Estrategias para Relacionar el Dominio del Problema con el Dominio del Programa para la Comprensión de Programas

José Luis Albanes
Departamento de Informática
Universidad Nacional de San Luis
San Luis – Argentina
email: jlalbanes@gmail.com

Mario Marcelo Berón
Departamento de Informática
Universidad Nacional de San Luis
San Luis – Argentina
email: mberon@unsl.edu.ar

Pedro Rangel Henriques
Departamento de Informática
Universidade do Minho
Braga – Portugal
pedrorangelhenriques@gmail.com

Maria João Pereira
Departamento de Informática
Instituto Politécnico de Bragança
Bragança – Portugal
mjoao@ipb.pt

Resumen

La comprensión de programas es una disciplina de la ingeniería de software cuyo propósito fundamental es simplificar al programador la tarea de entender programas. Uno de los principales desafíos en el área de Comprensión de Programas consiste en relacionar dos dominios muy importantes como lo son: el Dominio del Problema y el Dominio del Programa. El primero hace referencia a la salida del sistema y el segundo a las componentes del programa utilizadas para producir dicha salida. En este artículo se presenta una línea de investigación cuya principal finalidad es el estudio, análisis, creación y desarrollo de estrategias de interconexión de dominios que faciliten el entendimiento del software. Este artículo se centra principalmente en aquellas que relacionan el dominio del problema con el dominio del programa.

Palabras Clave: Comprensión de Programas, Dominio del Problema, Dominio del Programa.

1. Contexto

Este trabajo se encuentra enmarcado en el contexto del proyecto Quixote: Desarrollo de Modelos del Dominio del Problema para Inter-relacionar las Vistas Comportamental y Operacional en Sistemas de Software a Fin de Facilitar su Comprensión. Quixote es un proyecto bilateral entre Argentina y Portugal que está avalado por el Ministerio de Ciencia, Tecnología e Innovación Productiva de la Nación Argentina (Mincyt) y la Fundação para a Ciência e Tecnología de Portugal (FCT). En dicho proyecto participan la Universidad Nacional de San Luis y la Universidad de Minho. Entre las tareas realizadas en el contexto del proyecto Quixote se encuentran diferentes misiones de investigación realizadas desde Argentina hacia Portugal y viceversa. Todas las misiones realizadas han sido solventadas por el Mincyt y la FCT. Se prevé en que en el transcurso de este año y a inicios del dos mil doce la realización de dos misiones de investigación. En la primera docentes de la Universidade do Minho visitarán la Universidad de San Luis para establecer un punto de situación de las investigaciones realizadas. En la segunda docentes de la Universidad Nacional de San Luis, particularmente del Área de Programación, Metodologías y Desarrollo de Software, visitarán la Univerisdade do Minho para elaborar los informes correspondientes al cierre del proyecto. Es importante notar que la Universidad de San Luis y la Universidade do Minho mantienen un vínculo de amistad y laboral desde hace aproximadamente cinco años y por lo tanto Quixote es otro resultado fructífero del vínculo establecido entre ambas ins-

tituciones.

Para finalizar esta descripción, es importante mencionar que Quixote surge como iniciativa de los integrantes del proyecto de investigación: Ingeniería de Software: Conceptos, Métodos y Herramientas en un Contexto de Ingeniería de Software en Evolución y del Grupo de Procesamiento de Lenguajes de la Universidade do Minho. Ambos grupos tienen una trayectoria exitosa en el contexto de la realización de trabajos conjuntos que implican: la elaboración de: tesis, publicaciones y proyectos (Quixote es fruto de esa experiencia).

2. Comprensión de Programas

La comprensión de programas se traduce en la habilidad de entender una pieza de código escrito en un lenguaje de alto nivel. Un programa no es más que una secuencia de instrucciones que serán ejecutadas de forma tal que se garantiza una determinada funcionalidad. El lector de un programa consigue extraer el significado del mismo cuando comprende de que forma el código cumple con la tarea para la cual fue creado [Wal02].

El área de comprensión de programas es una de las más importantes de la Ingeniería del Software, porque es necesaria para tareas de reutilización, inspección, manutención, migración y extensión de sistemas de software [VMV95, SFM, EKS01]. Puede también ser utilizada en áreas como ingeniería reversa o en ıa enseñanza de lenguajes de programación. La tarea de comprensión de programas puede tener diferentes significados y puede ser vista desde diferentes perspectivas. El usuario puede estar interesado en como la computadora ejecuta las instrucciones con el objetivo de comprender el flujo de control y de datos, o puede querer verificar los efectos que la ejecución tiene sobre el objeto que esta siendo controlado por el programa. Considerando estos niveles de abstracción, el desarrollo de herramientas versátiles de inspección visual de código es crucial en la tarea de comprensión de programas.

