Documentación de Componentes: Una Aproximación Basada en Diagramas de Secuencia

Miguel Á. Pérez [*] Amparo Navasa [**] Juan M. Murillo [***]

Resumen

Los rápidos cambios que experimentan las reglas de negocio asociadas a las empresas, hacen que por una parte se incremente el número de nuevas soluciones software a construir, mientras que por otro lado aumente el tiempo y dinero destinado a la evolución de los sistemas existentes. La falta de información sobre el comportamiento preciso de los componentes utilizados dificulta la evolución de los sistemas desarrollados. De esta manera, añadir un nuevo componente o sustituir uno existente plantea problemas derivados por las nuevas interacciones que se dan para integrarlo en un sistema en funcionamiento. Este artículo presenta una herramienta que partiendo de los diagramas de secuencia y demás información obtenida en la fase de Análisis y Diseño, permite mejorar el conocimiento de los servicios de los componentes utilizados y de sus interacciones con el entorno en donde se utiliza. Así mismo, se estudia la posibilidad de agregación o sustitución, individual o mediante la utilización de conjuntos de componentes que cooperen para adaptarse a los nuevos requisitos del sistema. Con todo ello, se pretende minimizar los posibles problemas de integración de nuevos componentes en un sistema software.

Palabras clave: *componentes software, escenarios, reutilización, mantenimiento de sistemas, diagramas de paso de mensajes, autómatas.*

Abstract

Quick changes in enterprises business rules, cause two main effects. On one hand, the increasing number of new software solutions to be built. On the other hand, the money and time growth destined for software evolution. The lack of information about precise behavior of components used makes the evolution of systems built with them very difficult. This way, the adding of a new software component or the replacing of an existing one, create problems arise from new software interaction among the existing software systems and the components to be integrated. In this context, this paper presents a new tool. It starts with Sequence Diagrams and all the information obtained in Analysis and Design phases. Then, it allows the improvement of the knowledge about services offered by components and its interaction with the environment where they will be used. Also, the aggregation or replacement of individual or collective components from the systems is being studied. So, it is expected to minimize the integration problems of new components in a software system.

Keywords: *software components, scenarios, reuse, components based development, software evolution, message sequence charts and automata.*

[*] University of Extremadura, Avda. Universidad s/n - 10071 Cáceres (SPAIN), toledano@unex.es
[**] University of Extremadura, Avda. Universidad s/n - 10071 Cáceres (SPAIN), amparon@unex.es
[***] University of Extremadura, Avda. Universidad s/n - 10071 Cáceres (SPAIN), juanmamu@unex.es

1. Introducción

En el mundo actual, cada vez son mayores las restricciones de tiempo durante el desarrollo de sistemas software. Los cambios en las reglas de negocio, los problemas para mantener sistemas obsoletos y el entorno cambiante en el que se mueven las empresas, hacen que la construcción y el mantenimiento de sistemas evolucionen a gran velocidad para adecuarse a los nuevos requisitos. Para adaptarse a las nuevas realidades, la Ingeniería Software ha centrado sus esfuerzos en la disminución de los costes y tiempos de desarrollo. La creación de librerías de elementos software reutilizables [1] , fácilmente adaptables a cualquier entorno de ejecución es una de las viejas metas de la Ingeniería del Software. No obstante, a pesar del incremento experimentado en la construcción de repositorios de elementos software en los últimos tiempos, todavía no se ha universalizado su utilización.

Con la aparición del paradigma de desarrollo de sistemas basado en componentes software se han disminuido los tiempos de desarrollo, se ha incrementado la modularidad de los sistemas y se ha aumentado la calidad de las soluciones desarrolladas. Sin embargo, su utilización plantea nuevos problemas derivados de la dificultad de encontrar componentes que se adapten a los requisitos. Existen pocas herramientas fiables que permitan la comparación, búsqueda y selección de componentes de los repositorios. Esto hace que la mayoría de las veces los sistemas se terminen adaptando a los componentes disponibles.

