Creación Automática de Modelos DEVS para la Evaluación de Arquitecturas de Software Especificadas en Use Case Maps

Resumen: *En este trabajo se propone la creación de una herramienta de software, UCM2DEVS, que tiene como finalidad generar entornos de simulación para arquitecturas de software especificadas mediante la notación Use Case Maps (UCM). El modelo de simulación es generando automáticamente empleando el formalismo DEVS (Especificación de Sistemas de Eventos Discretos). A partir de escenarios especificados en UCM, la simulación de la arquitectura en DEVS permite la obtención de métricas sobre el desempeño de la arquitectura del software. De esta manera, el entorno de simulación asiste íntegramente al diseñador en el análisis de la calidad de arquitecturas de software, considerando en el análisis aspectos funcionales, no funcionales y cuantitativos.*

Palabras Claves: *Use Case Maps, DEVS, Generación automática de modelos, Evaluación de Arquitecturas de Software.*

Abstract: *In this work we propose the creation of a software tool, UCM2DEVS, which aims to generate simulation envronments for software architectures, specified by the notation Use Case Maps (UCM). The simulation model is automatically generated using the DEVS (Discrete Event System Specification) formalism. Starting from scenarios specified in UCM, simulation of architecture allows architects to obtain dynamic metrics on software architecture. Therefore, the simulation environment supports the designer to analyze the quality of software architectures. The analysis integrates functional, non-functional and quantitative aspects of software architectures.*

Keywords: *Use Case Maps; DEVS; Automatic generation of models; Evaluation of software architectures.*

María de los M. Becske Enrietto

Instituto INGAR–UTN, Conicet - Avellaneda 3657, Santa Fe.

Facultad Regional Santa Fe, UTN

E-mail de contacto: milagros.bec@gmail.com

INTRODUCCIÓN

En la actualidad muchos de los problemas de la producción y planificación de software se basan en errores introducidos en el diseño del software. Dado que la gran mayoría de los errores no puede detectarse hasta antes de la implementación o prueba del sistema, la corrección de estos errores suelen ser de costos elevados. Si el diseñador del sistema contara con las herramientas necesarias para la detección temprana de estos errores, se ahorraría dinero y aumentaría la oportunidad de entrega de los sistemas de software en tiempo y forma.

La arquitectura de software se define como la estructura o estructuras del sistema, las cuales contienen componentes de software, propiedades visibles externamente de esos componentes y las relaciones entre ellos (Bass et al., 2012). Por tal motivo, la simulación de dicha arquitectura puede ser tomada como una herramienta potente para cuantificar las propiedades de la arquitectura diseñada, siendo esto fundamental para el cumplimiento de los requerimientos de calidad del sistema, y mejorar la calidad del producto de software final.

Propuestas actuales (Bogado, 2013; Bogado et al., 2014; Bogado et al., 2011; Bogado et al., 2013) plantean el uso del framework DEVS (Zeigler et al., 2000), para la generación de un entorno de simulación a partir de arquitecturas de software especificadas con la notación de Use Case Maps (UCM), Dicha notación cubre los espacios que quedan entre los diagramas de Casos de Uso y los de Clase y Actividad de UML (Buhr, 1998), generando un puente entre los requerimientos y el diseño al permitir especificar cómo funciona el sistema sin incluir detalles de cómo se construye el mismo.

En las propuestas, mencionadas anteriormente, se presentan los lineamientos para la definición de un modelo DEVS a partir de una especificación en UCM. Sin embargo, para facilitar el trabajo de evaluación, es necesaria la generación automática de tales modelos de simulación, siendo de esta manera el empleo de DEVS transparente al arquitecto de software.