La Comprensión de Programas es una disciplina que tiene como objetivo, desarrollar Modelos, Métodos, Técnicas y Herramientas basados en un proceso cognitivo y de ingeniería con el propósito de facilitar el entendimiento del software a los desarrolladores o arquitectos de software. Existen muchos productos destinados a facilitar la comprensión de software [BHU07]. Sin embargo, en muchas situaciones, no es claro como las teorías cognitivas, estrategias de visualización y extracción de la información se instancian en dichos productos [SFM, Ext02, NG84].

2.1. El Desafío para la Comprensión de Programas

Actualmente se han realizado muchos trabajos que intentan facilitar la Comprensión de Programas [Sto05]. Esos trabajos presentan interesantes estrategias que exploran y visualizan el Domino del Programa dejando al programador la tarea de interpretar la tarea que realizan en el Dominio del Problema. También se pudo comprobar que son escasas las estrategias que intentan vincular (aunque sea débilmente) ambos dominios [LF94, SWM].

Por lo expuesto en el párrafo anterior se puede afirmar que el principal desafío en esta área consiste en Relacionar correctamente el Dominio del Problema con el Dominio del Programa. En otras palabras lo que se pretende relacionar es la salida del sistema (Dominio del Problema) con las componentes de software utilizadas para producir dicha salida (Dominio del Programa).

Una forma de concretar la relación mencionada previamente, consiste en reconstruir la relación real entre el Dominio del Problema y el Dominio del Programa, por medio de una relación virtual. Para alcanzar este objetivo se debe: i) Construir una representación del Dominio del Problema; ii) Construir una representación del Dominio del Programa y iii) Definir un Procedimiento de Vinculación. La aproximación descripta anteriormente se puede conceptualizar como un meta modelo de comprensión el cual puede ser instanciado con modelos particulares de comprensión, los cuales a su vez pueden ser utilizados como base para la creación de estrategias de comprensión. Esta conceptualización es la utilizada para el estudio y creación de estrategias de interconexión de dominios.

El artículo se encuentra organizado como se describe a continuación. La sección 3 presenta una descripción la línea de investigación, sus objetivos y resultados alcanzados. Finalmente, la sección 4 menciona las actividades, relacionadas con la formación

de recursos humanos, llevadas a cabo en el contexto del proyecto.

3. Línea de Investigación

Los investigadores en Ingeniería de Software sostienen que: *Un programador entiende un programa cuando puede relacionar la vista Comportamental (la salida del sistema), con la vista Operacional (las componentes del programa que se utilizaron para producir dicha salida).*

La afirmación realizada en el párrafo precedente se basa en que dicha relación permite que el programador pueda localizar rápidamente los objetos del programa que se utilizaron para una funcionalidad específica del sistema de estudio.

A través del estudio del estado del arte se ha podido detectar la existencia de algunas estrategias que permiten relacionar el Dominio del Problema con el Dominio del Programa [BPOdC10]. Entre esas estrategias se destacan:

- SVS (Simultaneous Visualization Strategy): Permite que las componentes de software usadas en tiempo de ejecución sean visualizadas mientras el sistema se ejecuta. Una debilidad de esta técnica es la imposibilidad de obtener explicaciones del comportamiento del sistema debido a la forma en la que se representa la información dinámica.

- BORS (Behavioral-Operational Relation Strategy): Requiere que el sistema sea ejecutado y que cierta clase de información dinámica, como por ejemplo las funciones usadas durante la ejecución del sistema, sea recuperada. Después de eso dicha información es procesada por algoritmos que construyen explicaciones relacionadas con la ejecución del sistema. En esta técnica existe una fase de inspección manual que recupera nombres de funciones relacionadas con un determinado concepto del dominio del problema lo que introduce complejidades en el proceso de comprensión. Es importante notar que dicho proceso de inspección se realiza utilizando la técnica del *grep*. Está técnica consiste en buscar palabras claves, por concordancia sintáctica, en el código fuente del programa utilizando el comando *grep* provisto por los sistemas operativos Linux, Unix, etc.

