Inferring Use-cases from GUI Analysis

E. Miranda, C. Abdelahad, M. Berón and D. Riesco

***Abstract*— One of the most arduous and tedious tasks in the life cycle of an application is Software Maintenance and Evolution. In this context, the most time-consuming activities are those that the programmer must execute to get a complete understanding of the system. Based on this need, Program Comprehension (PC), a Software Engineering discipline, arises to tackle the problem. This article presents SSPIA, a strategy that assists software engineers to understand a system by inferring its use case model. SSPIA uses concepts, techniques and tools of PC to carry out its purposes. In order to extract use cases, some specific system static information is collected. This information serves as input to a process that implements a clustering technique based on system's Graphical User Interface (GUI). Almost all clustering techniques take as main criteria graph's structural properties. However, in the context of PC, some Problem Domain information must be considered. The strategy proposed in this article takes into account an essential component strongly related to system's Problem Domain: Graphic User Interfaces. As a main aim, the proposed strategy seeks to assist the arduous cognitive process that involves understanding a system.**

***Keywords*— Reverse Engineering, Program Comprehension, Clustering, Information Extraction.**

I. Introducción

SIN lugar a dudas, una de las tareas más complejas y que más tiempo consume en el ciclo de vida de una aplicación es la de Mantenimiento y Evolución de Software (MES) [1][2]. Dentro del entorno de MES, las tareas que más tiempo demandan son aquellas que debe ejecutar el ingeniero para lograr un completo entendimiento del sistema [3]. A partir de la necesidad de asistir al arduo proceso de comprensión mencionado anteriormente, surge una disciplina de la Ingeniería de Software denominada Comprensión de Programas (CP). La CP se presenta como un área de investigación interesante para impulsar el trabajo de MES a través de técnicas y herramientas que asistan al ingeniero de software en la difícil tarea de analizar y comprender sistemas [3]-[5].

En este artículo se presenta SSPIA (*Strategy for Static Program Information Analysis*), una estrategia que propone asistir al ingeniero durante el proceso de comprensión de programas mediante la extracción y posterior análisis de información estática. El objetivo de SSPIA es detectar los principales casos de uso del sistema y las dependencias entre los mismos. Para esto se define un proceso de agrupamiento (*clustering*) que permite componer artefactos de software hasta llegar a entidades (*clusters*) que serán mapeadas con casos de uso del sistema bajo estudio. El proceso toma como base el análisis de artefactos de software fuertemente relacionados con el Dominio del Problema como lo son los componentes de la interfaz gráfica de usuario (o *GUI,* por sus siglas en inglés) del sistema.

Una de las principales ventajas de utilizar casos de uso, en el contexto de Ingeniería Reversa, es que los mismos son fáciles de entender para todos los involucrados en los procesos de desarrollo o MES. Por lo tanto, se presenta como un medio de comunicación conveniente en el entorno [6].

La estrategia propuesta es implementada mediante transformaciones sucesivas entre modelos en el contexto de Arquitectura Dirigida por Modelos (o por sus siglas en inglés, *MDA*) utilizando el lenguaje QVT [7]. Esto facilita la aplicación de la misma en distintas plataformas, usando como base los modelos definidos en este trabajo.

El artículo está organizado de la siguiente manera: en la sección II se exponen los trabajos relacionados. En la sección III se presentan conceptos y procesos involucrados en SSPIA. En la sección IV se presenta la concepción de la estrategia como un proceso de Ingeniería Reversa usando MDA y transformaciones entre modelos. En la sección V se expone el prototipo que implementa SSPIA. En la sección VI se muestra un caso de estudio donde se aplica SSPIA a Febrl, un sistema desarrollado en lenguaje Python. Finalmente, en la sección VII se presentan las conclusiones y trabajos futuros.

II. Trabajos Relacionados

Desde hace varios años se han propuesto numerosos métodos para extraer información y construir vistas de los sistemas de software. A continuación se mencionan las propuestas más significativas estrechamente relacionadas con las temáticas que aborda este artículo.

A) Estrategias de Clustering

Muchos autores proponen técnicas que extraen información estática o dinámica de los sistemas y realizan agrupamientos con el objetivo de generar representaciones de más alto nivel. Algunos trabajos usan criterios basados en propiedades estructurales de las distintas representaciones construidas a partir del código fuente para realizar los agrupamientos y de esta manera generar abstracciones del sistema. Shtern y Tzerpos [8] presentan un amplio estado del arte de este tipo de técnicas. No obstante, las mismas sólo toman en consideración información estructural de las representaciones utilizadas para modelar los sistemas, por ejemplo, grafos de llamadas a funciones, estructura de archivos, uso compartido de datos, entre otros. La mayoría de estas técnicas han sido concebidas en otras áreas donde no existe mayor información que la provista por la estructura a ser analizada (por ejemplo,

E. Miranda, Universidad Nacional de San Luis, San Luis, Argentina, eamiranda@unsl.edu.ar

C. Abdelahad, Universidad Nacional de San Luis, San Luis, Argentina, cabdelah@unsl.edu.ar

M. Berón, Universidad Nacional de San Luis, San Luis, Argentina, mberon@unsl.edu.ar

D. Riesco, Universidad Nacional de San Luis, San Luis, Argentina, driesco@unsl.edu.ar

biología, matemática, economía, redes sociales, etc). Este tipo de enfoques, donde la información sustancial se encuentra en las relaciones entre componentes de una representación del sistema (por ejemplo, relaciones entre nodos en un grafo), dejan de lado información relevante en un contexto de Ingeniería Reversa.

Por otra parte, también se han desarrollado técnicas de clustering que, a diferencia de las anteriores, consideran otro tipo de información del sistema además de la estructural como por ejemplo: equipos de desarrollo [9][10], cambios históricos durante el desarrollo y/o mantenimiento del sistema [11], identificadores de artefactos de software en el código fuente e información informal [12][13], detección de patrones [14], modelos del Dominio del Problema [15], entre otros. Si bien los trabajos mencionados previamente proponen alternativas distintas a las mencionadas al principio del apartado, la efectividad de sus resultados depende fuertemente de la existencia y claridad de cierto tipo de información. Tomar como base exclusivamente este tipo de información, puede resultar en la obtención de descomposiciones erróneas del sistema. Además es importante destacar que ésta información suele no estar presente en el contexto del proyecto o no estar del todo clara. Por otra parte, en el contexto de análisis de información informal, se debe sumar las dificultades que implica el tratamiento de lenguaje natural (consideración de sinónimos, verbos conjugados, errores ortográficos, etc.).

Un enfoque interesante es propuesto por el algoritmo ACDC [14]. Los autores afirman que, dentro del contexto de Comprensión de Programas, es más importante lograr que un agrupamiento ayude a entender el sistema antes que maximizar una métrica. Proponen ciertos factores a los cuales se les debe dar importancia, como mapeo de patrones y el nombramiento efectivo de los clusters [8]. No obstante, se ha evaluado el desempeño del algoritmo y los resultados no reflejan la efectividad planteada por la detección de patrones [16].

B) Ingeniería Reversa para obtener Casos de Usos

También existen trabajos que proponen extraer información del sistema con el objetivo de obtener y/o analizar casos de uso de un sistema determinado.

Bojic and Velasevic [6] integran técnicas de análisis estático y dinámico. Definen un conjunto de casos de uso con sus correspondientes casos de prueba. Las trazas de ejecución obtenidas a partir de los casos de prueba son mapeados con entidades en el código fuente. Aplicando análisis de conceptos sobre el código fuente, las entidades detectadas en las trazas que implementan funcionalidades similares son agrupadas y vinculados a los casos de uso correspondientes. Salah et al. [17] presentan un enfoque semejante, sin embargo no utilizan modelos UML sino que crean sus propias vistas con el objetivo de proveer diversos niveles de abstracción.

El-Ramly y Sorenson [18] plantean obtener casos de uso para representar el comportamiento del sistema usando trazas generadas a partir de la interacción del usuario con el sistema. Utilizan *data mining* y *pattern matching* para explorar estas trazas con el objetivo de analizar las tareas más importantes para el usuario. Cuando se encuentran patrones relevantes, son enriquecidos por el ingeniero de software con información semántica y tomando los mismos como base, se construyen los casos de uso.

