar que el subsistema obtenga correctamente la configuración de la BD.	
Condiciones de ejecución: Existencia del archivo de configuración y de al menos un plugin de BD	
Entradas / **Pasos de ejecución:** Ejecución de la funcionalidad loadDatabaseConfig().	
Resultados esperados: Se lee la configuración de la BD y se crea un objeto de IDatabaseConnector.	
Evaluación de la Prueba: Prueba Satisfactoria.	

Casos de Prueba de Funcionamiento	
Código de caso de prueba: ROL-05-3	**Nombre de Historia de Usuario:** Leer fichero de configuración.
Nombre de la persona que realiza la prueba: Yaritza Rodriguez Garcia	
Descripción de la Prueba: Comprobar que el subsistema obtenga correctamente la configuración del algoritmo que se utilizará para hacer la detección de roles.	
Condiciones de ejecución: Existencia del archivo de configuración y de al menos un plugin de algoritmos	
Entradas / **Pasos de ejecución:** Ejecución de la funcionalidad loadAnalysisConfig().	
Resultados esperados: Se lee la configuración del algoritmo que se va a utilizar y se crea un objeto de tipo IPlugins.	
Continúa en la próxima página	

Evaluación de la Prueba: Prueba Satisfactoria.

Resultados de las pruebas

Durante la primera iteración de las pruebas de funcionamiento realizadas, se detectaron varias irregularidades en el funcionamiento del subsistema, las cuales se describen a continuación:

- En la funcionalidad Detectar Roles se encontró una no conformidad, debido a que el tiempo de procesamiento era muy prolongado.
- En la funcionalidad Registrar actores según rol se detectó una no conformidad, debido a que guardar los resultados del proceso de detección de roles demoraba demasiado tiempo.
- En la funcionalidad Registrar trazas de ejecución, se encontró una no conformidad, debido a que no se registraban todos los ocurridos durante el proceso de detección de roles.

Para dar solución a las no conformidades encontradas se tomaron varias medidas que ayudaron a erradicarlas, como:

Se reestructuraron los criterios de clasificación del algoritmo, logrando una disminución en el tiempo de procesamiento. Se cambió la forma de guardar los resultados en la BD. Se incluyeron en el registro de las trazas las fases del proceso que antes no se registraban. Una vez tomadas las medidas necesarias, se realizó una segunda iteración de prueba, en la cual no se detectaron anomalías en las funcionalidades del subsistema.

3.3.2. Pruebas de integración

Las pruebas de integración son la fase de prueba de *software* en la cual módulos individuales son combinados y probados como un grupo. En este caso se realizaron pruebas de integración entre el Subsistemas de Detección de Roles y el Controlador de MOCICE.

Casos de Prueba de Integración	
Código de caso de prueba: ROL-03-1	**Nombre de Historia de Usuario:** Establecer comunicación con el Controlador de MOCICE.
Nombre de la persona que realiza la prueba: Yaritza Rodriguez Garcia	
Descripción de la Prueba: Comprobar que el subsistema reciba la información del controlador correctamente.	
Condiciones de ejecución: Existencia del ejecutable “role” en la dirección /etc/init.d/role y del archivo de configuración. El controlador debe estar activo.	
Entradas / **Pasos de ejecución:** Se procede a ejecutar el *script* del demonio de la aplicación, se lee el archivo de configuración que contiene la dirección del controlador y se envía un mensaje haciéndole saber que el Subsistema de Detección de Roles está disponible para analizar un lote de correo electrónico y luego se espera un mensaje del controlador, especificando el lote a analizar.	
Resultados esperados: Se recibe un identificador de un lote de correo electrónico enviado por el Controlador.	
Evaluación de la Prueba: Prueba Satisfactoria	

Casos de Prueba de Integración	
Código de caso de prueba: ROL-03-2	**Nombre de Historia de Usuario:** Establecer comunicación con el Controlador de MOCICE.
Nombre de la persona que realiza la prueba: Yaritza Rodriguez Garcia	
Descripción de la Prueba: Comprobar que el subsistema envía la información al controlador correctamente.	
Condiciones de ejecución: Existencia del ejecutable “role” en la dirección /etc/init.d/role y del archivo de configuración. El controlador debe estar activo.	
Continúa en la próxima página	