En este trabajo se propone, a partir de las contribuciones nombradas previamente, la creación y definición de la herramienta de software, UCM2DEVS. La misma, a partir de un modelo UCM (que representa la arquitectura del software a evaluar y un conjunto de parámetros que definen métricas de entidades del sistema), genera el modelo de simulación DEVS, para la evaluación de atributos de calidad del sistema. En este trabajo se propone la traducción de un modelo de variabilidad especificado en Kconfig a un modelo de características especificado en SPLOT, para poder luego analizar su variabilidad. En consecuencia, en primer lugar se analizan ambos modelos de representación: el modelo de características y su especificación en SPLOT, y el modelo de variabilidad en Kconfig. Luego, se incluye la representación de Kconfig mediante un modelo de características.

Modelo de Características

Un modelo de características es una representación compacta de todos los productos de una SPL en términos de características (Features). Una "Feature" es definida como un aspecto prominente o distintivo visible por el usuario, calidad, o característica de un sistema de software o sistema (Kang et al., 1990).

Las características se representan mediante una estructura de árbol, en donde se vinculan características "padres" con sus "hijos". Además pueden existir restricciones que afecten a dos o más características de cualquier lugar del modelo. Las relaciones permitidas entre características son las siguientes:

- Obligatoria: Relación que vincula un característica hijo con su padre, indicando que el hijo aparecerá en todos los productos en los que el padre esté incluido.

- Opcional: Relación que vincula una característica hijo con su padre, indicando que el hijo podrá opcionalmente estar incluidos en algunos productos en los que esté incluido su padre.

- Cardinalidad grupal: La cardinalidad grupal es un intervalo denotado por [min … máx] que limita el número de características hijo que pueden ser incluidas en un producto, cuando su padre está incluido, siendo los límites inferior y superior respectivamente.

- Requiere: Esta relación indica una implicación entre una característica requerida y una característica restringida. La característica restringida sólo puede incluirse en aquellos productos en los que se encuentre la característica requerida.

- Excluye: Esta relación indica una exclusión mutua entre dos características.

DESARROLLO

UCM2DEVS está conformado por tres componentes, Parser, Transformador y Generador, implementando una arquitectura "Pipe & Filters". En primera instancia, se toma como entrada la arquitectura del sistema que se detalla en UCM, un archivo en formato xml. Es posible generar tal arquitectura mediante el uso de la herramienta jUCMNav (2009); la misma es un editor gráfico de UCM, que se adquiere como plugin en el entorno de desarrollo Eclipse. Esta entrada se ingresa al primer elemento de la arquitectura de UCM2DEVS, un Parser (Figura 1). El parser se compone de dos partes, el analizador léxico, implementado en JLex (Berk, 2003), y el analizador sintáctico, implementado en CUP (Hudson et al., 1999), siendo ambos basados en Java.

El analizador léxico, es el encargado de identificar los diferentes elementos que conforman a la arquitectura de software representada en el modelo UCM, creando múltiples tokens para dicho fin. Con este motivo, se especificó en el analizador sintáctico la gramática libre de contexto para representar la sintaxis de los archivos generados por jUCMNav.

Una Responsabilidad (Figura 2) es una declaración general sobre un objeto de software: una acción que realiza el elemento, un conocimiento que mantiene sobre algo o una decisión importante que afecta a otro objeto de software. Una responsabilidad tiene asociado un conjunto de parámetros, que son empleados para modelar el comportamiento de diferentes aspectos dinámicos de la ejecución del software (Buhr, 1998).

Figura 2. Responsabilidad representada en UCM, adaptado de Bogado (2013).

Un elemento del diagrama del tipo Responsabilidad, es especificado en el analizador léxico mediante la secuencia que se puede observar en la Figura 1 (Parte izquierda), de "<responsibilities" (B_RESP), seguida del identificador (ID), nombre (NAME), número del nodo (NUMB_NODE); luego si los posee, metadatos asociados a las responsabilidad (B_MD y metadatas), que se utilizan como parámetros de simulación. Finalmente termina con la secuencia "/>" (E_RESP). Estos elementos son identificados por el analizador léxico, y enviados al analizador sintáctico (Parser).