- SVSi (Simultaneous Visualization Strategy Improved): Esta estrategia es una combinación de SVS y BORS. A través de SVS se puede observar la salida del sistema y las funciones del programa usadas para producir dicha salida. Dichas funciones, son almacenadas en un archivo y luego interpretadas por los algoritmos utilizados por BORS para construir explicaciones. Es importante notar que SVSi elimina la necesidad de inspeccionar manualmente el Dominio del Problema, ya que permite visualizar la salida del sistema de estudio sincronizada con la lista de funciones llamadas.

A pesar de los resultados obtenidos, en cuanto a la interconexión de los Dominios del Problema con el Dominio del Programa se refiere, se piensa que es posible encontrar relaciones más fuertes que las actuales. Con esto se hace referencia a construir un modelo del Dominio del Problema y decorar ese modelo con las componentes del programa que implementan cada una de sus partes. La no trivialidad de este proceso llevó al grupo de investigación a profundizar los estudios relacionados con la elaboración de estrategias de interconexión del Dominio del Problema con el Dominio del Programa con las características antes mencionadas.

3.1. Análisis Comportamental

El **A**nálisis **C**omportamental (AC) es una propuesta de estrategia de interconexión del Dominios del Problema con el Dominio del Programa elaborada por el grupo de investigación. AC es una estragia muy importante porque provee una solución al problema de vincular un modelo del Dominio del Problema con las componentes del programa que implementan cada una de sus partes.

AC es una estrategia que permite relacionar el Dominio del Problema con el Dominio del Programa; utilizando información estática y dinámica del sistema de estudio.

La primera hace referencia a: tipos, variables, funciones, etc.; definidas en el código fuente del programa. La segunda se centra en la recuperación de las componentes de software usadas en tiempo de ejecución.

Para la extracción de información estática se utilizan técnicas de compilación tradicionales. Es decir se construye un analizador sintáctico con las accio-

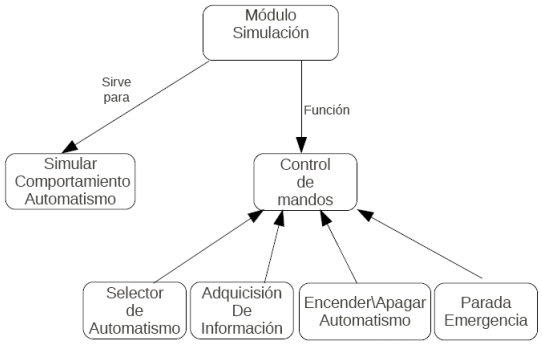

Figura 1: Mapa Conceptual de un Módulo de simulación perteneciente a un sistema SCADA.

nes semánticas apropiadas para extraer la información deseada.

Para la extracción de la información dinámica, se define un esquema de Instrumentación de Código. Esta técnica consiste en la inserción de sentencias dentro del código fuente del sistema de estudio con la finalidad de recuperar las componentes del programa que se utilizaron para producir la salida. Este esquema inserta funciones de inspección que muestran los nombres de las funciones ejecutadas por el sistema y otra información que el programador considere importante.

Básicamente el Análisis Comportamental consiste en: i) Reproducir cada comportamiento del sistema; ii) Registrar las operaciones realizadas y iii) Procesar la información con el objetivo de recuperar conocimiento. Por una parte, se utiliza una técnica de intrumentación de código como la descripta al comienzo de esta sección, para extraer la información dinámica del sistema. Por la otra, el programador debe ejecutar el sistema y usar todas sus funcionalidades. Para cada funcionalidad, se registra cada traza de ejecución y se asigna a cada una de ellas un nombre semántico.

Cuando se realizan todas las tareas explicadas previamente se aplican técnicas de análisis del flujo de ejecución de las funciones que posibiliten la extracción de la información más relevante. Con el procedimiento descrito previamente, el programador tiene solamente las funcionalidades del sistema aisladas con las componentes del programa asociadas. El lector puede observar que esta es una clase de relación entre los Dominios del Problema y Programa. Sin embargo, es posible mejorar esta correspondencia a través de la introducción de representaciones del Dominio del Problema, como por ejemplo los Mapas Conceptuales, Modelos de Dominio o Casos de Uso de UML decorados con las funciones del sistema que implementan cada concepto.