Entradas / **Pasos de ejecución:** Se procede a ejecutar el *script* del demonio de la aplicación, se lee el archivo de configuración que contiene la dirección del controlador y se envía un mensaje haciéndole saber que el Subsistema de Detección de Roles ha terminado el proceso y está disponible nuevamente para analizar otro lote de correos electrónicos.
Resultados esperados: Se envía al controlador un mensaje para que este sepa que se realizó la detección de roles de forma exitosa.
Evaluación de la Prueba: Prueba Satisfactoria

Durante la primera iteración de las pruebas de integración realizadas al Subsistema de Detección de Roles, se encontró una no conformidad debido a que la comunicación con el Controlador de MOCICE no se establecía correctamente. Para dar solución al problema detectado se tomaron medidas que ayudaron a eliminarlo. Una vez realizada la segunda iteración de pruebas no se detectó problema alguno, funcionando correctamente la integración con MOCICE.

3.3.3. Pruebas de rendimiento y de estrés

Pruebas de rendimiento

En la Ingeniería del *Software*, las pruebas de rendimiento son aquellas que se realizan para determinar qué tan rápido un sistema realiza una tarea bajo ciertas condiciones de trabajo. Estas pruebas también son utilizadas para validar y verificar diferentes aspectos de la calidad de *software*, como por ejemplo, fiabilidad y el buen uso de los recursos.

Prueba de estrés

Las pruebas de estrés se realizan para determinar la solidez de la aplicación en los momentos de carga extrema, poniendo demanda en el sistema y luego midiendo su respuesta. Permite identificar cuellos de botella, reduciendo el riesgo de caída del sistema, conocer los limites que soporta el sistema, etc. Todo esto con el objetivo de mejorar el rendimiento, escalabilidad y estabilidad del sistema.

La máquina de trabajo utilizada para ejecutar las pruebas posee las siguientes características:

- Procesador: Core i3 2.27 GHz
- Memoria RAM: 4 GB
- Almacenamiento:500 GB

Resultados prueba de rendimiento

Para la realización de las pruebas de rendimiento se utilizaron 3 colecciones de datos, dos de ellas generadas aleatoriamente utilizando una funcionalidad del *software* UCINET [10] y la otra es parte de la colección de correos de Enron. Los datos de estas colecciones se muestran a continuación:

Colecciones de correo electrónico			
Nombre	255 nodos	1000 nodos	Enron-Flat
Cantidad de nodos	255	1000	9514
Cantidad de aristas	6477	102655	6477
Densidad	0.10	0.10	0.10

Durante las pruebas se comparó el algoritmo implementado con otros algoritmos de detección de roles, arrojando como resultado las siguiente tabla.

Resultados de la comparación de los algoritmos			
Algoritmo	**Consumo CPU**	**Consumo Memoria**	**Tiempo en seg.**
Caso de Prueba Grafo Aleatorio 255 nodos			
Triad Census	90 %	7 MB	3
Rege	80 %	41 MB	10
Maxsim	75 %	41 MB	6
Concor	65 %	28.48 MB	3
Caso de Prueba Grafo Aleatorio 1000 nodos			
Triad Census	85 %	15 MB	191
Rege	85 %	62 MB	240
Maxsim	85 %	41 MB	165

Continúa en la próxima página

Concor	70 %	30.72 MB	50
Caso de Prueba Enron Flat 9514 nodos			
Triad Census	95 %	691 MB	420
Rege	95 %	1423 MB	+3600
Maxsim	90 %	364 MB	+3600
Concor	95 %	758 MB	+3600

A partir de la tabla anterior se puede apreciar cómo el algoritmo implementado mantiene un rendimiento en cuanto a uso del CPU, Memoria y tiempo aceptable y en muchos de los casos tiene un mejor rendimiento.

Vale destacar que las implementaciones probadas de los otros algoritmos no pueden analizar volúmenes de más de 9000 nodos pues el tiempo que demora la ejecución es relativamente elevado.