Los tokens son tomados como entrada para el analizador, quien se encarga de la creación de los objetos que corresponden al modelo UCM. En la Figura 1 (Parte derecha), podemos observar cómo se codifican los elementos Responsabilidad en dicho analizador, en este se incluye parcialmente la gramática especificada

Figura 1. Especificación parcial del Parser.

junto al comportamiento vinculado a las distintas reglas de producción. Se detalla el símbolo inicial de la misma, UCMSpecification, y los símbolos no terminales responsibilities y responsibility.

Como se detalló anteriormente, del Parser resultan los objetos correspondientes al modelo detallado en la notación UCM, los cuales se generan siguiendo la estructura que se detalla en la Figura 3. En esta figura, se muestra un modelo conceptual propuesto por Bogado (2013) que integra los elementos estructurales, funcionales de UCM con información cuantitativa para la obtención de indicadores de calidad a partir de una arquitectura de software desde un punto de vista dinámico.

Las instancias de este modelo, son tomadas por el segundo elemento, el Transformador, que traduce los distintos elementos arquitectónicos en los modelos DEVS. Como salida de este componente se crean las clases relacionadas al modelo DEVS a generar, dichas clases se encuentran detalladas en la Figura 4 (Parte izquierda).

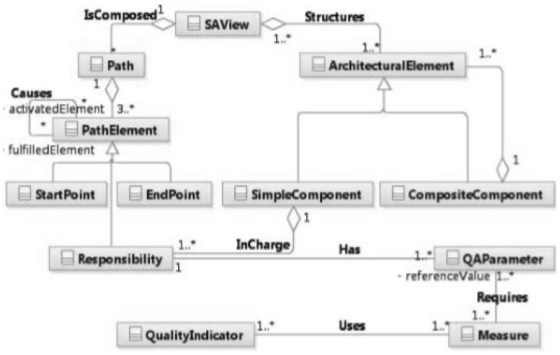
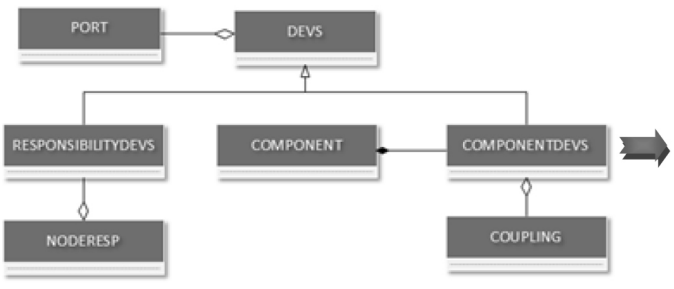


Figura 3. Conceptos para la evaluación de arquitecturas de software especificadas en UCM, adaptado de Bogado (2013).

En la Figura 5, se muestra cómo se estructura el entorno de simulación, SAESE (SAE - Simulation Environment), el cual es una jerarquía de modelos DEVS propuesta en Bogado (2013). Los dos elementos que lo componen son la vista de la arquitectura (SAVSM, SAView - Simulation Model) y el marco experimental (SAEEF, SA Evaluation - Experimental Frame). El modelo de simulación (SAVSM) representa la entidad




```
public class ComponentDEVS extends DEVS{
    public Vector<Coupling>inputCouplings =
                                new Vector<Coupling>();
    public Vector<Coupling> outputCouplings =
                                new Vector<Coupling>();
    public Vector<Coupling> internalCouplings =
                                new Vector<Coupling>();

    public ComponentDEVS(String name){
        this.name=name;
    }

    public void addCoupling(String couplingType, DEVS
                                devsTo, DEVS devsFrom){
        Coupling aCoupling = new Coupling (couplingType,
                                devsTo, devsFrom);
        if(couplingType.equals("inputCouplings")){
                inputCouplings.add(aCoupling);
        }
        else if(couplingType.equals("outputCouplings")){
                outputCouplings.add(aCoupling);
        }
        else{
                internalCouplings.add(aCoupling);
        }
    }
}
```

Figura 4. Estructura para la creación de los modelos DEVS, más el código parcial de la Clase ComponentDEVS.

bajo estudio, es decir, el software representado por su arquitectura y los escenarios funcionales. Por otro lado, el marco experimental (SAEEF) define el ambiente con el cual el software simulado interactúa, resume las medidas tomadas en los elementos del modelo y calcula los indicadores de calidad.