Zhang et al. [19] presentan un enfoque para asociar código fuente a los casos de uso a través de análisis estático. El método es semi-automático y genera un grafo de llamadas que considera el flujo de control de los programas. Luego se usan heurísticas para filtrar funciones de bajo nivel. Por último, arman el modelo de casos de uso. El principal criterio de detección de casos de uso son las bifurcaciones en el código y para la detección de dependencias entre estos se analizan las llamadas a funciones. Si et al. [20] presentan una aproximación parecida, sólo que construyen un grafo de llamadas a funciones con bifurcaciones. Luego realizan un pesaje sobre el mismo de acuerdo a una métrica de complejidad. Posteriormente extraen trazas de manera estática (simulan ejecución) para capturar todas los posibles caminos de ejecución. Durante este proceso, se asiste al ingeniero de software para que: i) determine los distintos escenarios; ii) determine que escenarios corresponden a un mismo caso de uso; iii) identifique los actores, la dependencia entre los casos de uso y los tipos de dependencia (relaciones *include*, *extend* o generalización). Li et al. [21] proponen un enfoque similar al de Zhang et al. y Si et al. sólo que usan información dinámica.

Dugerdil et al. [22] proponen extraer los casos de uso de un sistema usando información dinámica. Los autores definen un nuevo formato para representar trazas de ejecución, el mismo permite especificar un *árbol de decisión dinámico* (representa las alternativas de ejecución dentro de la traza) para cada caso de uso. Además se propone un algoritmo de reducción de dicho árbol en conjunto con la identificación de flujos alterados por el usuario. De esta manera el algoritmo posibilita la identificación de flujos principales y alternativos en la descripción de cada caso de uso.

Pereira et al. [23] proponen extraer el modelo de casos de uso extrayendo información estática y dinámica del sistema. En primer lugar, transforman el código fuente en una representación que sólo refleja las llamadas a métodos. Posteriormente, ejecutan un algoritmo que aproxima el conjunto de casos de uso de manera estática (consideran a cada método un caso de uso). Finalmente, ejecutan un conjunto de reglas que permiten detectar: i) las dependencias entre los casos de uso; ii) el tipo de relación (generalización, *include o extend*); iii) casos de uso de más alto nivel y iv) flujos principales y alternativos dentro de cada caso de uso (usa información dinámica). Las reglas se aplican usando transformaciones de modelos, para esto los autores definen un framework de Ingeniería Reversa basado en MDA.

Todas las propuestas difieren en distintos aspectos de la desarrollada en este artículo y presentan ciertas desventajas en común, por ejemplo, muchos de los trabajos no construyen las relaciones entre casos de uso, sino que vinculan artefactos del

código a determinados casos de uso ya identificados [6][20][22]. La mayoría de las propuestas requieren de un usuario con un amplio conocimiento del sistema para: i) detectar casos de uso esenciales [6][20][22]; ii) identificar propiedades en los modelos de casos de uso como tipos de relaciones (*include*, *extend* o generalizaciones), incluso actores o agrupamiento de casos de usos [20][23]; o iii) insertar acciones semánticas que ayuden a detectar componentes en el modelo [18]. Otras propuestas detectan casos de uso usando como principal criterio el flujo de control en el código [19][20]. Sin embargo, no es directa la relación entre los diferentes flujos de control y los casos de uso.

Una de las principales desventajas que tienen en común todas las propuestas es que no consideran ciertos artefactos importantes presentes en la mayoría de los sistemas actuales, como lo son los componentes de la interfaz gráfica de usuario (GUI). Estos componentes poseen información fuertemente relacionada con el Dominio del Problema; los mismos han sido diseñados para interactuar con el usuario con el objetivo de resolver el problema para el cual el sistema ha sido desarrollado. Los componentes de la interfaz brindan información respecto a términos del Dominio del Problema, qué tipo de función cumple cada uno, la vinculación con las partes del código que implementan el mismo, clasificación de funcionalidades, entre otros aspectos. Además, salvo la propuesta planteada en [23], es difícil encontrar trabajos donde se utilicen enfoques ingenieriles como MDA. Esto, claramente, hace a las propuestas que utilizan este enfoque, más independientes respecto a determinados factores de las plataformas utilizadas, como por ejemplo, lenguajes de programación, librerías gráficas, ciertos paradigmas de programación, entre otros.

Para contrarrestar las desventajas mencionadas en párrafos previos se presenta la estrategia SSPIA. La misma define una representación interna que incluye información sobre las dependencias entre los métodos/funciones en el código fuente y sobre los elementos de la GUI del sistema. A partir de esta representación, se utiliza toda la información extraída del sistema (dependencia entre métodos/funciones, manejadores de elementos GUI, métodos/funciones relacionadas con elementos GUI, métricas específicas, etc.) para realizar clustering de métodos/funciones. Finalmente, el agrupamiento obtenido se mapea a un modelo de casos de uso. Para el nombramiento de los casos de uso, SSPIA hace uso de información provista por elementos que componen la GUI.

III. Estrategia SSPIA

En esta sección, todos los conceptos y procesos involucrados en SSPIA son desarrollados.

A) Representación de la Información Extraída

La primera etapa de la estrategia consta de la extracción de la información sobre el código fuente del sistema. Una de las estructuras más utilizadas en el contexto de Ingeniería Reversa es el Grafo Estático de Llamada a Funciones (GELF) [24]. En el caso de lenguajes con soporte multiparadigma (como Python), la versión del grafo es Grafo Estático de Llamada a Método/Función (GELMF). Por otra parte, también se extrae información respecto de la jerarquía de la GUI y sus componentes en una estructura denominada Árbol GUI. En los próximos párrafos se explican brevemente estas 2 estructuras.

Grafo Estático de Llamada a Método/Función. En pocas palabras, en un GELMF para un sistema determinado, cada nodo representa un método o función del mismo; mientras que un arco desde el nodo *a* a otro *b* se interpreta como una dependencia estática desde el método/función *a* hacia *b*. Esta estructura es explicada más detalladamente en [24]. Para esta instancia de la investigación, el GELMF ha sido *decorado* con información propia del sistema. Para ser más específico, por cada nodo (método/función) del grafo:

i) se calcula y almacena un conjunto de métricas;
ii) se registran los componentes de la interfaz gráfica referenciados por el método/función que representa el nodo
iii) si el nodo es un manejador de un componente interactivo de la interfaz, se guarda información completa del mismo.

La información antes descripta, es utilizada con diferentes propósitos en la estrategia, uno de estos es determinar la *importancia relativa* de cada nodo (método/función) para con el sistema.

Árbol GUI. Esta estructura mantiene la representación estática de la GUI del sistema reflejando la jerarquía de componentes de la misma. Es una estructura arbolada ya que posee una única raíz (un nodo que representa la aplicación) y no posee ciclos [25]. Cada nodo en el árbol representa un componente de la GUI; estos pueden ser: *contenedor* (ventanas, paneles, cuadros, etc.) o *elemento final* (botones, rótulos, entradas de texto, imágenes, etc.). Una estructura similar es utilizada por Memon et al. en la herramienta *GUI Ripping* [25].

B) Técnica de Clustering

En el contexto de Comprensión de Programas, los algoritmos de clustering agrupan artefactos de software de un sistema (por ejemplo, métodos, clases, archivos, etc.) en subsistemas, con el principal objetivo de asistir al proceso de comprensión de un sistema de software. La estrategia SSPIA propone un algoritmo de clustering que agrupa progresivamente nodos del GELMF, tomando como principal criterio cierta información provista por los componentes GUI, con el objetivo de obtener los casos de uso del sistema bajo estudio. A continuación se exponen las etapas que comprenden la estrategia de clustering.

1. Extracción de Información. Se extrae información estática correspondiente al sistema (descripta en el apartado III.*A*). Por cada método/función en el código fuente:

i) se extraen las dependencias entre los mismos (GELMF);
ii) se registra cierta información referente a los componentes de la interfaz de usuario (cantidad de componentes GUI utilizados, tipo de componente GUI implementado, etc.) y
iii) se calculan métricas por cada uno de estos (complejidad ciclomática, cantidad de líneas de código, *intermediación* (medida de centralidad), cantidad de sentencias de conexión

con bases de datos, entre otras).