En este contexto, la utilización de componentes software en el desarrollo de soluciones, es posible plantear dos opciones:

- El componente software a utilizar, ha sido desarrollado para residir en un repositorio en donde será localizado por los usuarios. En este caso, es posible que el usuario pueda, no sólo recuperar el componente, sino además tener acceso a detalles sobre su implementación, lo cual facilitará su adaptación al sistema.
- El tipo de componente sea Commercial Off-the-Shelf (C.O.T.S.). En esta ocasión, el código del componente no está disponible. El conocimiento al que tiene acceso el usuario se limitará a las especificaciones proporcionadas por el fabricante, y su evolución y mantenimiento serán realizados por los proveedores.

Este trabajo se centra en el desarrollo de sistemas utilizando el primer tipo de elementos software, ya que la utilización de componentes C.O.T.S., hipoteca su mantenimiento hacia terceras personas ajenas a los equipos de desarrollo.

La construcción de aplicaciones mediante componentes, requiere la identificación de requisitos de las reglas de negocio, además de la búsqueda y selección o implementación de los elementos que puedan cubrir las necesidades. En este tipo de desarrollos, la documentación creada durante la fase de Análisis y Diseño del sistema debe servir como herramienta que facilite no solo la documentación del software, sino también la evolución de los sistemas desarrollados.

El mantenimiento de sistemas desarrollados con componentes software, es una tarea que no se puede limitar al estudio sobre los servicios ofrecidos por los componentes y su posible correspondencia sintáctica con los requisitos del sistema; es preciso estudiar en detalle la compatibilidad entre el comportamiento del elemento a introducir y el sistema. Esa compatibilidad se debe basar además, en el estudio de las interacciones entre componente y sistema. El objetivo, planteado en este trabajo, es el de construir una herramienta que permita la comparación de los comportamientos de componentes software, expresados a partir de diagramas de paso de mensajes. La idea principal consiste en automatizar la información obtenida durante el desarrollo, de manera que permita la creación de documentaciones (individuales o colectivas) que puedan ser comparadas y de esta manera estudiar la integración de nuevos componentes en el sistema.

Para explicar con más detalle lo anteriormente expuesto, este artículo se organiza de la siguiente manera: en el punto dos se explica en detalle el problema para el que se ha construido la herramienta, en el punto tres se exponen los trabajos relacionados, el cuatro describe el funcionamiento de la herramienta, en el quinto apartado se recopilan las aportaciones de la herramienta y, para finalizar, en el sexto punto las conclusiones y los trabajos futuros. Por último se relaciona la bibliografía utilizada.

2. Exposición del Problema

En la actualidad existen metodologías como Catalysis [2], encargadas de desarrollar soluciones software a partir de la cooperación entre diferentes componentes, e incluso se pueden adaptar herramientas como UML [3, 4], diseñadas para describir sistemas orientados a objetos, como herramientas para la descripción de sistemas utilizando componentes. Estas aproximaciones se basan en la representación de los servicios ofrecidos por las diferentes interfaces de los componentes, en las especificaciones de los contratos de uso, así como en las descripciones y restricciones sintácticas. Estos trabajos, se desarrollan a partir de un exhaustivo estudio en las fases de Análisis y Diseño, sobre el comportamiento esperado de los elementos software que van a intervenir en el sistema. Sin embargo, ninguno de ellos permite la automatización de la información obtenida de manera que se puedan realizar procesos de comparación entre los comportamientos de diferentes elementos.

Para una correcta integración de un elemento software dentro de un sistema, se necesita cumplir tres condiciones. Primero que el elemento software ofrezca los servicios requeridos [5, 6, 7, 8]. Segundo, poder establecer correspondencia sintáctica con el sistema [9] y tercero que el comportamiento esperado del componente permita su correcta integración. Es, en este último punto, donde se incluye nuestro trabajo.

Los trabajos antes comentados, acaban con la comprobación de los servicios y la posible correspondencia sintáctica. Sin embargo, la integración de un componente en un sistema no se puede quedar ahí, se necesita además información sobre cómo se utiliza ese componente, y si su funcionamiento esperado es congruente con el sistema donde va a ser integrado. Es decir, se precisa estudiar con detenimiento las interacciones del componente a sustituir con el sistema, para ver si son compatibles. En este punto, los diagramas de interacción propuestos en UML y otras metodologías orientadas a objetos como OMT [10] describen el comportamiento esperado de los objetos en los diferentes escenarios (expresados mediante diagramas de paso de mensajes) que describen las situaciones que se producen en los sistemas [11].