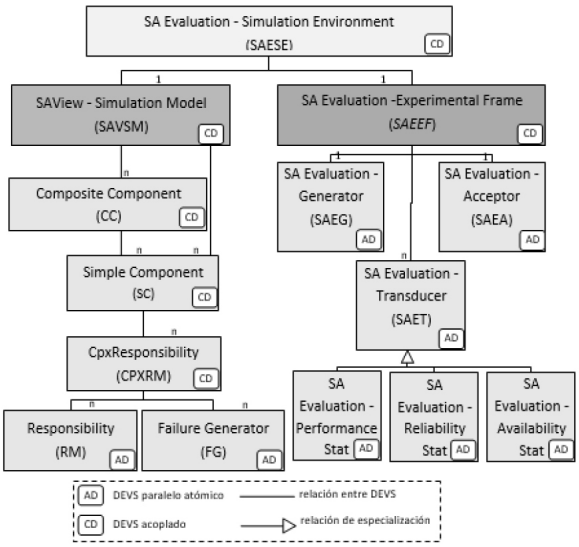


Figura 5. Jerarquía de elementos de simulación y su relación con los elementos conceptuales de la arquitectura de software, Bogado (2013).

En la Figura 6 se brinda un ejemplo de un software de gestión de licencias (LMSystem) detallado en la notación UCM (Bogado, 2013). En el ejemplo se pueden observar los siguientes parámetros pertenecientes a la responsabilidad R1: i) tiempo de caída (ref_downtime), el mismo es el tiempo de referencia que se emplea para calcular el tiempo de caída de la responsabilidad debido a fallas; ii) tiempo de ejecución (ref_execution_time), el cual se define como el tiempo de ejecución de referencia empleado para calcular el tiempo que una responsabilidad demora en responder a cada solicitud dada; iii) tiempo medio entre fallas (ref_mtbf), que es el tiempo medio entre una falla y otra; iv) tiempo de recuperación (ref_recovery_time), el cual es el tiempo de referencia que se emplea para determinar el tiempo de recuperación de una responsabilidad.