Toda esta información será utilizada posteriormente en las diferentes etapas de la estrategia. Uno de los principales objetivos de las métricas extraídas, es la aproximación de la *importancia relativa* de cada método para con el sistema. Para más detalles acerca de la información y las métricas extraídas se puede consultar [26].

2. Reducción de Información. Debido al tamaño que puede alcanzar el GELMF, es necesario contar con métodos de reducción de información [27]. Como es posible inferir, dichos métodos deben reducir considerablemente la pérdida de artefactos de software sensibles a la lógica del programa. Para esto, es necesario aplicar técnicas que permitan discernir entre artefactos que son relevantes respecto de la lógica subyacente del sistema y los que no lo son. Para la reducción del GELMF se ha tomado un enfoque similar al planteado por distintos autores en este contexto [20] [28] [29], donde se prioriza la detección de "utilidades". Para conocer la técnica de reducción del GELMF en detalle, el lector interesado puede consultar un trabajo previo donde se explica la misma [24].

3. Identificación de nodos "inicializadores". En esta etapa se identifican aquellos nodos que inicializan el sistema, particularmente, los que crean o establecen conexiones con componentes GUI del sistema. Estos son agregados a un nuevo cluster denominado "INIT". Por lo general, estos nodos suelen tener una fuerte interacción con los componentes GUI y en determinados tipos de aplicaciones, tienen una intensa carga computacional y lógica dentro del programa; por estos motivos es una buena práctica detectarlos y visualizarlos para que el ingeniero pueda analizar los mismos (etapa 6: Nombramiento de Clusters).

4. Detección de Manejadores. Cada componente GUI interactivo, como botones, listas desplegables (*comboBox*), elementos del menú, etc., poseen un método manejador (también puede ser una función, dependiendo del lenguaje). Es decir, un método o función que determina el comportamiento de un componente GUI determinado. Dichos componentes están vinculados con los casos de uso ya que están fuertemente relacionados con el usuario (actor) y la interacción con el sistema [6][30]. La estrategia define un nuevo cluster por cada uno de estos métodos/funciones (referenciado como *cluster manejador*). Además de estar compuesto por el nodo manejador, también se agrupan en el mismo aquellos nodos que son dependencia exclusiva del manejador en cuestión. Cada uno de estos clusters, en principio, es nombrado con algún término extraído del componente GUI interactivo. En muchas oportunidades, éste puede no proveer ninguna denominación concisa para el cluster; para estos casos el nombre es determinado por el ingeniero (etapa 6).

5. Agrupamiento Estructural. En esta etapa se analizan aquellos nodos que no han sido agrupados en etapas previas. Para esto, se toman en consideración dos criterios principales que utilizan la información extraída del sistema (etapa 1):

- ***Integrar nodos (o clusters) a ciertos clusters ya definidos:*** se ha definido un conjunto de reglas que determinan cuándo un nodo debe agruparse a un cluster ya definido. A continuación se explican brevemente tres de las más relevantes: i) si un cluster depende exclusivamente de un nodo (o viceversa), este último es agrupado a dicho cluster; ii) si un cluster *a* depende exclusivamente, de otro cluster *b,* y este último no es un cluster formado a partir de un nodo manejador, posee importancia relativa inferior a la media y no está fuertemente relacionado con componentes de la interfaz gráfica, entonces el cluster *b* es absorbido por *a*; iii) si dos clusters dependen mutuamente uno de otro, entonces el cluster con menor importancia relativa es agrupado al que tenga mayor importancia de ambos. Para la determinación de la importancia relativa de cada cluster, se utilizan las métricas calculadas en la etapa 1.

- ***Definir nuevos clusters:*** posibilita la creación de nuevos clusters a partir de nodos que no han podido ser agrupados hasta el momento. En este caso también se ha definido un conjunto de reglas, entre las que se pueden destacar: i) si un nodo está relacionado con componentes de la interfaz gráfica y posee importancia relativa superior a la media, entonces este nodo compone un nuevo cluster; ii) si dos o más clusters dependen de un nodo, entonces dicho nodo pasa a componer un nuevo cluster; entre otras reglas de menor relevancia.

Todas las reglas de esta etapa se ejecutan hasta que ya no haya nuevas modificaciones en la estructura; cuando esto suceda, por cada uno de los nodos que no han sido agrupados se define un cluster que lo contiene. De esta manera, al finalizar esta etapa, todo nodo está agrupado en un cluster. El lector interesado en profundizar sobre los procesos involucrados en esta etapa puede consultar [26].

6. Nombramiento de Clusters. En esta etapa se le provee al ingeniero información respecto a los elementos que componen cada cluster y sus dependencias con el objeto de que éste determine el nombre más conveniente para los mismos. Esto es muy importante ya que una estrategia de nombramiento débil puede hacer fracasar la compresión del sistema [14]. La información provista está relacionada con los métodos/funciones que componen el agrupamiento, los elementos GUI vinculados al cluster y toda la información extraída en la primera etapa. Entre la información que se provee se encuentra:

i) rótulo, identificador de la variable en el Árbol GUI, tipo de componente, *tooltip* (o descripción emergente), etc. correspondientes al botón asociado al método/función manejador que originó el cluster;

ii) signatura de los componentes de cada cluster, señalando los métodos/funciones que no poseen arcos de entrada en todo el sistema y métodos/funciones que no poseen arcos de entrada dentro de propio cluster ("raíz" de cluster) y

iii) métricas asociadas a cada cluster como complejidad ciclomática (sumatoria de complejidades de los componentes del cluster), cantidad de elementos GUI referenciados, si es cluster manejador y si es inicializador.

Mediante el uso de esta información, el ingeniero puede

definir el nombre correspondiente a cada cluster. Esta etapa debe ser realizada por el ingeniero ya que es el indicado para determinar el mejor nombre de acuerdo a toda la información provista. En el peor de los casos, el cluster recibe como nombre la signatura del método/función que sólo posea arcos de salida dentro del mismo (es decir, el método/función raíz del cluster). Esta etapa permite, en cierta forma, contrarrestar el nombramiento deficiente que suelen presentar los métodos/funciones de un sistema [13]. En la sección V se explica el prototipo de SSPIA y la interfaz gráfica que permite al ingeniero definir los nombres de los clusters.

7. Generación de casos de uso. En la última etapa, se genera una aproximación al modelo de casos de uso del sistema. Se mapea cada cluster a un caso de uso y los arcos entre clusters en dependencias entre los casos de uso. En la sección IV se detalla el proceso de transformación de esta etapa. En primera instancia, las dependencias se establecen del tipo *include*. Se deja como trabajo futuro la determinación de las dependencias *extend* y las generalizaciones ya que las mismas son poco frecuentes y complejas de determinar automáticamente [20].

En todas las técnicas de clustering aplicadas en el contexto de Ingeniería Reversa, un factor esencial es el nombramiento efectivo de los clusters [14]. SSPIA lleva a cabo esta tarea tomando como principal criterio los componentes que se presentan en la GUI del sistema. Así, muchos de los clusters son nombrados con términos vinculados con el Dominio del Problema del sistema. Además, en la etapa 6 se permite al ingeniero definir/cambiar los nombres de los clusters tomando como base información extraída desde el sistema. De esta manera, la mayoría de los casos de uso resultantes tendrán nombres relacionados al Dominio del Problema de la aplicación, siendo de suma importancia para la comprensión del sistema bajo estudio.

IV. ESPECIFICANDO SSPIA CON MDA

Con el objetivo de independizar la estrategia de plataformas, lenguajes de programación y librerías gráficas específicas, SSPIA hace uso de Arquitectura Dirigida por Modelos (MDA).

La idea principal de MDA, es separar la funcionalidad del sistema de su implementación sobre plataformas específicas, considerando la evolución desde modelos abstractos hasta su implementación, acrecentando el grado de automatización y logrando interoperabilidad con múltiples plataformas, lenguajes formales y lenguajes de programación. MDA distingue tres tipos diferentes de modelos: CIM (*Computation Independen Model*), PIM (*Platform Indepedent Model*), PSM (*Platform Specific Model*). En MDA, los modelos son artefactos generados a través de transformaciones durante el proceso de desarrollo. Para permitir esto, MDA requiere que estos modelos estén expresados en un lenguaje de metamodelo. La esencia de MDA es que estos metamodelos están basados en MOF (*Meta Object Facility*) permitiendo utilizar diferentes artefactos de distintos proveedores para ser combinados en un mismo proyecto. Desde su concepción, numerosos enfoques han utilizado MDA con diferentes propósitos [31]-[34].

