Modeling User Temporal Behaviors Using Hybrid Simulation Models

M. Blas, S. Gonnet and H. Leone

***Abstract*— This paper presents a simulation model structure that allows build different types of users behavior using workload models. To develop this simulation structure a combination between continuous and discrete models is detailed. In order to formalize the composition of the discrete models defined, DEVS formalism is used. Five different types of user temporal behaviors are specified as a result of the structure proposed: static user, periodic user, once in a lifetime user, unpredictable user and continuously changing user. Then, our proposal gives a context that it could be used in any simulation process that requires a set of requests produced by users as input.**

***Keywords*— Temporal Behavior, User Behavior, Simulation Model, Discrete Event Simulation, Continuous Signals.**

I. INTRODUCCIÓN

HOY en día la utilización de herramientas de software para el modelado y simulación de sistemas es de gran utilidad. Estas herramientas son aplicadas durante el proceso de ingeniería de software, mientras se están especificando las necesidades de hardware, software, bases de datos y/o de personas; posibilitando al ingeniero examinar el rendimiento de diferentes especificaciones del sistema [1]. Es decir, el uso de herramientas de simulación posibilita la evaluación de múltiples hipótesis previo al desarrollo del producto final [2].

En este contexto, la ingeniería de software ha logrado un notable avance en el área. Los avances de los últimos años en la simulación de productos específicos [3]-[6], como así también en la simulación del propio proceso de desarrollo [7]-[10], son evidentes. Sin embargo, en muchos de estos casos es necesario desarrollar, no sólo el modelo de simulación del sistema proyectado, sino también el mecanismo específico que brinde soporte a las entradas requeridas en dicho modelo. Tales mecanismos suelen reflejar el comportamiento de los usuarios.

Los usuarios de cualquier tipo de sistema de software experimentan, de acuerdo al tipo de carga de trabajo, un comportamiento temporal (es decir, un comportamiento que varía con el tiempo). En base a este comportamiento, el usuario genera el conjunto de solicitudes que ingresan al sistema (a fin de ser procesadas) en un período de tiempo específico. En este contexto, la ausencia de mecanismos de soporte que reflejen diferentes comportamientos temporales es evidente. Por este motivo, en este trabajo se propone el diseño e implementación de un mecanismo de entrada genérico aplicable a la generación temporal de solicitudes en base a diferentes tipos de carga de trabajo. El mecanismo propuesto se basa en un modelo de simulación híbrido desarrollado combinando señales continuas con eventos discretos, el cual da lugar a la definición de cinco tipos de usuarios, entre los que se destacan: usuario estático, periódico e impredecible.

El resto del trabajo se encuentra estructurado de la siguiente manera. La sección II introduce los fundamentos y conceptos preliminares de la propuesta, detallando el esquema de simulación genérico diseñado para dar soporte a los modelos específicos. La sección III resume las principales características del formalismo de simulación de eventos discretos utilizado para conceptualizar el conjunto de modelos que componen dicho esquema. La formalización de cada uno de estos modelos se detalla en la sección IV. A fin de evaluar los resultados obtenidos, en la sección V se presentan las pruebas realizadas. Finalmente la sección VI sintetiza las conclusiones y trabajos futuros relacionados.

II. MOTIVACIÓN Y PROPUESTA

El término carga de trabajo (*workload*, en inglés) refiere al nivel de utilización de un recurso de tecnología de información [11]. En cualquier sistema de software, la carga de trabajo (CT) puede ser vista como una consecuencia del acceso de los usuarios al sistema [12]-[14]. En este contexto, la CT toma distintas formas de acuerdo al tipo de recurso sobre el cual esté siendo analizada, a saber: los servidores experimentan cargas de procesamiento, los medios de almacenamiento experimentan un aumento o disminución de la cantidad de datos a almacenar y/o del volumen de consultas a realizar, los recursos de comunicación experimentan variaciones de tráfico, entre otros.

En términos abstractos, la CT puede ser vista como la petición a un recurso o servicio de la ejecución de una tarea predefinida, la cual se repite y varía en función de un patrón temporal. Bajo esta perspectiva, la CT es interpretada como el patrón de utilización del recurso o servicio a lo largo del tiempo, el cual queda definido por la cantidad de peticiones realizadas en dicho período. De esta manera, las solicitudes que realizan los usuarios al recurso /servicio (que se traducen en cargas de procesamiento, tráfico de comunicación y/o volumen de información) son el resultado de un patrón de CT.