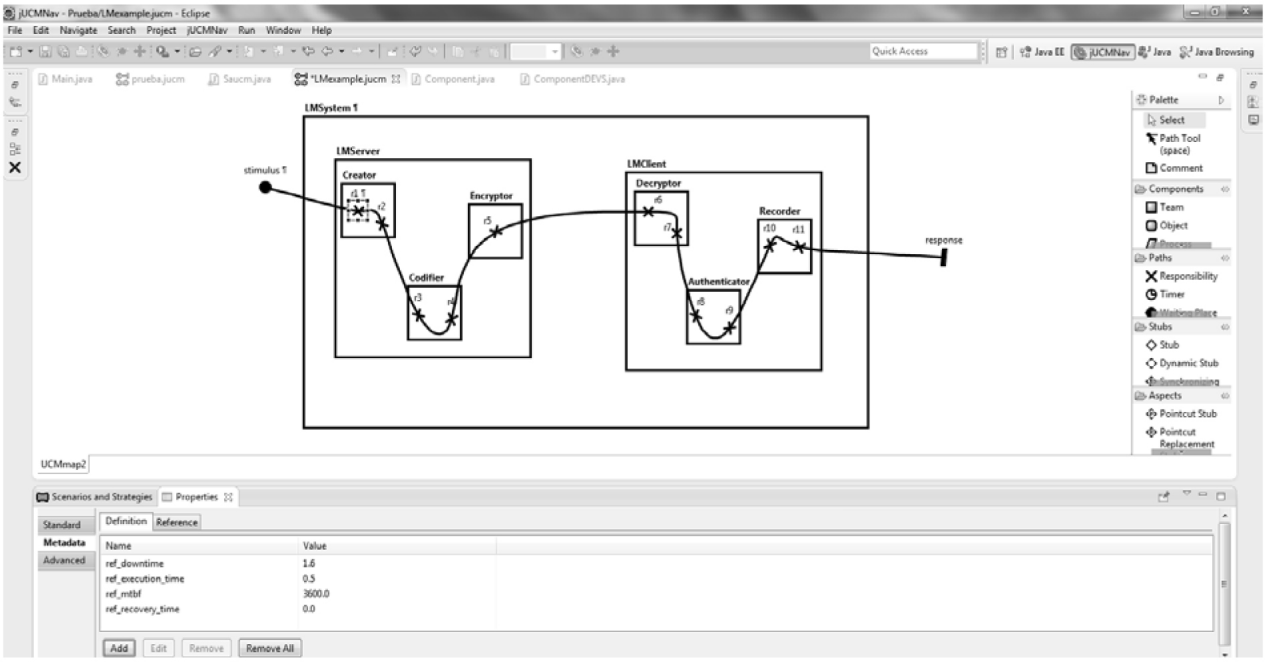


Figura 6. Arquitectura de Software especificada en UCM más el entorno de ejecución especificado como metadatos.

El modelo de entrada o vista de la arquitectura del sistema en UCM, especifica la estructura y los escenarios relevantes del sistema. La descripción de cada escenario incluye el estímulo de entrada que activa al sistema iniciando la ejecución del escenario. En UCM, se especifican los distintos escenarios mediante caminos (Path, Figura 3). Los caminos vinculan por medio de asociaciones de causa-efecto las responsabilidades contenidas en los componentes de la arquitectura. El inicio de un camino es un elemento que especifica la espera de un estímulo (StartPoint, Figuras 1 y 3). Su efecto inmediato es la ejecución de la primera responsabilidad del camino. Esta, una vez ejecutada, habilita la ejecución del siguiente elemento del camino (PathElement, vinculados por la asociación Causes, Figura 3). El camino finaliza cuando se emite una respuesta, final del escenario es alcanzado (EndPoint, Figuras 1 y 3).

Retomando la Figura 6, se puede observar un camino que posee los siguientes elementos: "stimulus", como elemento inicial del camino (StartPoint), luego el camino continúa su trayecto ingresando a los componentes LMSystem, LMServer y Creator; este último posee los próximos elementos del camino que son las responsabilidades "r1" y "r2", luego de cumplir dichas responsabilidades el camino ingresa al componente Codifier, el cual contiene a las responsabilidades "r3" y "r4". El camino continúa con "r5", contenida por Encryptor, para salir del componente LMServer y conectarse con LMClient; dentro de LMClient el camino prosigue con "r6" y "r7" (en Decryptor), luego con "r8" y "r9" (en Authenticator), y después con "r10" y "r11" (en Recorder). Al finalizar estas dos últimas responsabilidades, el camino sale de LMClient y LMSystem para terminar con el elemento final (EndPoint), denominado "response".

Además de los parámetros vinculados a las responsabilidades, puede incorporarse información asociada a la ejecución del software para otros elementos del UCM. Por ejemplo en un camino, es posible vincular al elemento StartPoint, la tasa de arribos de requerimientos de ese escenario.

Como ya detallamos anteriormente, el modelo UCM visto en la Figura 6 es utilizado como entrada para el Parser de UCM2DEVS, que genera como salida los objetos correspondientes a la Figura 3. Luego estos son tomados como entrada en el segundo componente, el Transformador, que es el encargado de la creación de los modelos DEVS. Finalmente, el Generador, tiene la función de crear el entorno de simulación DEVS.

En la Figura 7, se puede observar un fragmento del código correspondiente a la especificación del modelo DEVS en Java del Componente LMServer (Figura 6). Dicha especificación del modelo de simulación se implementa en la herramienta DEVS-Suite (DEVSSuite, 2011), la cual provee el conjunto de paquetes, incluyendo el paquete original DEVSJAVA para implementar los modelos DEVS usando el lenguaje de programación Java. Además, se pueden observar diferentes puertos de entrada (peip) y salida (seop, taop, dtop, failop) que lo componen.