Usando MDA, cualquier especificación puede ser expresada con modelos, en consecuencia un proceso de Ingeniería Reversa puede comprender un proceso de refinamiento y transformación entre modelos de manera tal que el nivel de abstracción vaya aumentando hasta llegar a una representación independiente de la plataforma (PIM). La definición de transformaciones de modelos requiere la aplicación de lenguajes específicos. Estos lenguajes deberían tener una base formal, y al menos un metamodelo que describa su sintaxis abstracta [35]. Un metamodelo describe el conjunto de modelos admisibles y define formalmente los elementos de un lenguaje de modelado junto con sus relaciones y restricciones.

Para la estrategia presentada en este artículo se utiliza QVT-R (QVT Relations), componente de QVT [7], el cual permite formalizar transformaciones dirigidas por modelos. Las transformaciones QVT describen relaciones entre el metamodelo origen y el metamodelo destino, ambos especificados en MOF.

A) Arquitectura de la transformación

Para exponer el proceso de Ingeniería Reversa a nivel metamodelo, es necesario especificar los metamodelos de origen y destino para la transformación (Fig. 1 y Fig. 2). El metamodelo origen se compone de los elementos referidos en la descripción de las etapas de la estrategia (los componentes del Árbol GUI y del GELMF). La Fig. 1 muestra un diagrama de clase con los elementos del metamodelo de SSPIA. Los elementos básicos del metamodelo origen están contenidos en la clase *SspiaModel*. La clase *Cluster*, como su nombre lo indica, representa el agrupamiento de elementos funcionales. Estos elementos funcionales son modelados con la clase

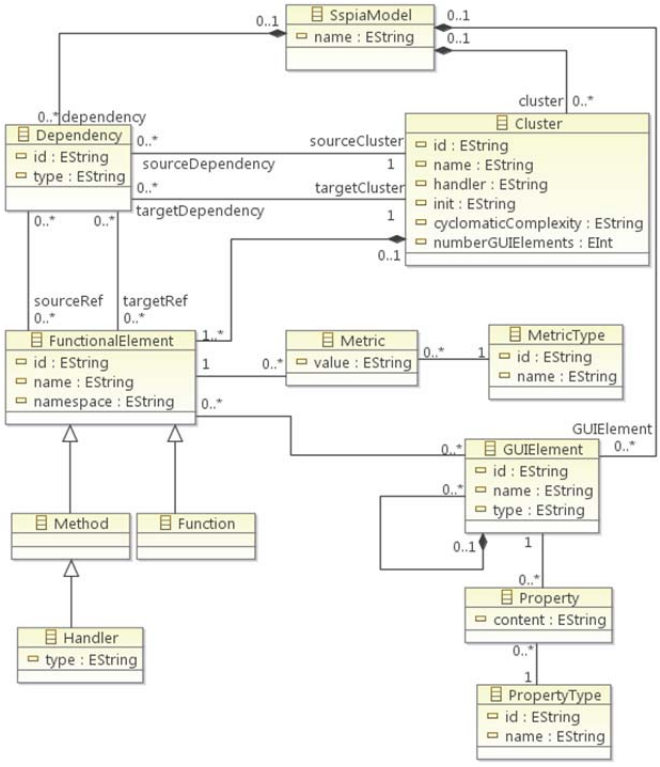

Figura 1.Metamodelo de SSPIA (origen de la transformación).

FunctionalElement y los mismos pueden ser métodos o funciones del GELMF. Los arcos entre elementos funcionales y entre clusters son representados a través de la clase *Dependency*. Se puede observar también que los elementos funcionales poseen relaciones con la clase *Metric*, la cual es utilizada durante todo el proceso de transformación a través de las etapas de la estrategia, y con la clase *GUIElement*. Estos elementos GUI pueden estar agrupados en un Árbol GUI, el cual es representado a través de una relación de composición. Además, estos elementos poseen distintas propiedades (*Property*) de acuerdo a cada clase de componente en la interfaz.

La instancia inicial del metamodelo origen es construida luego que se extrae información del sistema (etapa 1). Este modelo es recibido como parámetro por la transformación para luego generar el modelo destino.

El metamodelo destino corresponde al metamodelo de UML [36]. La Fig. 2 muestra una vista parcial del metamodelo para casos de uso. En ella se observa como el elemento *UseCase* puede estar vinculado con otro caso de uso por medio de la relación del tipo *include* o relacionarse con actores por medio de *asociaciones.*

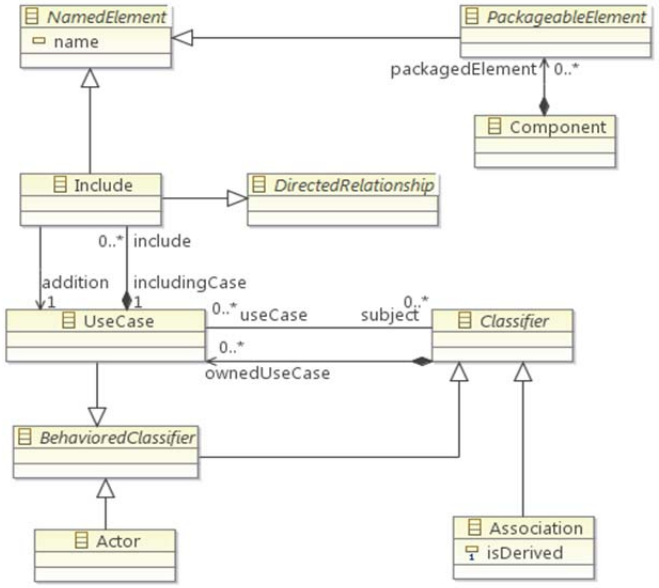

Figura 2. Vista parcial del metamodelo de UML para Casos de Uso (destino de la transformación).

Es importante destacar que la transformación utilizada para llevar a cabo las distintas etapas de la estrategia puede ser desagregada en transformaciones parciales donde se utiliza el metamodelo de SSPIA como origen y destino. Esto se da en las etapas 2, 3, 4, 5 y 6. Es decir, en un principio, el modelo no posee clusters, sólo elementos funcionales (métodos/funciones) relacionados entre sí. Las transformaciones parciales van alterando el modelo original, de manera tal de que al finalizar la etapa 5, el modelo sólo contiene clusters compuestos por elementos funcionales. Es decir, todo elemento funcional pertenece a un cluster determinado. En la etapa 6 y 7 se establece el nombre a cada cluster y posteriormente, se mapea el modelo SSPIA obtenido hasta el momento a un modelo de Casos de Uso.

B) Especificación de la transformación en QVT-R

Por motivos de extensión del artículo, a continuación se presenta el código QVT de la transformación correspondiente a la última etapa de la estrategia. Como se explicó en la sección III, en esta etapa se transforma un modelo SSPIA en un modelo de Casos de Uso. Es importante destacar que el modelo recibido como origen para esta transformación es el producto de las transformaciones en las etapas previas. En el mismo se encuentran los clusters completamente definidos (etapas 1-5) y nombrados (etapa 6).

A continuación, se muestran las relaciones que definen el mapeo entre algunas clases del metamodelo SSPIA y clases del metamodelo UML (Fig. 3, 4 y 5). En la Fig. 3 se observa la relación *sspiaModel2UmlModel*, la cual constituye el punto de entrada para comenzar la transformación. La finalidad de esta relación es crear la etiqueta *model*, etiqueta principal en cualquier modelo UML, e invocar a la relación *cluster2UseCase*. Esta relación define el mapeo entre los clusters del modelo SSPIA y los casos de usos del modelo UML. El código QVT presentado en esta relación muestra el procedimiento mediante el cual se obtiene el nombre del cluster (*clusterName*) para generar el caso de uso correspondiente.

```
transformation   phase7(source : sspia, target : uml) {

  top relation sspiaModel2UmlModel{
    modelName : String;

   checkonly domain source sspiaModel :sspia::SspiaModel{
     name = modelName
   };
   enforce domain target umlModel : uml::Model {
     name = modelName
   };
   where {
      cluster2UseCase(sspiaModel,umlModel);
   }
  }

  relation cluster2UseCase {
    clusterName : String;

   checkonly domain source sspiaModel :sspia::SspiaModel{
     cluster = cluster : sspia::Cluster {
       name= clusterName
      }
   };
   enforce domain target umlModel : uml::Model {
     packagedElement = useCase : uml::UseCase {
       name = clusterName
     }
   };
  where{
    edge2Include(sspiaModel,cluster,umlModel,useCase);
    createActor(sspiaModel,cluster,umlModel,useCase);
    createAssociation(sspiaModel,useCase,umlModel,
                      umlModel);
    }
  }
```

Figura 3. Fragmento de código QVT-R para transformación del modelo de SSPIA a Casos de Uso. Generación de Casos de Uso.