Resultados prueba de estrés

Mediante esta prueba se pudo detectar que durante la etapa de clasificación (cálculo de las disimilitudes entre los nodos del grafo) del proceso de detección de roles es el momento en que el algoritmo consume la mayor cantidad de memoria. Debido al uso de la estructura de datos cola con prioridad, donde se almacenan las disimilitudes que existen entre cada par de nodos. Es posible calcular la cantidad de memoria necesaria durante el momento de consumo pico del algoritmo haciendo uso de la fórmula descrita a continuación:

$$n\,(n-1)\,8 \tag{3.1}$$

Donde n es la cantidad de nodos de la red y 8 es la cantidad de bytes reservados sobre 2, utilizado para almacenar la distancia existente de un nodo a otro es decir un double y dos enteros.

Por ejemplo:
Para una red de 9500 nodos se necesitan guardar $9500\,(9500-1)\,8 = 721924000$ bytes lo que equivaldría a 688 MB de memoria aproximadamente. Por lo que con 2 GB de memoria asignada a la ejecución del subsistema se podrían analizar aproximadamente 15000 nodos.

Conclusiones parciales

La creación del diagrama de componentes permite obtener la estructura física del subsistema y las dependencias entre el ejecutable, las bibliotecas y la base de datos.

La implementación del Subsistema de Detección de Roles, aporta a MOCICE la capacidad de detectar los roles existentes en las redes de correo electrónico.

El resultado de las pruebas valida la correcta ejecución de las funcionalidades del subsistema y posibilita que el mismo sea aprobado para su liberación.

Conclusiones

Al culminar el desarrollo del presente trabajo de diploma se arribó a las siguientes conclusiones:

La investigación realizada, evidenció la necesidad de implementar un subsistema propio utilizando el algoritmo Triad Census, que resultó ser el más factible para realizar la detección de roles.

Con la implementación del Subsistema de Detección de Roles, MOCICE es capaz de detectar los roles que desempeñan los usuarios en una red de correo electrónico, lo que permite tener una base para determinar y comparar los comportamientos de los usuarios en la red.

La realización de pruebas de funcionamiento, integración y rendimiento permitió obtener una aplicación con una mejor calidad y estabilidad, garantizando un mejor funcionamiento de la misma.

Recomendaciones

Al concluir el presente trabajo de diploma, se realizan las siguientes recomendaciones con el fin de mejorar el funcionamiento de la aplicación:

- Se recomienda la investigación e implementación del modelo *Role Author Recipent Topic* (RART), pues este algoritmo parece tener una mayor eficiencia que el implementado.
- Reimplementar el algoritmo haciendo uso de programación concurrente. Con esto se reduce el tiempo de análisis.
- Implementar un módulo que permita visualizar los roles detectados.

Lista de acrónimos

BD Base de Datos

BSD Distribución de Software Berkeley

CPU Unidad Central de Procesamiento

GoF *Gang of Four*

GRASP Patrones Generales de Software para Asignación de Responsabilidades

GVT *GeoViz Tollkit*

HU Historias de Usuario

HTML *Hyper Text Markup Language*

IDE Entorno de Desarrollo Integrado

IP Protocolo de Internet

JUNG *Java Universal Network/Graph Framework*

LRP Lista de Reserva del Producto

MOCICE Motor de Categorización Inteligente de Contenido para Correo Electrónico

PHP Preprocesador de Hipertexto

RART *Role Author Recipent Topic*

RATS *Rough Auditing Tool for Security*

SNA Análisis de Redes Sociales

SQL Lenguaje de Consulta Estructurado

SSL Capa de Conexión Segura

SXP ScrumXP

TCP/IP Protocolo de Control de Transmisión/Protocolo de Internet

UCI Universidad de las Ciencias Informáticas

XP Programación eXtrema

XML Lenguaje de Marcado Extensible

Referencias Bibliográficas

[1] H. Hummell and W. Sodeur, "Strukturbeschreibung von positionen in sozialen beziehungsnetzen," *Methoden der netzwerkanalyse*, 1987.

[2] R. Merton, "Social structure and anomie," *American sociological review*, vol. 3, no. 5, pp. 672–682, 1938.

[3] R. Cragun, D. Cragun, *et al.*, *Introduction to Sociology*. Blacksleet River, 2006.