Por último, la Figura 8 (Parte de arriba) ilustra la implementación del entorno de simulación (License Manager) usando la herramienta DEVS-Suite, donde se configura para su ejecución. Los datos para la configuración fueron estimados a partir de información de sistemas de similares características, componentes y funcionalidades.

La ejecución del entorno de simulación permite la evaluación de la arquitectura, brindando estimaciones de indicadores de calidad relacionados a atributos del software (Figura 8, parte de abajo).visto en la Figura 6 es utilizado como entrada para el Parser de UCM2DEVS, que genera como salida los objetos correspondientes a la Figura 3. Luego estos son tomados como entrada en el segundo componente, el Transformador, que

es el encargado de la creación de los modelos DEVS. Finalmente, el Generador, tiene la función de crear el entorno de simulación DEVS.

En la Figura 7, se puede observar un fragmento del código correspondiente a la especificación del modelo DEVS en Java del Componente LMServer (Figura 6). Dicha especificación del modelo de simulación se implementa en la herramienta DEVS-Suite (DEVSSuite, 2011), la cual provee el conjunto de paquetes, incluyendo el paquete original DEVSJAVA para implementar los modelos DEVS usando el lenguaje de programación Java. Además, se pueden observar diferentes puertos de entrada (peip) y salida (seop, taop, dtop, failop) que lo componen.

Por último, la Figura 8 (Parte de arriba) ilustra la implementación del entorno de simulación (License

```
import java.awt.*;
import view.modeling.ViewableComponent;
import view.modeling.ViewableDigraph;

public class LMServerDEVS extends ViewableDigraph{
        public LMServerDEVS(){
                super("LMServer");
                construct();
        }

public LMServerDEVS(String nm){
        super(nm);
        construct();
}

public void construct(){
        //input ports
        addInport("peip");

        //output ports
        addOutport("seop");
        addOutport("taop");
        addOutport("dtop");
        addOutport("failop");
        //Creating Components
        //assigning the responsibilities to the simple
                                component (present model)

        ViewableDigraph Creator = new CreatorDEVS();

        ViewableDigraph Codifier = new CodifierDEVS();

        ViewableDigraph Encrytor = new EncryptorDEVS();
```

Figura 7. Especificación parcial un modelo DEVS, automáticamente generado a partir del Componente LMServer.

Manager) usando la herramienta DEVS-Suite, donde se configura para su ejecución. Los datos para la configuración fueron estimados a partir de información de sistemas de similares características, componentes y funcionalidades.

La ejecución del entorno de simulación permite la evaluación de la arquitectura, brindando estimaciones de indicadores de calidad relacionados a atributos del software (Figura 8, parte de abajo).

RESULTADOS Y DISCUSIÓN

La especificación de una arquitectura de software mediante el empleo de UCM, constituye una vista de la misma, la cual está compuesta de entidades de software que tienen presencia en tiempo de ejecución y pueden ser simuladas. La herramienta propuesta en este trabajo permite la construcción automática del entorno de simulación DEVS desde la especificación de la arquitectura en UCM.

A partir del sistema de ejemplo estudiado (LMSystem), se han podido realizar pruebas para la evaluación de atributos de calidad del mismo (Figura 8). Para realizar tal evaluación, junto al modelo de la arquitectura en UCM se especificaron los siguientes parámetros: tiempo de ejecución y tiempo de recuperación en cada responsabilidad, tiempo medio entre fallas y tiempo de caída en cada generador de fallas, tiempo medio entre solicitudes y tiempo de simulación en el marco experimental.

La automatización de la generación de los modelos DEVS contribuye a facilitar el empleo del mismo en el contexto del diseño arquitectónico, ya que evita que el arquitecto deba comprender los detalles técnicos.

El desarrollo de la herramienta de software UCM2DEVS se encuentra basado en las propuestas (Bogado, 2013; Bogado et al., 2014; Bogado et al., 2011; Bogado et al., 2013).