Una vez generado el caso de uso, la transformación se dirige hacia la cláusula *where* para invocar a las relaciones *edge2Include*, *createActor* y *createAssociation*. La Fig. 4 muestra la relación *edge2Include* que define el mapeo de las dependencias entre cluster del modelo SSPIA y las dependencias *include* entre casos de uso del modelo UML.

```
relation edge2Include {
  clusterName : String;

  checkonly domain source sspiaModel:sspia::SspiaModel{
    cluster = newCluster : sspia::Cluster {
      name = clusterName
    }
  };
  checkonly domain source oldCluster:sspia::Cluster {};

  checkonly domain target umlModel :uml::Model {
    packagedElement = useCaseIncluded :uml::UseCase {}
  };

  enforce domain target useCase :uml::UseCase {
    include = includeCU : uml::Include {
      name = 'UseCaseDependency',
      addition = useCaseIncluded
    }
  };
  when {
    if (oldCluster.targetDependency =
          newCluster.sourceDependency) and
       (useCaseIncluded.name=clusterName) then
      true
    else
      false
    endif;
     }
  }
```

Figura 4. Continuación de fragmento de código QVT-R para transformación del modelo de SSPIA a Casos de Uso. Generación de Relaciones *include.*

```
relation createActor {
  checkonly domain source sspiaModel:sspia::SspiaModel
{};
  checkonly domain source oldCluster:sspia::Cluster {};

  enforce domain target umlModel : uml::Model {
    packagedElement =  actor : uml::Actor {}
   };

  checkonly domain target useCase : uml::UseCase {};
  when {
    if ( oldCluster.handler = 'true' ) and
      (actorQuery(umlModel))then
      true
    else
      false
    endif;
     }
 }

relation createAssociation{

  checkonly domain source sspiaModel :sspia::SspiaModel
{};
  checkonly domain target useCase : uml::UseCase {};

  checkonly domain target tempUmlModel :  uml::Model {
    packagedElement = actor : uml::Actor {}
   };

  enforce domain target umlModel : uml::Model {
    packagedElement =  association : uml::Association {
      ownedEnd = association2useCase : uml::Property {
        type = useCase,
        association = association
       },
      ownedEnd = actor2association : uml::Property {
        type = actor,
        association = association
       }
      }
   };
  when {not actorQuery(umlModel);
  }
 }
...
 }
```

Figura 5. Continuación de fragmento de código QVT-R para transformación del modelo de SSPIA a Casos de Uso. Generación de Actor/es y relaciones de asociación.

Las relaciones *createActor* y *createAssociation* se muestran en la Fig. 5. Como se mencionó anteriormente, estas relaciones son invocadas por la relación *cluster2UseCase*. Estas relaciones son ejecutadas en el caso de que los clusters hayan sido originados a partir de un método/función manejador. Esta correspondencia no es directa, ya que dependiendo de si el cluster es manejador o no será necesario crear el actor (*createActor*) y las asociaciones (*createAssociation*) a todos los casos de uso que hayan sido generados por un cluster manejador.

V. Implementación de SSPIA

Para implementar la estrategia SSPIA, se ha desarrollado un prototipo que permite llevar a cabo todas las tareas descriptas anteriormente. El mismo está desarrollado en Java y utiliza la librería gráfica JUNG [37] para la visualización del GELMF. En la Fig. 6 se exhibe una aproximación de la arquitectura de dicho prototipo.

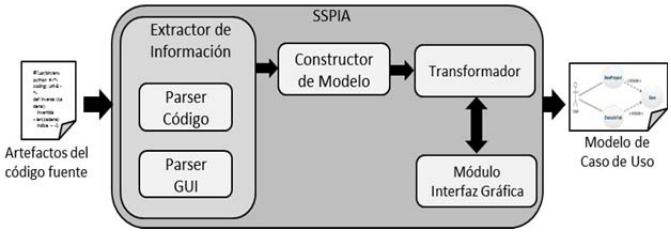

Figura 6.Arquitectura del prototipo de SSPIA.

La primera fase de la estrategia plantea extraer información desde el código fuente del sistema bajo estudio. El módulo *Extractor de Información* permite obtener información correspondiente al sistema. Para esto se utilizan técnicas de análisis sintáctico que permiten extraer el GELMF, las métricas asociadas a cada componente de dicho grafo y la relación de cada método/función con los elementos del Árbol GUI. Para extraer el Árbol GUI, se utiliza el archivo .glade [38] de cada sistema en conjunto con información extraída desde el código fuente con técnicas de análisis sintáctico. El *toolkit* Glade permite generar interfaces gráficas de manera visual usando GTK/GNOME. Dicha herramienta es independiente del lenguaje de programación y genera un archivo XML que mantiene los elementos de la interfaz y la estructura de la misma. Usando toda la información extraída desde el código fuente, el módulo *Constructor de Modelo* define el modelo del sistema tomando como base el metamodelo explicado en la sección precedente. Posteriormente, se llevan a cabo las transformaciones a dicho modelo hasta llegar a la etapa 6 en donde el ingeniero debe especificar el nombre de cada cluster.

La utilización tanto de MDA como de las trasformaciones, permite que los pasos de dicha estrategia se lleven a cabo considerando muy pocos aspectos dependientes de plataformas y tecnologías específicas. Un claro ejemplo de esto es la posibilidad de aplicar la estrategia a un lenguaje distinto y/o una librería o framework gráfico distinto. Para esto sólo sería necesario incorporar nuevos parsers para los elementos del componente *Extractor de Información* de la herramienta (*Parser de Código* y *Parser GUI*). Es decir, incorporando nuevos analizadores de código e interfaces

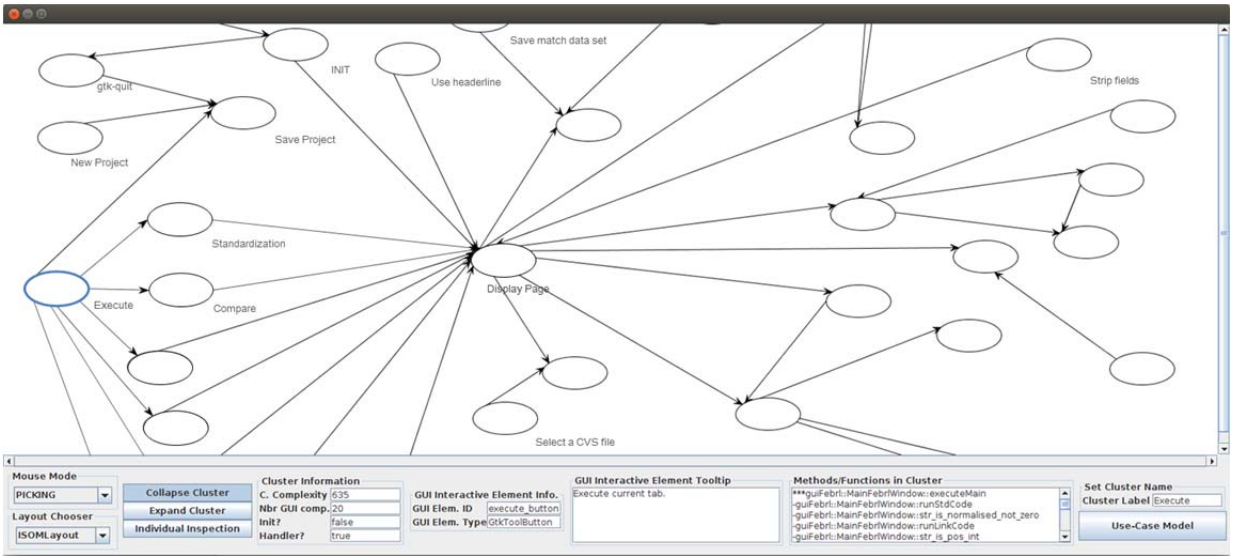

Figura 7. Captura prototipo SSPIA. Etapa de interacción con el ingeniero.

gráficas para los lenguajes/frameworks que se desean contemplar.