[4] H. Tischler, *Introduction to sociology*. Wadsworth Pub Co, 2010.

[5] H. Ebaugh, *Becoming an ex: The process of role exit*. University of Chicago Press, 1988.

[6] K. Faust and S. Wasserman, "Social network analysis: Methods and applications," 1994.

[7] B. Biddle, "Recent development in role theory," *Annual review of sociology*, pp. 67–92, 1986.

[8] Orgnet, "Inflow," *Consulta: 18 de febrero 2012. Disponible en: http://www.orgnet.com/inflow3.html*.

[9] W. Luo, A. MacEachren, P. Yin, and F. Hardisty, "Spatial-social network visualization for exploratory data analysis," in *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Location-Based Social Networks*, p. 3, ACM, 2011.

[10] S. Borgatti, M. Everett, and L. Freeman, "Ucinet for windows: Software for social network analysis," *Harvard Analytic Technologies*, vol. 2006, 2002.

[11] V. Batagelj and A. Mrvar, "Pajek-program for large network analysis," *Connections*, vol. 21, no. 2, pp. 47–57, 1998.

[12] J. Lerner, "Role assignments," *Network analysis*, pp. 216–252, 2005.

[13] R. Diestel, "Graph theory," 2000.

[14] F. Lorrain and H. White, "Structural equivalence of individuals in social networks," *Journal of mathematical sociology*, vol. 1, no. 1, pp. 49–80, 1971.

[15] M. Everett and S. Borgatti, “Role colouring a graph,” *Mathematical Social Sciences*, vol. 21, no. 2, pp. 183–188, 1991.

[16] M. Everett and S. Borgatti, “Regular equivalence: General theory,” *Journal of Mathematical Sociology*, vol. 19, no. 1, pp. 29–52, 1994.

[17] L. Sailer, “Structural equivalence: Meaning and definition, computation and application,” *Social Networks*, vol. 1, no. 1, pp. 73–90, 1979.

[18] D. White and K. Reitz, “Graph and semigroup homomorphisms on networks of relations,” *Social Networks*, vol. 5, no. 2, pp. 193–234, 1983.

[19] U. Brandes and J. Lerner, “Role-equivalent actors in networks–group structures beyond dense communities,” 2007.

[20] M. Nunkesser and D. Sawitzki, “Blockmodels,” *Network Analysis*, pp. 253–292, 2005.

[21] R. Breiger, S. Boorman, and P. Arabie, “An algorithm for clustering relational data with applications to social network analysis and comparison with multidimensional scaling,” *Journal of Mathematical Psychology*, vol. 12, no. 3, 1975.

[22] P. Sneath, R. Sokal, *et al.*, *Numerical taxonomy. The principles and practice of numerical classification.* 1973.

[23] J. Rice, *Mathematical statistics and data analysis*. Thomson Learning, 2006.

[24] P. Doreian, “Using multiple network analytic tools for a single social network,” *Social Networks*, vol. 10, no. 4, pp. 287–312, 1988.

[25] K. Faust, “Comparison of methods for positional analysis: Structural and general equivalences,” *Social Networks*, vol. 10, no. 4, pp. 313–341, 1988.

[26] F. Sim and M. Schwartz, “Does concor find positions,” *Unpublished manuscript*, 1979.

[27] M. Everett and S. Borgatti, “Calculating role similarities: An algorithm that helps determine the orbits of a graph,” *Social networks*, vol. 10, no. 1, pp. 77–91, 1988.

[28] W. D. R., “Rege: A regular graph equivalence algorithm for computing role distances prior to blockmodelling.” University of California, Irvine.

[29] A. McCallum, X. Wang, and A. Corrada-Emmanuel, “Topic and role discovery in social networks with experiments on enron and academic email,” *Journal of Artificial Intelligence Research*, vol. 30, no. 1, pp. 249–272, 2007.

[30] G. M. Penalver Romero, “Metodogía ágil para proyectos de software libre,” Master’s thesis, Universidad de las Ciencias Informáticas, 2008.

[31] R. F. Céspedes and S. P. García, “Propuesta de un expediente, para los proyectos productivos del polo de software libre, de la facultad 10,” Master’s thesis, Universidad de las Ciencias Informáticas, 2008.