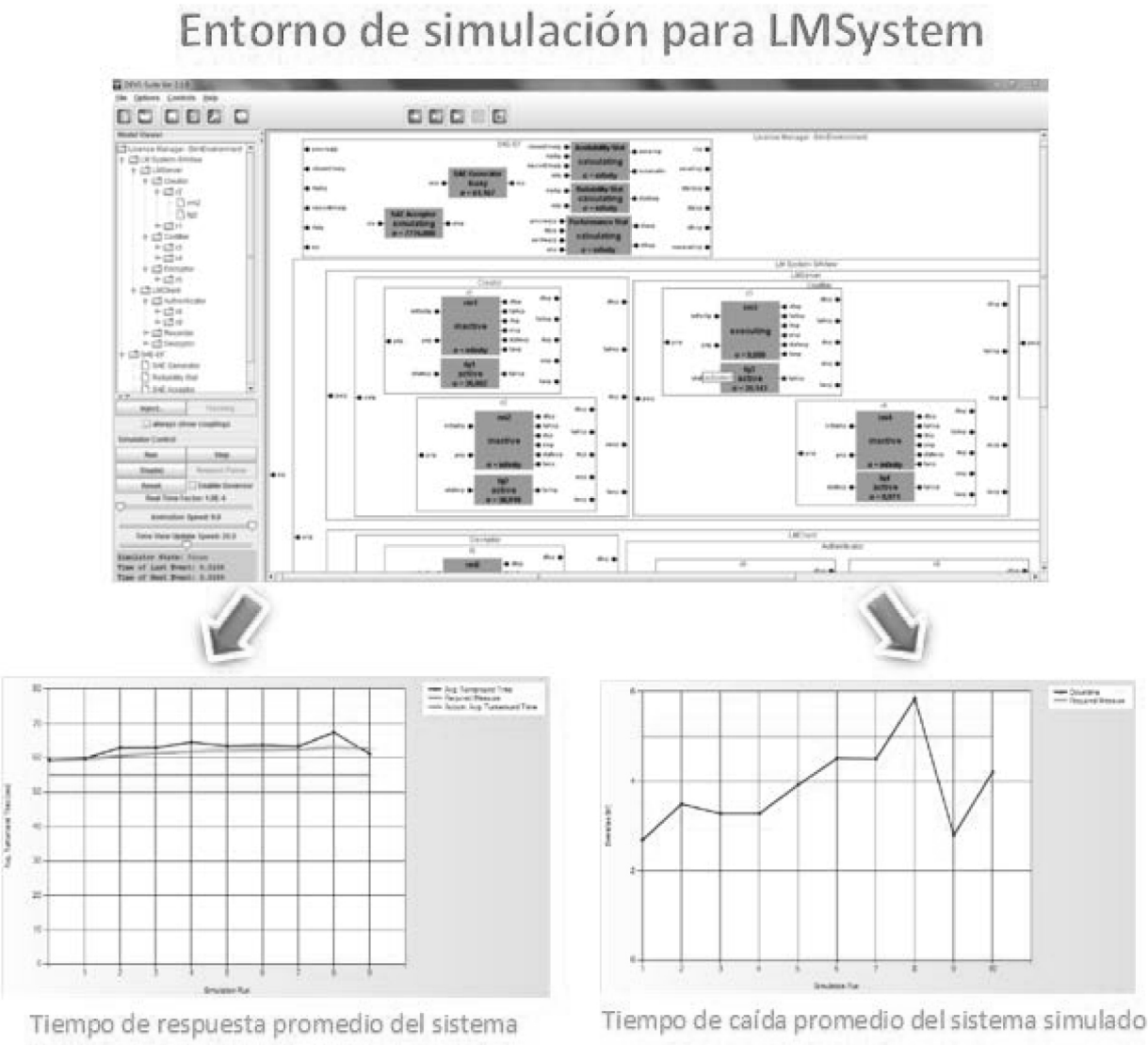


Figura 8. Entorno de simulación de modelos DEVS, más los resultados de la simulación.

UCM2DEVS facilita el proceso de evaluación y evita la introducción de errores que existiría si se debería construir el entorno DEVS manualmente.