A) Módulo de Interfaz Gráfica

Antes de la última etapa de transformación, el prototipo presenta una interfaz (*Módulo Interfaz Gráfica*) para que el ingeniero determine los nombres a los clusters, como se explicó en la sección III. En la Fig. 7 se muestra una captura de dicha interfaz. A continuación se detallan los componentes gráficos que la misma presenta:

Panel central

Muestra el GELMF del sistema con los clusters definidos (nodos color blanco con forma elíptica) y las dependencias entre los mismos (arcos dirigidos). Para la visualización de esta representación se utilizó la librería gráfica JUNG [37].

Panel inferior

Exhibe un conjunto de campos de texto que despliegan toda la información disponible para el cluster seleccionado. A continuación se describen brevemente cada uno de estos:

Métricas de clusters. Conjunto de campos textuales que visualizan métricas referente al cluster (sumatoria de complejidades ciclomáticas, cantidad de componentes GUI referenciadas, si es cluster de nodos inicializadores y si es cluster de un nodo manejador).

Información del componente GUI interactivo. Siempre que el cluster fuera creado a partir de nodos manejadores de este tipo de elementos (como botones, listas desplegables, *checkbox* o cualquier tipo de componente interactivo que sea implementado por un manejador), se muestra información sobre dicho componente como: *tooltip,* nombre y tipo del componente en el Árbol GUI.

Componentes del cluster. Signatura de los métodos/funciones que están agrupados en el cluster seleccionado. Además por cada método/función se indica si es "fuente" en el cluster, es decir, si sólo posee arcos de salida dentro del mismo (con la marca "***" antes de la signatura). Esto es conveniente ya que los nombres dichos métodos/funciones pueden determinar significativamente el rótulo del cluster en cuestión.

Botones navegación de clusters. Se proveen 3 botones que permiten: i) *Collapse Clusters*: agrupar y desagrupar todos los nodos en sus respectivos clusters (los nodos de cada cluster son pintados del mismo color); ii) *Expand Cluster*: desagregar el cluster seleccionado y iii) *Individual Inspection*: analizar un cluster por separado, visualizando únicamente los nodos que lo componen. En conjunto, estos botones permiten navegar el GELMF para analizar la composición de los clusters y las relaciones internas y externas entre métodos/funciones que componen cada uno de estos. Claramente, dichas funcionalidades mejoran el proceso de inspección del sistema bajo estudio.

Botón casos de uso. Finalmente, una vez que el ingeniero ha completado la etapa de nombramiento de clusters, presiona el botón *Use-case Model* del panel inferior para generar el modelo de caso de uso del sistema analizado. Dicho modelo es generado en un archivo XMI (*XML Metadata Interchange*), un estándar de la OMG para la integración de herramientas de diversa índole. El mismo puede ser abierto en cualquier herramienta de modelado de UML que siga con el estándar. En el caso de SSPIA, este archivo permite la visualización del modelo de Casos de Uso obtenido.

En pocas palabras, el prototipo recibe el código fuente del sistema y el archivo .glade de su interfaz y retorna como resultado un archivo con formato XMI con una aproximación al modelo de casos de uso del sistema analizado.

El uso de MDA en la estrategia permite independizar

En la próxima sección se presenta un caso de estudio con el objetivo de mostrar la aplicabilidad del enfoque.

VI. Caso de Estudio

Con el fin de evaluar la utilidad de la propuesta, se aplicó SSPIA al sistema Febrl (*Freely Extensible Biomedical Record Linkage*) [39]. El mismo implementa técnicas avanzadas para limpieza de datos, estandarización, indexación, comparación

de archivos, clasificación de registro por pares, etc. Es utilizado por varias investigaciones en el contexto de la minería de datos. El proyecto completo posee ~45KLOCs y utiliza la librería PyGTK y el *toolkit* Glade [38] para la especificación de la GUI independiente de la plataforma.

En la Fig. 7 se muestra una vista de la herramienta en la etapa de nombramiento de clusters. En esta instancia, se han ejecutado las primeras cinco etapas de la estrategia automáticamente y se deben establecer los nombres de los clusters de acuerdo a la información provista por el sistema (etapa 6). En la captura es posible visualizar un conjunto de clusters nombrados y otros sin un rótulo definido. Usando la información provista en la interfaz que presenta el prototipo, es posible nombrar la totalidad de los clusters exhibidos. Para el caso de los clusters generados a partir de nodos manejadores, la elección siempre estuvo determinada por la información del componente GUI interactivo. Por ejemplo, uno de los clusters fue nombrado "*Save Project*", esto fue determinado a partir de la información del componente GUI interactivo (en este caso, un botón), ya que si bien este no posee rótulos, el identificador de la variable en el árbol GUI es *save_button* y el campo tooltip es "*save current proyect to a file"*. De la misma manera, otro cluster fue nombrado "*Execute*" (cluster seleccionado en la captura de la Fig. 7), cuyo componente GUI es de tipo "*button*", el identificador de la variable "*execute_button*" y el tooltip "*Execute current Tab*". En la mayoría de los casos, los tooltips sirven como documentación importante para determinar el nombre del cluster; claramente es información fuertemente relacionada con el Dominio del Problema debido a que es provista al usuario a través de componentes GUI del sistema bajo análisis. Por otra parte, para los clusters que no fueron creados a partir de un nodo manejador, se utilizó como principal criterio el nombre del nodo "fuente" de cada cluster. Por ejemplo, un cluster fue denominado "*Compare*" ya que era dependencia de "*Execute*" y estaba compuesto de un único método denominado "*compareExecute*" que referenciaba a una componente GUI y cuyo valor de importancia relativa era muy elevado por sobre la media en el proyecto. La estrategia detecta este tipo de clusters ya que son dependencia exclusiva de clusters manejadores y poseen relevancia dentro de la lógica subyacente del sistema. Otro cluster fue nombrado "*Display Page*" ya que: i) referenciaba distintas componentes GUI como paneles, pestañas y *notebook* (panel con pestañas) y ii) estaba compuesto por el nodo fuente "*displayCurrentNotebookPage*" y otros nodos como "*standardView*", "*evaluateView*", "*dataView*", etc.

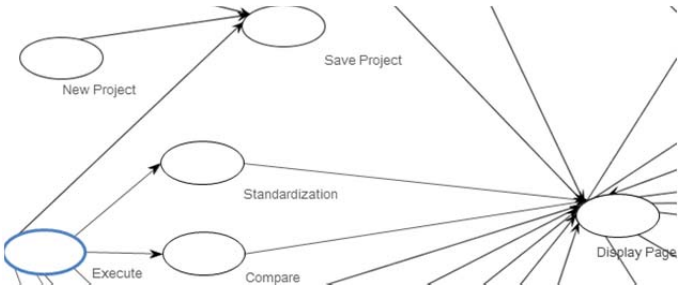

Figura 8. Conjunto de clusters analizados en este caso de estudio.

Con el objetivo de mantener el artículo autocontenido, se analiza un subconjunto de los casos de usos detectados por SSPIA para la herramienta Febrl. En la Fig. 8 se puede observar un acercamiento de la captura en la Fig. 7 para dicho subconjunto. De acuerdo a la parte de la transformación en la Fig. 5, los clusters "*Execute*", "*New Project*" y "*Save Project*" han sido generados a partir de nodos manejadores, por lo tanto, todos son relacionados por medio de una asociación con el usuario del sistema (actor). Las dependencias restantes en el GELMF son mapeadas a relaciones *include* como se especifica en la parte de la trasformación de la Fig. 4.

En la Fig. 9 se exhibe el subconjunto de casos de uso analizados en el párrafo previo (Fig. 8). El archivo XMI retornado por el prototipo se visualizó con la herramienta *Modelio* [40].