[32] B. Stroustrup and S. T. B. Online, *The C++ programming language*, vol. 3. Addison-Wesley Reading, MA, 1997.

[33] C. Oracle, “Netbeans,” *Consulta: 15 octubre 2011. Disponible en: http://www.netbeans.org*.

[34] T. P. G. D. Group, “Postgresql,” *Consulta: 23 febrero 2012. Disponible en: http://www.postgresql.org*.

[35] T. P. D. Team, “Pgadmin documentation,” *Consulta: 23 de febrero 2012. Disponible en: http://www.pgadmin.org/*.

[36] V. Paradigm, “Visual paradigm for uml,” *Consulta: 15 octubre 2011. Disponible en: http://www.visual-paradigm.com*.

[37] C. Cowan, “Software security for open-source systems,” *Security & Privacy, IEEE*, vol. 1, no. 1, pp. 38–45, 2003.

[38] J. Seward, N. Nethercote, and J. Fitzhardinge, “Valgrind, an open-source memory debugger for x86-gnu/linux,” *Consulta: 25 abril 2012. Disponible en: http://www.ukuug.org/events/linux2002/papers/html/valgrind*.

[39] C. de autores, “Introducción a lyx,” 2009.

[40] D. van Heesch, “Doxygen home page,” *Consulta: 15 octubre 2011. Disponible en: http://www.doxygen.org*.

[41] J. Ivar, C. Magnus, J. Patrik, and Ö. Gunnar, “Object-oriented software engineering, a use case driven approach,” 1992.

[42] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-oriented modeling and design*, vol. 38. Prentice hall, 1991.

[43] I. Jacobson, G. Booch, and J. Rumbaugh, *El proceso unificado de desarrollo de software*, vol. 464. Addison Wesley, 2000.

[44] C. Szyperski, "Component-oriented programming: a refined variation on object-oriented programming," *The Oberon Tribune*, vol. 1, no. 2, p. 1, 1995.

[45] C. Szyperski, D. Gruntz, and S. Murer, *Component software: beyond object-oriented programming*. Addison-Wesley Professional, 2002.

[46] L. Craig, "Uml y patrones. introducción al análisis y diseño orientado a objetos," *Pretince Hall, Hispanoamérica México*, 1999.

[47] C. Larman, "Uml y patrones," *Prentice-Hall*, 2002.

[48] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design patterns: Abstraction and reuse of object-oriented design," *ECOOP'93—Object-Oriented Programming*, pp. 406–431, 1993.

Bibliografía

[BE93] S.P. Borgatti and M.G. Everett. Two algorithms for computing regular equivalence. *Social Networks*, 15(4):361–376, 1993.

[Bel78] R. Bellman. *An introduction to artificial intelligence: Can computers think?* Boyd & Fraser Pub. Co., 1978.

[BK08] R. Brendel and H. Krawczyk. Detection of roles of actors in social networks using the properties of actorsñeighborhood structure. In *Dependability of Computer Systems, 2008. DepCos-RELCOMEX'08. Third International Conference on*, pages 163–170. IEEE, 2008.

[BL05] U. Brandes and J. Lerner. Structural similarity in graphs. *Algorithms and Computation*, pages 184–195, 2005.

[BL10] U. Brandes and J. Lerner. Structural similarity: spectral methods for relaxed blockmodeling. *Journal of classification*, 27(3):279–306, 2010.

[Bur90] R.S. Burt. Detecting role equivalence. *Social Networks*, 12(1):83–97, 1990.

[CMCM09] G. Chin, A. Marquez, S. Choudhury, and K. Maschhoff. Implementing and evaluating multithreaded triad census algorithms on the cray xmt. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–9. IEEE, 2009.

[DNMB11] W. De Nooy, A. Mrvar, and V. Batagelj. *Exploratory social network analysis with Pajek*, volume 34. Cambridge Univ Pr, 2011.

[Dor87] P. Doreian. Measuring regular equivalence in symmetric structures. *Social Networks*, 9(2):89–107, 1987.

[EB02] M. Everett and S. Borgatti. Computing regular equivalence: practical and theoretical issues. *Metodoloski zvezki*, 17:31–42, 2002.