Estos escenarios están pensados para facilitar la captura de requisitos y son una herramienta muy útil para facilitar la comunicación entre usuarios y desarrolladores, además de ser fáciles de representar y comprender. No obstante, los escenarios tal como los entendemos presentan una serie de limitaciones que dificultan su uso como herramienta para comparar comportamientos de diferentes elementos software:

- Los escenarios dan sólo una visión parcial del problema, de manera que es posible que se necesiten varios para describir una situación compleja. Esto puede provocar un número grande de escenarios.
- Los escenarios no dan una visión global del sistema, de manera que habrá que agruparlos para tener una visión completa de él.
- Los escenarios describen situaciones mediante instancias entre elementos, de manera que habrá que abstraer de ellos el comportamiento de los elementos de manera individual para poder usarlos en las comparaciones.

Sin embargo, una vez resueltas esas limitaciones, la información obtenida a partir de los diagramas de paso de mensajes puede ser pasada a representaciones formales [14], lógica temporal [12] y autómatas [13] sin pérdida de información. Esto nos va a permitir chequear el modelo desarrollado [15], comprobar la completitud de las especificaciones [16], las inconsistencias y los conflictos entre escenarios. Para poder realizar esas operaciones hay que vincular las informaciones obtenidas de los diferentes escenarios mediante el uso de etiquetas entre las diferentes transiciones.

Una vez automatizada la información, las operaciones de comparación entre las documentaciones de los elementos software candidatos y los comportamientos esperados en el sistema a mantener, se pueden reducir a operaciones entre autómatas. Estas se deben completar, al establecer correspondencia, con las restricciones expresadas en Object Constraint Language [17], de cada uno de los eventos descritos en los diagramas.

3. Trabajos Relacionados

Para describir el comportamiento y las interacciones entre elementos, existen diferentes herramientas como los diagramas de paso de mensajes (MSC-2000, [18]), o los diagramas usados en metodologías orientadas a objetos como UML [3] y OMT [10]. Los más completos, en cuanto a capacidad de representación, parecen ser los MSC-2000 (no permiten la representación de eventos síncronos) y los diagramas de secuencia de UML (carecen de etiquetas de estado) que permiten la concatenación de diferentes escenarios. Es por ello que la herramienta utilizada en el trabajo presenta una sintaxis propia que integra las ideas de los diagramas de secuencia de UML y las de los MSC-2000.

La mayoría de los trabajos, relacionados con la automatización de la información proporcionada por los diagramas de paso de mensaje, están relacionados con los sistemas de telecomunicaciones. Este tipo de sistemas fueron los primeros que se plantearon el uso de los diagramas como una forma sencilla y fácil de expresar sus flujos de información. De esta manera, trabajos como el proyecto ISAT [19] estaban dirigidos a validar y asegurar la calidad de los sistemas de telecomunicaciones desarrollados. Más tarde fueron surgiendo proyectos como SPIN [12] el cual, utilizando el lenguaje de especificación formal PROMELA, se encargaba de realizar validaciones de los sistemas especificados mediante lógica temporal. No obstante todo estas soluciones estaban enfocadas hacía la validación de sistemas reactivos, y no hacia generar descripciones individual de los elementos que intervenían en él.

En la actualidad existe el proyecto UML++ [20]. Uno de sus objetivos consiste en la síntesis de información a partir de diseños UML, sin embargo la mayoría de los esfuerzos en este proyecto se encamina a sintetizar información de los diagramas de estados y a usar los diagramas de secuencia como manera de estructurar las implementaciones. No se plantean la idea de generar informaciones individuales para los elementos que intervienen en el sistema.

Existen otras herramientas que se encargan de describir interacciones entre elementos que cooperan en un sistema. Trabajos como los del proyecto Rigi [21] describen las interacciones mediante grafos; otros como los del proyecto Shrimp [22] añaden, además, hipertexto para la descripción detallada de los arcos de información contenida en los grafos. Con todo ello, se facilita la comprensión de los elementos del sistema, así como sus interacciones, pero no se describe su comportamiento dinámico.