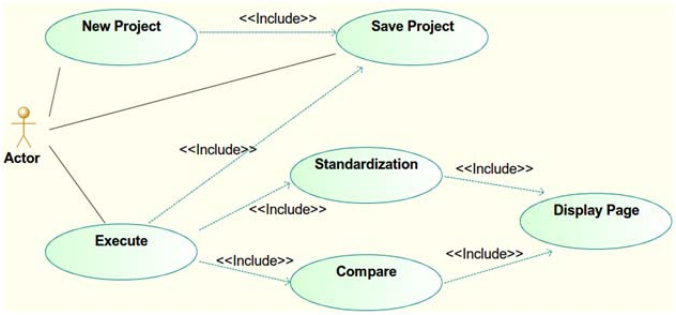

Figura 9. Vista acotada del modelo de caso de uso para la herramienta Febrl.

Para comprobar que el modelo obtenido refleja los casos de uso del sistema bajo análisis, se consultó la documentación provista por los desarrolladores y los trabajos de investigación que usan Febrl [39]; además se realizaron ejecuciones llevadas a cabo en tutoriales y guías del sistema. La funcionalidad principal de la herramienta pasa por el botón de ejecución ("*Execute*") y de acuerdo a la pestaña que se esté visualizando, se llevan a cabo las tareas que corresponden a la misma (por ejemplo, estandarización de los datos, comparación, evaluación, clasificación, etc.). De acuerdo al modelo exhibido en la Fig. 9, el usuario puede hacer uso de la funcionalidad "*Execute*", esta misma depende de "*Compare*" y esta última despliega los datos por medio del caso de uso "*Display Page*". Este patrón se repite para la mayoría de las funcionalidades que presenta la interfaz gráfica de Febrl.

Es importante destacar que el caso de estudio propuesto es una aplicación de amplio uso en el contexto de minería de datos [39]. El sistema presenta una GUI poco común, ya que posee escasos botones y administra la ejecución de cada funcionalidad de acuerdo a la pestaña que está activa. Por otra parte, también genera componentes de la interfaz de manera dinámica, por lo tanto, el comportamiento de las mismas sólo se puede determinar en tiempo de ejecución. Teniendo en cuenta estas características del sistema que se presentan como dificultades para el enfoque propuesto, la estrategia exhibió resultados interesantes mostrando una aproximación cercana a un modelo de caso de uso de la herramienta Febrl que facilita el entendimiento y análisis de la misma. Es posible diferenciar las distintas funcionalidades que ofrece distinguiendo los casos de uso más relevantes y las relaciones entre los mismos. Por otra parte, el prototipo también permite la visualización

del conjunto de métodos/funciones (como también de las dependencias) que componen cada caso de uso. De esta manera, se puede identificar los artefactos de software relacionados a cada funcionalidad del sistema, facilitando la comprensión y por lo tanto la ejecución de tareas de mantenimiento y evolución sobre el mismo.

VII. Conclusión

En este artículo se presentó SSPIA, una estrategia que asiste al ingeniero en la difícil tarea de comprender un sistema. La misma propone extraer cierto tipo de información del sistema como el Grafo Estático de Llamada a Métodos/Funciones (GELMF), el Árbol de componentes de la GUI, entre otros, para luego realizar agrupamientos sucesivos de nodos en el GELMF y finalmente lograr mapear dichos agrupamientos y sus relaciones a un modelo de caso de uso. La estrategia es implementada en un prototipo que integra técnicas de extracción de información estáticas, análisis y filtrado de información y una interfaz gráfica que facilita la interacción con el ingeniero de software. Una de las características más importantes de la estrategia es que gran parte de la misma es llevada a cabo mediante construcción y trasformación de modelos utilizando el enfoque *MDA*. Esta última característica permite independizar SSPIA de determinados lenguajes, paradigmas de programación y plataformas utilizados.

Para mostrar la aplicabilidad del enfoque se aplicó la estrategia SSPIA a la herramienta Febrl. El modelo de casos de uso resultante permite obtener las principales funcionalidades de la herramienta analizada y sus dependencias. El prototipo de SSPIA facilita la interacción con el ingeniero mediante una interfaz gráfica que provee la información y los componentes interactivos necesarios para una correcta inspección y nombramiento de los clusters.

La estrategia presentada permite al ingeniero de software visualizar un modelo de caso de uso del sistema analizado e identificar los componentes del Dominio del Programa estrechamente vinculados con los elementos de este modelo. Es decir, brinda una abstracción general y en conjunto permite identificar los artefactos de software relacionados con los componentes de dicha abstracción. Por lo tanto, la aproximación propuesta asiste al ingeniero en el arduo proceso de comprender un sistema.

SSPIA puede ser extendida en distintos aspectos para obtener una estrategia más precisa, robusta y efectiva para generar distintos modelos. Una posible extensión al trabajo es la extracción y análisis de información dinámica. Las técnicas de análisis dinámico proveen información relevante respecto del comportamiento del sistema en ejecución y fortalecería la información extraída con técnicas estáticas. De esta manera, se pretende integrar ambas técnicas de análisis con el objetivo de generar distintos tipos de modelos como diagramas de secuencia y actividad, mapas de casos de uso, visualización de trazas, etc. Claramente, estos modelos facilitarán aún más la comprensión brindando otras perspectivas del sistema bajo estudio. Además, se pretende proveer a la herramienta de un ambiente integrado de visualización e interacción con el ingeniero de software.

Por otra parte, con el objetivo de brindar otra perspectiva de la utilidad del enfoque, se pretende realizar sesiones de trabajo en donde se provea la herramienta de comprensión a un conjunto de ingenieros de software. A partir de esta experiencia, de acuerdo a la retroalimentación recibida por los ingenieros, llevar a cabo un análisis pormenorizado de las ventajas y desventajas de utilizar la herramienta propuesta.

REFERENCIAS

[1] Keith Bennett and Václav Rajlich. “Software Maintenance and Evolution: a Roadmap”. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE ’00, pages 73–87, New York, NY, USA, 2000. ACM.

[2] Gonzalo Salvatierra, Cristian Mateos, Marco Crasso, Alejandro Zunino, and Marcelo Campo. “Legacy System Migration Approaches”. *Latin America Transactions, IEEE (Revista IEEE America Latina)*, 11(2):840–851, 2013.

[3] Spencer Rugaber. “Program Comprehension”. *Encyclopedia of Computer Science and Technology*, 35(20):341–368, 1995.

[4] Mario Berón. “Program Inspection to interconnect Behavioral and Operational Views for Program Comprehension”. *Ph.D. Thesis Dissertation at University of Minho*. Braga. Portugal, 2010.

[5] Margaret-Anne Storey. “Theories, Methods and Tools in Program Comprehension: Past, Present and Future”. *Proceedings of the 13th International Workshop on Program Comprehension*, pages 181–191, 2005.

[6] Dragan Bojic and Dusan Velasevic. “A Use-case Driven Method of Architecture Recovery for Program Understanding and Reuse Reengineering”. In *Proceedings of the Conference on Software Maintenance and Reengineering,* CSMR ’00, pages 23-32, Washington, DC, USA. IEEE Computer Society. 2000.

[7] QVT. OMG. http://www.omg.org/spec/QVT/, 2015.

[8] Mark Shtern and Vassilios Tzerpos. “Clustering Methodologies for Software Engineering”. *Advances in Software Engineering*, pages 1-18, 2012.

[9] Ivan T Bowman and Richard C Holt. “Software Architecture Recovery Using Conway’s Law”. In *Proceedings of the 1998 Conference of the Centre for Advanced Studies on Collaborative Research*, CASCON ‘98 page 6. IBM Press, 1998.

[10] Periklis Andritsos and Vassilios Tzerpos. “Information-theoretic Software Clustering”. *Software Engineering, IEEE Transactions on*, 31(2):150–165, 2005.

[11] Dirk Beyer and Andreas Noack. “Clustering Software Artifacts Based on Frequent Common Changes”. In Proceedings of the 13th International Workshop on *Program Comprehension*, 2005. IWPC 2005. pages 259–268. IEEE, 2005.

[12] Jonathan Maletic and Andrian Marcus. “Supporting Program Comprehension Using Semantic and Structural Information”. In *Proceedings of the 23rd International Conference on Software Engineering*, pages 103–112. IEEE Computer Society, 2001.