[FFBS04] E. Freeman, E. Freeman, B. Bates, and K. Sierra. *Head first design patterns*. O'Reilly & Associates, Inc., 2004.

[FLL07] T.F. Fan, C.J. Liau, and T.Y. Lin. Positional analysis in fuzzy social networks. In *Granular Computing, 2007. GRC 2007. IEEE International Conference on*, pages 423–423. IEEE, 2007.

[GD05] L. Getoor and C.P. Diehl. Link mining: a survey. *ACM SIGKDD Explorations Newsletter*, 7(2):3–12, 2005.

[GV04] P.J.F. Groenen and M. Velden. Multidimensional scaling. Technical report, Erasmus School of Economics (ESE), 2004.

[HBL83] P.W. HollandKathryn Blackmond and S. Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

[HMS01] D.J. Hand, H. Mannila, and P. Smyth. *Principles of data mining*. The MIT press, 2001.

[JMF99] A.K. Jain, M.N. Murty, and P.J. Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

[Lar04] C. Larman. *Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development*. Prentice Hall PTR, 2004.

[LBJE03] J.J. Luczkovich, S.P. Borgatti, J.C. Johnson, and M.G. Everett. Defining and measuring trophic role similarity in food webs using regular equivalence. *Journal of Theoretical Biology*, 220(3):303–321, 2003.

[Miz93] M.S. Mizruchi. Cohesion, equivalence, and similarity of behavior: A theoretical and empirical assessment. *Social Networks*, 15(3):275–307, 1993.

[MWG+08] I. McCulloh, M. Webb, J. Graham, K. Carley, and D.B. Horn. Change detection in social networks. Technical report, DTIC Document, 2008.

[Sch90] R.J. Schalkoff. *Artificial Intelligence Engine*. McGraw-Hill, Inc., 1990.

[Sch95] H. Schildt. *C++: manual de referencia*. McGraw-Hill Interamericana de España, 1995.

[Sof05] P. Soffer. Refinement equivalence in model-based reuse: Overcoming differences in abstraction level. *Journal of Database Management (JDM)*, 16(3):21–39, 2005.

[T+10] R. Team et al. R: A language and environment for statistical computing. *R Foundation for Statistical Computing Vienna Austria*, (01/19), 2010.

[Tee11] Paul Teetor. *R Cookbook*. O'Reilly, first edition, 2011.

[TL10] L. Tang and H. Liu. Graph mining applications to social network analysis. *Managing and Mining Graph Data*, pages 487–513, 2010.

[WF94] S. Wassermann and K. Faust. Social network analysis: Methods and applications. *New York*, 1994.

[WJ04] AP Wolfe and D. Jensen. Playing multiple roles: Discovering overlapping roles in social networks. In *ICML-04 Workshop on Statistical Relational Learning and its Connections to Other Fields*, 2004.

[WM83] C. Winship and M. Mandel. Roles and positions: A critique and extension of the blockmodeling approach. *Sociological methodology*, 1984:314–344, 1983.

[WMAB09] F. Walter Mora and A. Alex Borbón. Edición de textos científicos con latex, latex2html y presentaciones con beamer. *Escuela de Matemática Instituto Tecnológico de Costa Rica*, 2009.

[Žib08] A. Žiberna. Direct and indirect approaches to blockmodeling of valued networks in terms of regular equivalence. *Journal of Mathematical Sociology*, 32(1):57–84, 2008.

[ZSZZ05] D. Zhou, Y. Song, H. Zha, and Y. Zhang. Towards discovering organizational structure from email corpus. In *Machine Learning and Applications, 2005. Proceedings. Fourth International Conference on*, pages 6–pp. IEEE, 2005.

Anexo A

Lista de Reserva del Producto

La LRP define las características del sistema y todo el trabajo que se realizará en el proyecto para dar solución al problema planteado. Está expuesta a cambios y modificaciones pues cuando un proyecto inicia es muy difícil tener claro todos los requerimientos del producto, teniendo en cuenta que solo puede ser cambiada entre iteraciones. Tiene como objetivo que el proyecto final sea más concreto y útil [30].