4. Propuesta

Para comprender la utilidad de la herramienta y su funcionamiento, supongamos que en un sistema ya desarrollado (del cual poseemos la documentación generada durante las fases

de Análisis y Diseño) necesitamos reemplazar uno o varios componentes software. Con esas especificaciones el usuario obtiene una descripción de los servicios que necesita. Utilizando cualquiera de los métodos enumerados en el punto dos, se puede realizar una selección. En ella se obtendrán una serie de componentes, los cuales a primera vista cumplen con los requisitos sintácticos y semánticos exigidos. En un principio puede parecer que el usuario puede escoger entre cualquiera de los componentes para ajustarlo dentro del sistema, sin embargo cada uno de ellos tiene un funcionamiento diferente que condiciona su integración. El usuario deberá estudiar el comportamiento de cada uno de ellos para comprobar cuál es el que se ajusta al dominio del problema. En este punto es donde entra nuestro trabajo.

Pero, llegados aquí, ¿es realmente necesario estudiar las interacciones entre los diferentes elementos software? Para contestar a esta pregunta tenemos que plantearnos cómo podemos integrar un componente software en un sistema sin estudiar el tipo de notificación de un mensaje (síncronos o asíncronos), el orden en el que se tienen que dar los eventos para que el sistema cumpla los requisitos, el tipo de mensaje entre elementos (permiso de ejecución, retorno o no de resultados, invocación de un método remoto,...) o los métodos ejecutados con origen y destino en el mismo elemento. Claramente, la integración de componentes en un sistema requiere el estudio de todos los eventos producidos desde y hasta el componente.

La siguiente cuestión es ¿cómo realizar ese estudio? Para representar el funcionamiento de los componentes, se han utilizado diagramas de paso de mensajes. En ellos, cada rectángulo representa un componente (identificado mediante el nombre), cada mensaje viene identificado por un nombre (una pantalla posterior describe la sintaxis del método), cada línea representa un tipo de mensaje (notificación, permiso de ejecución, retorno), y cada flecha el tipo de comunicación (síncrona o asíncrona). En los diagramas de la figura 1 sólo se han representado componentes, pero se podían haber utilizado también interfaces (dependientes de alguno de los componentes del sistema) y actores.

Para establecer relaciones entre el comportamiento expresado en diferentes escenarios, se han añadido etiquetas en las líneas de vida[1] de los elementos. Estas etiquetas ya se utilizan en otros trabajos [18, 13], para describir las condiciones por las que pasa un elemento o conjunto de elementos en un escenario. Cada una de esas etiquetas, representada mediante hexágonos, expresa la condición en la que se encuentra el elemento en el momento de enviar o recibir los eventos. Esas condiciones pueden ser expresadas utilizando O.C.L. o lenguaje natural, y servirán de referencia a la hora de establecer correspondencias entre sistema y componente.

Una vez obtenidos los diagramas de secuencia, se selecciona el componente (o componentes) cuyo funcionamiento se va describir y se obtienen las proyecciones de todos los escenarios en los que interviene. En el caso que nos ocupa serían las representadas en la figura 2.

Con las proyecciones obtenidas, se identifica la etiqueta que representa el estado inicial del componente y a partir de ahí se ordenan todos los mensajes enviados o recibidos por el componente que se está describiendo. Cada mensaje vendrá definido por la tupla:

<origen/destino, mensaje, tipo de mensaje>

Una vez establecido el orden, se construye el autómata resultante. Este autómata indeterminista se traduce en el autómata determinista equivalente, para facilitar las operaciones entre autómatas que se pretenden realizar. El siguiente paso es optimizar el autómata, este es el que se usará para comparaciones, de manera que las comparaciones entre diagramas se transformen en operaciones entre autómatas. En el ejemplo que se está realizando, el autómata resultante, partiendo de la etiqueta A1 sería el representado en la figura 3.

En los arcos del autómata resultado se han representado gráficamente los tipos de men-

[1] La línea de la vida de un componente en un diagrama, esta formada por el conjunto de mensajes, ordenados, emitidos y recibidos por ese componente dentro de un escenario. Se consigue realizando la proyección del componente dentro del diagrama.