[13] Adrian Kuhn, Stéphane Ducasse, and Tudor Gîrba. “Enriching Reverse Engineering with Semantic Clustering”. In *Reverse Engineering, 12th Working Conference on*, pages 10–pp. IEEE, 2005.

[14] Vassilios Tzerpos and Richard C Holt. “ACDC: an Algorithm for Comprehension-driven Clustering”. In *20th Working Conference on Reverse Engineering* (WCRE), pages 258–258. IEEE Computer Society, 2000.

[15] Andreas Christl, Rainer Koschke, and Margaret-Anne Storey. “Automated Clustering to Support the Reflexion Method”. *Information and Software Technology*, 49(3):255–274, 2007.

[16] Jingwei Wu, Ahmed E Hassan, and Richard C Holt. “Comparison of Clustering Algorithms in the Context of Software Evolution”. In *Proceedings of the 21st International Conference on Software Maintenance*, ICSM’05, pages 525–535. IEEE Computer Society, 2005.

[17] Maher Salah, Spiros Mancoridis, Giuliano Antoniol, and Massimiliano Di Penta. “Scenario-driven Dynamic Analysis for Comprehending Large Software Systems”. *In Proceedings of the 10th European Conference on Software Maintenance and Reengineering*, CSMR ‘06, pages 10–pp. IEEE, 2006.

[18] Mohammad El-Ramly, Eleni Stroulia, and Paul Sorenson. “Mining System-user Interaction Traces for Use Case Models”. *In Proceedings of*

the 10th International Workshop on Program Comprehension, pages 21–29. IEEE, 2002.

[19] Lu Zhang, Tao Qin, Zhiying Zhou, Dan Hao, and Jiasu Sun. "Identifying Use Cases in Source Code". *Journal of Systems and Software*, 79(11):1588–1598, 2006.

[20] Haiping Si, Yanling Li, Baogang Chen, and Wei Fang. "A Method of Use Case Oriented Semiautomatic Reverse Engineering". *Journal of Computational Information System*, 9(5):2093–2101, 2013.

[21] Qingshan Li, Shengming Hu, Ping Chen, Lihong Wu, and Wei Chen. "Discovering and Mining Use Case Model in Reverse Engineering". In Fourth International Conference on Fuzzy Systems and Knowledge Discovery, FSKD '07, volume 4, pages 431–436. IEEE, 2007.

[22] Philippe Dugerdil and David Sennhauser. "Dynamic Decision Tree for Legacy Use-case Recovery". In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1284–1291. ACM, 2013.

[23] Claudia, Pereira and Liliana, Martinez and Liliana, Favre. "Recovering Use Case Diagrams from Object Oriented Code: an MDA-based Approach". In *Eighth International Conference on Information Technology: New Generations*, ITNG '11, pages 737–742. IEEE, 2011.

[24] Enrique Miranda, Mario Berón, Montejano Germán, Riesco Daniel, and Debnath Narayan. "A Strategy for Detecting and Clustering Functionalities in Object Oriented Systems". *In 30th International Conference on Computers and Their Applications*, CATA '15, ISCA, volume 1, Honolulu, Hawaii, USA, 2015.

[25] Atif Memon, Ishan Banerjee, and Adithya Nagarajan. "GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing". *In 20th Working Conference on Reverse Engineering* (WCRE), pages 260–260. IEEE Computer Society, 2003.

[26] Enrique Alfredo Miranda, Mario Marcelo Berón, and Daniel Riesco. "Reverse Engineering Clustering Based on GUI". *Technical report*, Universidad Nacional de San Luis - CONICET, 2015.

[27] Karen Spärck Jones. "Automatic Summarizing: The State of the Art". *Inf. Process. Manage.*, 43(6):1449–1481, November 2007.

[28] Philippe Dugerdil and Julien Repond. "Automatic Generation of Abstract Views for Legacy Software Comprehension". In *Proceedings of the 3rd India Software Engineering Conference*, pages 23–32. ACM, 2010.

[29] Abdelwahab Hamou-Lhadj and Timothy Lethbridge. "Summarizing the Content of Large Traces to Facilitate the Understanding of the Behaviour of a Software System". In 14th *International Conference on Program Comprehension*, ICPC '06. pages 181–190. IEEE, 2006.

[30] JM Almendros-Jimenez and L Iribarne. "Designing GUI Components from UML Use Cases". *In 12th International Conference and Workshops on Engineering of Computer-Based Systems*, ECBS'05, pages 210–217. . IEEE Computer Society, 2005.

[31] JM Duarte, MG Tonanez, L Cernuzzi, and OP Lopez. "Evaluation of Software Development through an MDA Tool: a Case Study". *Latin America Transactions, IEEE (Revista IEEE America Latina)*, 6(3):252–259, 2008.

[32] Beatriz Mora, Felix García, Francisco Ruiz, Mario Piattini, Artur Boronat, JA Carsí, I Ramos, et al. "Software Generic Measurement Framework Based on MDA". *Latin America Transactions, IEEE (Revista IEEE America Latina)*, 9(1):864–871, 2011.

[33] Corina Abdelahad, Daniel Riesco, Alessandro Carrara, Carlo Comin, and Carlos Kavka. "Towards the Standardization of Industrial Scientific and Engineering Workflows with QVT Transformations". *International Journal on Advances in Software*, Volume 6, Number 1 & 2.2013

[34] Nemury Silega, Tania Teresa Loureiro, and Manuel Noguera. "Model-driven and Ontology-based Framework for Semantic Description and Validation of Business Processes". *Latin America Transactions, IEEE (Revista IEEE America Latina)*, 12(2):292–299, 2014.

[35] Roxana Giandini, Claudia Pons, and Gabriela Pérez. "A two-level Formal Semantics for the QVT Language". *In Proceeding of Conferencia Iberoamericana de Software Engineering,* CIbSE '09, pages 73–86, 2009.

[36] Superstructure Specification OMG. UML 2.4.1. http://www.omg.org/spec/UML/2.4.1/, 2015

[37] JUNG. http://jung.sourceforge.net/, 2015.

[38] Galde User Interface Designer. https://glade.gnome.org/, 2015.

[39] Febrl. ANU Data Mining Group. http://datamining.anu.edu.au/projects/linkage.html, 2015.

[40] Modelio. https://www.modelio.org/, 2015.

Enrique Miranda es Licenciado en Ciencias de la Computación de la Universidad Nacional de San Luis (UNSL), San Luis Argentina. Se desempeña como docente-investigador del Departamento de Informática (UNSL) en el cargo Jefe de Trabajos Prácticos. Es estudiante de doctorado en la carrera Doctorado en Ingeniería Informática (UNSL) y becario doctoral del CONICET. Los principales tópicos de investigación son Comprensión de Programas, Ingeniería Reversa, Ingeniería de Software, Lenguajes de Programación, entre otros.

Corina Abdelahad es Magíster en Ingeniería de Software de la Universidad Nacional de San Luis (UNSL), San Luis - Argentina. Se desempeña como docente-investigador del Departamento de Informática (UNSL) en el cargo Jefe de Trabajos Prácticos. Es estudiante de doctorado en la carrera Doctorado en Ingeniería Informática (UNSL). Los principales tópicos de investigación son Ingeniería de Software, MDA, Transformaciones a nivel metamodelo, entre otros.

Mario Berón es Doctor en Ciencias de la Computación de la Universidad Nacional de San Luís (UNSL) de Argentina con reconocimiento del mismo grado por parte de la Universidade do Minho de Portugal. Se desempeña como docente-investigador del Departamento de Informática (UNSL). Es docente en la Maestría en Ingeniería de Software (UNSL). Los principales tópicos de investigación son Comprensión de Programas, Ingeniería Reversa, Lenguajes de Programación, Seguridad Informática, Sistemas Embebidos, entre otros.

Daniel Riesco es Doctor de la Universidad de Vigo, España. Tiene una master de la Universidad Politécnica de Madrid, España. Es profesor de la Universidad Nacional de San Luis (UNSL), Argentina. Es director de un proyecto de investigación con más de 15 miembros. Es docente y codirector del Doctorado en Ingeniería Informática y la Maestría en Ingeniería de Software (UNSL). Tiene más de 100 publicaciones en congresos y revistas internacionales con referato, como IEEE, ACM, Springer, entre otros.