A continuación se presenta la LRP de la solución propuesta:

Asignado a	Item*	Descripción	Estimación	Estimado por
Requisitos funcionales				
Prioridad: Muy Alta				
Jorge Enrique Miralles (Programador)	1	Detectar roles.	3	Programador
Prioridad: Alta				
Jorge Enrique Miralles(Programador)	2	Registrar actores según rol.	2	Programador
Yaritza Rodriguez Garcia(Programador)	3	Establecer comunicación con el Controlador de MOCICE.	2	Programador
Prioridad: Media				
Yaritza Rodriguez Garcia(Programador)	4	Registrar trazas de ejecución del Subsistema de Detección de Roles.	2	Programador
Jorge Enrique Miralles(Programador)	5	Leer fichero de configuración.	2	Programador
Requisitos no funcionales				
Categoría: Restricciones en el diseño y la implementación				

Yaritza Rodriguez Garcia(Programador) Jorge Enrique Miralles(Programador)	6	Usar TCP/IP para la comunicación.	1	Programador
Yaritza Rodriguez Garcia(Programador) Jorge Enrique Miralles(Programador)	7	Utilizar XML para el fichero de configuración.	1	Programador
Yaritza Rodriguez Garcia(Programador) Jorge Enrique Miralles(Programador)	8	Utilizar como lenguaje de programación C++.	1	Programador
Yaritza Rodriguez Garcia(Programador) Jorge Enrique Miralles(Programador)	9	Utilizar NetBeans como IDE.	1	Programador
Yaritza Rodriguez Garcia(Programador) Jorge Enrique Miralles(Programador)	10	Utilizar RapidSVN para el control de versiones.	1	Programador
Yaritza Rodriguez Garcia(Programador) Jorge Enrique Miralles(Programador)	11	Utilizar Rats y Valgrind para la revisión automática del código.	1	Programador
Categoría: Software				
Yaritza Rodriguez Garcia(Programador) Jorge Enrique Miralles(Programador)	12	Usar Ubuntu 10.04 como sistema operativo.	1	Programador

Yaritza Rodriguez Garcia(Programador) Jorge Enrique Miralles(Programador)	13	El subsistema debe utilizar PostgresSQL como gestor de BD.	1	Programador
Yaritza Rodriguez Garcia(Programador) Jorge Enrique Miralles(Programador)	14	La documentación del código se realizará con Doxygen y se exportará en formato html y pdf.	1	Programador
Categoría: Escalabilidad				
Yaritza Rodriguez Garcia(Programador) Jorge Enrique Miralles(Programador)	15	El subsistema debe permitir adicionar diferentes algoritmos para la detección de roles.	1	Programador

Cuadro A.1: Lista de Reserva del Producto.

Anexo B

Archivo de configuración

```
<!--Configuration file of the Role Detection Subsistem-->
<config>
    <!--Controller connection setting-->
    <control>
        <!--Name of the controller (Do not modify!!)-->
        <name>mocice-control</name>
        <!--Name of the module that is stablishing the comunication whit the
            controller (Do not modify!!)-->
        <module>roles</module>
        <!--Ip address of the server where the controller is installed-->
        <ip>10.8.149.141</ip>
        <!--Port listening in the controller server-->
        <port>28503</port>
    </control>
    <!--Database connection settings -->
    <database>
        <!--Host where the database is installed-->
        <host>localhost</host>
        <!--Port to connect to the database-->
        <port>5432</port>
        <!--Name of the database-->
        <name>mail</name>
        <!--User to login in the database server-->
        <user>mocice</user>
        <!--Password to login in the database server-->
```

```
        <passwd>mocice</passwd>
        <!--Database connector libraries settings -->
        <connector>
                <!--Path where the databases connectors libraries are-->
                <path>DatabaseConnector/</path>
                <!--Name of the library to connect to the database-->
                <type>libPostgreeConnector</type>
        </connector>
    </database>
    <!--Role detection algorithms settings-->
    <algorithm>
        <!--Path where the role detection libraries are-->
        <path>Algorithms/</path>
        <!--Name of the library of role detection to be loaded-->
        <name>libtriads</name>
    </algorithm>
</config>
```