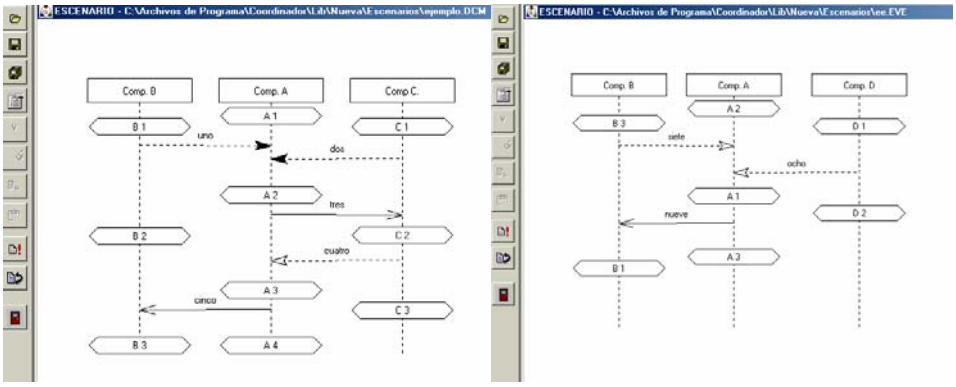

Figura 1: Representación de los escenarios usando etiquetas de condición.

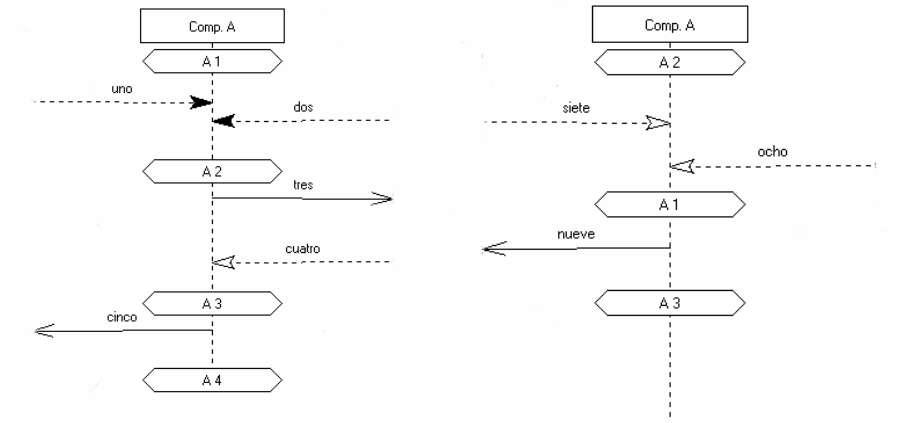

Figura 2: Líneas de la vida del componente A, obtenidas a partir de los escenarios descritos en la figura 1.

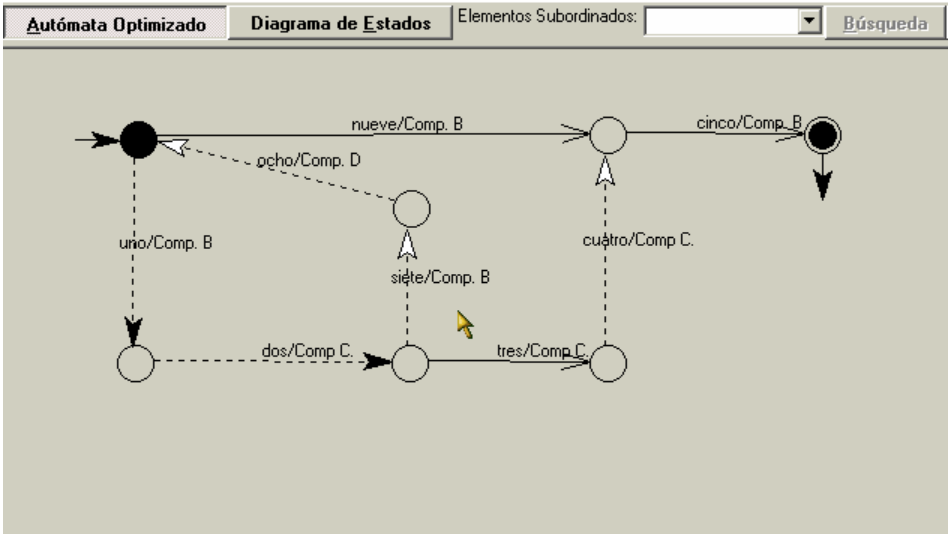

Figura 3: Autómata resultado.

saje, pero podía haberse incluido esa información en la etiqueta que existe entre los arcos de las transiciones. Haber elegido esta nomenclatura permite evitar la sobrecarga de información en las etiquetas entre transiciones sin afectar al comportamiento, ni a las operaciones a realizar.