Para clarificar la idea descripta en el párrafo precedente, asuma que el sistema de estudio es un Módulo de simulación perteneciente a un SCADA (Sistema de Supervisión, control y Adquisición de datos) y que su mapa conceptual es el que se muestra en la Figura 1. Después (Antes o al mismo tiempo) de la construcción de la representación el programador puede ejecutar el sistema instrumentado para concretizar cada concepto. En otras palabras, el programador usa el sistema para: Seleccionar Automatismo, adquirir información, encender/apagar automatismo, etc. Para cada operación, se registran las trazas de ejecución y esta información se asocia a cada concepto.

El resultado es una representación del Dominio del Problema decorada con las componentes del programa empleadas para llevar a cabo cada concepto. El lector puede percibir que la relación comportamental-operacional que se alcanza es una relación comportamental-operacional con más semántica. Es posible hacer esta afirmación, porque el esquema conceptual describe el dominio del problema y por cada concepto se muestran las funciones utilizadas para concretarlo. El mismo procedimiento se puede aplicar usando herramientas estándares de ingeniería tal como los Diagramas de Caso de Uso de UML o el Modelo de Dominio. La ventaja de esta aproximación es que los resultados obtenidos se integran con más facilidad a los proyectos de desarrollo de software.

La implementación de este esquema basicamente requiere de: i) Técnicas de recuperación de las trazas de ejecución; ii) Estrategias de extracción de información estática que brinden información más precisa de los datos reportados por el análisis dinámico; y iii) Una herramienta de Modelado Conceptual.

Esta aplicación se puede implementar como parte de una Herramienta de Comprensión de Programas, en este caso tal vez sea posible asociar los elementos conceptuales con las componentes del programa automáticamente, decorando el modelo con información dinámica y estática proveniente del código fuente.

4. Formación de Recursos Humanos

Actualmente los temas abordados por esta línea de investigación están siendo desarrollandos como parte de una tesis de Maestría en Ingeniería de Software en la Universidad Nacional de San Luis. Además se está trabajando en conjunto con integrantes de otra línea de investigación que se encarga de estudiar técnicas de instrumentación de código. A través del trabajo colaborativo de la línea de investigación aquí presentada y la previamente mencionada se están desarrollando trabajos de licenciatura relacionados con la vinculación del Dominio del Problema con el Dominio del Programa para faciliar el entendimiento de los sistemas de software. Se espera en el futuro poder extender las investigaciones realizadas en esta línea de investigación con el propósito de elaborar tesis de grado y posgrado.

Referencias

[BHU07] M. Beron, P. Henriques, and R. Uzal. Program Inspection to interconnect Behavioral and Operational Views for Program Comprehension. *Technical Report*, 2007.

[BPOdC10] Mario M. Berón, Maria João V. Pereira, Nuno Oliveira, and Daniela da Cruz. Svs, bors, svsi: Three strategies to relate problem and program domains. In *Proceedings of the 2010 IEEE 18th International Conference on Program Comprehension*, ICPC '10, pages 60–61, Washington, DC, USA, 2010. IEEE Computer Society.

[EKS01] T. Eisenbarth, R. Koschke, and D. Simon. Aiding program comprehension by static and dynamic feature analysis. In *ICSM'01: Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01)*, page 602, Washington, DC, USA, 2001.

[Ext02] C. Exton. Constructivism and program comprehension strategies. *Program Comprehension, 2002. Proceedings. 10th International Workshop on*, pages 281–284, 2002.

[LF94] H. Lieberman and C. Fry. Bridging the gulf between code and behavior in programming. In *ACM Conference on Computers and Human Interface*, Denver, Colorado, April 1994.

[NG84] J. D. Novak and D. B. Gowin. *Learning How to Learn*. Cambridge University Press, 1984.

[SFM] M. A. Storey, F. D. Fracchia, and Müller. Cognitive design elements to support the construction of a mental model during software exploration.

[Sto05] M. A. 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.

[SWM] M. A. Storey, K. Wong, and Müller. How do program understanding tools affect how programmers understand programs?

[VMV95] A. Von Mayrhauser and A. M. Vans. Program comprehension during software maintenance and evolution. *Computer*, 28(8):44–55, 1995.

[Wal02] A. Walenstein. Theory-based analysis of cognitive support in software comprehension tools. In *IWPC '02: Proceedings of the 10th International Workshop on Program Comprehension*, page 75, Washington, DC, USA, 2002. IEEE Computer Society.