En la actualidad, la implementación de los elementos pertenecientes a estas propuestas es la que puede observarse en la Figura 3. La misma es todavía parcial y se encuentra en un proceso de elaboración, faltando todavía incorporar (en forma total o en parte) más funcionalidades, como ser el uso e implementación de los constructores que se utilizan en los caminos de UCM, or-fork, and-fork, or-join, and-join, los cuales pertenecen a PathElement (Figura 3).

Otra de las funcionalidades faltantes es el hecho de poder contar con más de un camino en el diagrama, de

manera que estos puedan ejecutarse en paralelo. Obteniendo de esta forma, una mayor capacidad de representación, para una mejor simulación de arquitecturas de software más complejas.

CONCLUSIONES

La propuesta de este trabajo se basa en una herramienta de software, que a partir de la especificación de la arquitectura de un sistema mediante el empleo de UCM, permita generar automáticamente un modelo DEVS de la misma.

El entorno de simulación DEVS creado permite obtener métricas, las cuales fueron incorporadas como metadatos a la arquitectura del sistema en UCM. Dichas métricas son utilizadas para la evaluación de múltiples indicadores de calidad del sistema de software, como ser su performance o la disponibilidad del mismo dentro de la simulación.

Estos indicadores son cruciales en el desarrollo actual de los sistemas de software, y su análisis en etapas tempranas resulta muy prometedor a la hora del diseño del mismo, dado que el diseñador puede adquirir dichos resultados estadísticos como base en la toma de decisiones. Obteniendo así una idea más precisa, sobre qué tipo de arquitectura de software implementar para cumplir con los requerimientos deseados. Reduciendo de esta forma, el costo de oportunidad y el tiempo de entrega; como también mejorando la calidad final del producto.

REFERENCIAS

Bass, L., Clements, P., Kazman, R. (2012). Software Architecture in Practice, 2nd edition. Addison-Wesley Professional.

Bogado, V. (2013). Un Modelo de Soporte al Análisis de Arquitecturas de Software Mediante Simulación de Eventos Discretos. Tesis Doctoral. Universidad Tecnológica Nacional.

Bogado, V., Gonnet, S., Leone, H. (2014). Modeling and simulation of software architecture in discrete event system specification for quality evaluation. Simulation, 90:3, 290-319.

Bogado, V.; Gonnet, S.; Leone, H. (2011). A Discrete Event Simulation Model for the Analysis of Software Quality Attributes, CLEI Electronic Journal 14:3, Paper 3, 2011.

Bogado, V., Lazzaroni, E., Leone, H., Gonnet, S. (2013). Generación Automática de Modelos DEVS a partir de UCM en el Contexto de la Evaluación de Arquitecturas de Software. Congreso Nacional de Ingeniería Informática/Sistemas de Información (CoNaIISI 2013).

Zeigler, B., Praehofer, H., Kim, T. (2000). Theory of Modeling and Simulation. Integrating Discrete Event and Continuous Complex Dynamic Systems, 2nd edition. Academic Press.

Buhr, R. (1998). Use Case Maps as Architectural Entities for Complex Systems. IEEE Transactions on Software Engineering, 24:12, 1131–1155.

Hudson, S., Flannery, F., Ananian, C.S. (1999). CUP Parser Generator for Java. Web: http://www.cs.princeton.edu/~appel/modern/java/CUP/

Berk, E., Ananian, C.S. (2003). JLex: A Lexical Analyzer Generator for Java. Web: http://www.cs.princeton.edu/~appel/modern/java/JLex/

DEVSSuite (2011). Arizona Center for Integrative Modeling and Simulation- DEVS-Suite. Web: http://acims.asu.edu/software/devs-suite

jUCMNav (2009). jUCMNav-Eclipse Plugin for the User Requiriments Notation. Web: http://marketplace.eclipse.org/content/jucmnav#.UgvqJ6z-VrU

FECHA DE PRESENTACIÓN: 22/02/16
FECHA DE APROBACIÓN: 10/08/16