La información obtenida se completaría con una detallada descripción sintáctica de cada uno de los métodos y parámetros definidos (Figura 4). Así mismo, se puede construir el diagrama de estados del elemento que está siendo descrito a partir de las etiquetas de estado.

De la misma manera que se ha documentado un componente, se podían haber seleccionado varios al mismo tiempo y haber generado la misma documentación para un componente que abarcase a los anteriores. Esto crearía un componente de mayor granularidad que ofreciese la unión de los servicios de todos los componentes que lo componen, y podrían ser integrados en un sistema de manera global, lo que ahorraría el estudio de las interacciones entre los elementos individuales que lo componen, disminuiría la complejidad del mantenimiento de los sistemas y facilitaría su comprensión.

5. Aportaciones del Trabajo

La evolución que experimentan los sistemas, hace que el mantenimiento sea la fase del ciclo de vida en el que más dinero invierten las empresas. El trabajo presentado pretende facilitar esta tarea mediante el estudio semántico y sintáctico de los componentes, apoyado en informaciones sobre el estudio de paso de mensajes esperados para facilitar su integración. La herramienta diseñada permite describir los requisitos y el funcionamiento esperado de un componente software a integrar en un sistema en ejecución, y comparar esa documentación con la de los componentes candidatos.

Otra de las aportaciones del trabajo es la posibilidad de ensamblar componentes para construir uno de mayor tamaño. De esta manera el estudio de su integración se limitará al paso de información entre el macrocomponente y el entorno, eliminando el estudio del paso de mensajes entre los componentes internos. Esto permitirá además, guardar información

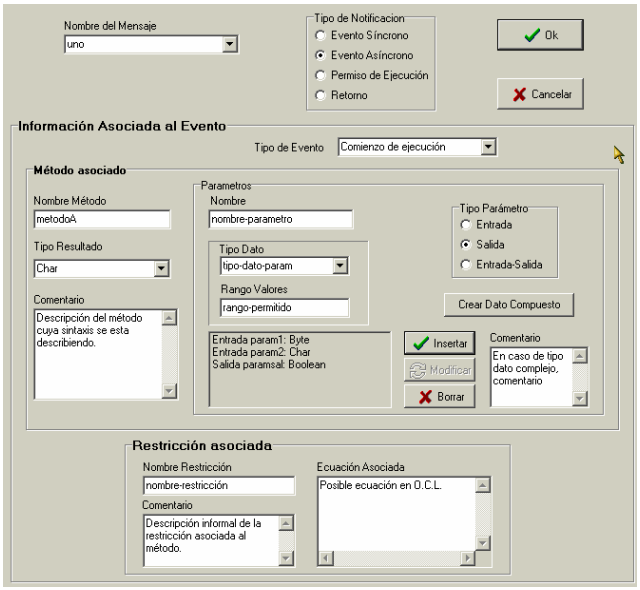

Figura 4: Descripción detallada de mensajes.

sobre los componentes de manera individual o formando parte de otros, lo que facilitará las búsquedas destinadas a encontrar componentes que satisfagan los requisitos solicitados.

Además, la herramienta construida no requiere un conocimiento especial por parte de los usuarios, ni a la hora de crear las documentaciones de los componentes, ni a la hora de operar con ellas. Las documentaciones de los componentes se pueden realizar durante las fases de Análisis y Diseño, utilizando diagramas de secuencia, que es una herramienta utilizada en muchas metodologías de desarrollo de software y no requiere un aprendizaje por parte de los usuarios finales. Y por otra parte, los autómatas generados pueden ser invisibles para los usuarios limitándose las operaciones de comparación entre componentes a devolver resultados, indicando el grado de congruencia con los componentes del repositorio.

Por último, las operaciones entre los autómatas pueden relajarse o restringirse dependiendo del grado de abstracción con el que se hayan expresado los diagramas de secuencia que contienen los requisitos, ya que salvo la descripción inicial del componente que debe detallar el comportamiento completo, las operaciones de comparación se pueden realizar a partir de descripciones parciales de componentes e ir refinando la búsqueda.

6. Conclusiones y Trabajos Futuros

Este trabajo presenta una herramienta que permite documentar componentes software de manera que sea posible su reutilización en nuevos dominios de aplicación. Permite además comparar los requisitos de usuario con los comportamientos expresados en las documentaciones de los componentes software residente en repositorios, de tal forma que se pueden realizar operaciones de búsqueda y selección utilizando la herramienta construida.

Los trabajos futuros se deben centrar en construir un repositorio de componentes software documentados de mayor tamaño que el utilizado para las pruebas, de manera que se pueda utilizar la herramienta en un entorno de ejecución mucho más amplio. Así mismo, se está trabajando en traducir la herramienta para que adapte y extienda a la sintaxis propuesta

de U.M.L, facilitando así su entendimiento por parte de los usuarios, y simplificando la integración de la herramienta en otras herramientas de desarrollo de software. Por último, se está trabajando en la formalización de todas las operaciones descritas y utilizadas por la herramienta construida.

Referencias

[1] M. D. McIlroy. *Mass-produced Software Component. In Software Eng. Concepts and Techniques*, N.A.T.O. 1968.

[2] C. W. Francis D´Souza. *Objects, Components and Frameworks with UML. The Catalysis Approach.* http://www.catalysis.org.

[3] G. Booch, J. Rumbaugh, and I. Jacobson. *El Lenguaje Unificado de Modelado.* Addison Wesley Iberoamericana. ISBN 84-7829-028-1. Madrid 1999.

[4] J. Cheesman, J. Daniels. *UML Components, A Simple Process for Specifying Component-Based Software.* Addison-Wesley, Pearson Education October 2000. ISBN 0-201-70851-5.

[5] J. S. Phasier: *A System for Interactive Document Retrieval using Keyphrases.* In Proceeding of 22nd Annual International ACM SIGIR Conference of Research and Development in Information Retrieval.ISBN: 1-58113-096-1, 1999. Pages 160-167.

[6] A. Podgursky, L. Pierce. *Retrieving reusable software by sampling behaviour.* ACM Transaction on Software Engineering and Methodology 2(3),286-303, 1993.

[7] Y. Pai and P. Bai. *Retrieving software components by execution.* Proceedings of the 1st Component User Conference, Munich 1996. Pag 39-48.

[8] M. Rittri. *Using types as search keys in function libraries. Journal of Functional Programming* 1(1): 71-89, January 1991.

[9] A.M. Zaremski and J.M. Wing. Signature Matching, a Tool for Using Software Libraries. 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering, pages 6-17. October 1995. Also CMU-CS-95-127.

[10] J. Rumbaugh, M. Blaha.*Modelado y Diseño Orientado a Objetos, metodología OMT.* Ed. Prentice may, 1995. ISBN:013-240698-9.

[11] K. Koskimies, T. Männistö, T. Systä and J. Tuomi. *On the Role of Scenarios in Object-Oriented Software Design.* Technical Report A-1996-1, University of Tampere, Department of Computer Science, January 1996.

[12] *Formal Verification, Spin Project.* http://spinroot.com/spin/whatispin.html

[13] I. H. Krüger. *Distributed System Design with Message Sequence Charts.* PhD Thesis. University of, July 2000.

[14] R. Alur, K. Etessami, and M. Yannakis.*Inference of message sequence charts.* In 22ndInternational Conference on Software Engineering, pages 304-313, 2000.

[15] S. Uchitel, J. Kramer. *A Workbench for Synthesising Behaviour Models from Scenarios.* 23rd International Conference on Software Engineering, Toronto, Canada, 2001.

[16] S. Uchitel, J. Magee, and J. Kramer. *Detecting Implied Scenarios in MSCs Specifications*. Department of Computing, Imperial College, 2001.

[17] J. Warmer, A. Kleppe. *The Object Constraint Language*. Precise Modeling with the UML. Addison-Wesley, ISBN 0-201-37940-6.

[18] Project: Animation of Sequence *Diagrams in the MSC 2000 Standard*. http://www.astec.uu.se/astec-testing/msc02-semantics.html

[19] *ISAT: Interactive Specification Acquisition Tools Project*. http://www.research.att.com/hall/isat-project.html

[20] *UML++ Project. Techniques for UML Based Software Development*. http://practise.cs.tut.fi/umlpp/index.html.

[21] K. Wong. *The Rigi User's Manual - Version 5.4.4*. June 30, 1998. http://www.rigi.csc.uvic.ca/Pages/publications.html.

[22] *Shrimp Project*. http://www.cs.princeton.edu/shrimp.