La utilización de estrategias de simulación en el área de ingeniería de software constituye uno de los mecanismos de análisis más difundidos. En los últimos años, numerosos trabajos de investigación han utilizado este tipo de técnicas como soporte en diferentes ámbitos, entre los que se destacan cloud computing [15]-[17], análisis de productos de software [5],[6], evaluación de la calidad [18].

M. Blas, Instituto de Desarrollo y Diseño INGAR, CONICET-UTN, Santa Fe, Argentina, mariajuliablas@santafe-conicet.gov.ar

S. Gonnet, Instituto de Desarrollo y Diseño INGAR, CONICET-UTN, Santa Fe, Argentina, sgonnet@santafe-conicet.gov.ar

H. Leone, Instituto de Desarrollo y Diseño INGAR, CONICET-UTN, Santa Fe, Argentina, hleone@santafe-conicet.gov.ar

Corresponding author: María Blas

Todos estos enfoques tienen en común la necesidad de modelar, y posteriormente simular, la generación de entradas al sistema. Es decir, para demostrar la viabilidad del modelo de simulación desarrollado, es necesario que los autores detallen un mecanismo que suministre el conjunto de entradas requeridas. Tales entradas corresponden (en un contexto real) a la forma en la cual distintos tipos de usuarios envían solicitudes al sistema de software bajo análisis. Mientras que algunos de los trabajos plantean un proceso de generación de solicitudes parametrizable, otros integran la especificación de tales mecanismos como anexo del modelo final. Sin embargo, ninguno de los casos considera patrones de comportamiento de usuarios reales.

En este contexto, la definición de un modelo que posibilite la generación de solicitudes de usuario aplicables a distintos entornos pero que, al mismo tiempo, respete patrones de comportamientos reales, constituye un problema de interés para la comunidad. Teniendo en cuenta que los patrones de CT pueden utilizarse como marco de referencia para representar diferentes tipos de comportamientos de usuario, en este trabajo se propone un modelo de simulación híbrido que posibilita la especificación de diferentes tipos de comportamientos temporales en base al uso de señales continuas. La Fig. 1 visualiza la estrategia propuesta.

En la parte superior de la figura se visualiza el conjunto de tareas genéricas que se llevan a cabo entre un usuario *U* y un sistema de software *SW*, mientras que en la parte inferior se presenta la estrategia usada para conceptualizar un modelo que permita simular el envío de solicitudes. Como puede observarse, el usuario *U* envía conjuntos de solicitudes CS_N a un sistema *SW* a lo largo del tiempo a fin de realizar diferentes ejecuciones. La conformación de cada conjunto dependerá del tipo de petición a realizar, pudiendo quedar integrado por solicitudes del mismo o de diferente tipo. Cada conjunto *CS* es identificado por el subíndice *N* que refiere al orden de emisión del conjunto. Si se realiza un análisis en el tiempo del proceso de envío de solicitudes del usuario, se obtendrá un patrón de comportamiento temporal. Dicho patrón corresponde a la CT que el usuario *U* ejerce sobre el sistema *SW*. Pensando esta carga como una función continua, es posible realizar un muestreo (con un criterio previamente establecido para la frecuencia de los intervalos) a fin de obtener un conjunto de valores discretos que la representen. El valor de cada una de las *N* muestras indica la cantidad de solicitudes que forman el conjunto. Para cada muestra obtenida M_N, resta definir el tipo de solicitudes que integran el conjunto.

De esta manera, es posible simular el comportamiento de distintos tipos de usuarios por medio de la utilización de técnicas de muestreo sobre funciones continuas. En base a esta conceptualización, se propone utilizar modelos de simulación continuos en combinación con modelos de eventos discretos con el fin de definir un único modelo de simulación aplicable a distintos tipos de usuarios. La Fig. 2 esquematiza esta propuesta. El *modelo de usuario* resultante consta de dos componentes de alto nivel interconectados: *workload* (modelo continuo) y *request generator* (modelo de eventos discretos). Mientras que el primero da soporte al patrón de CT, el segundo realiza el proceso de muestreo y, posteriormente, la definición del tipo de solicitudes que forman parte de un conjunto dado. Para llevar a cabo esta tarea, el modelo denominado *request generator* utiliza cuatro modelos internos que realizan tareas específicas, a saber: *sampler*, *generator*, *queue* y *creator*. La especificación de cada uno de estos modelos (así como también la del propio *request generator*) se realizó haciendo uso del formalismo de simulación de eventos discretos DEVS [19].

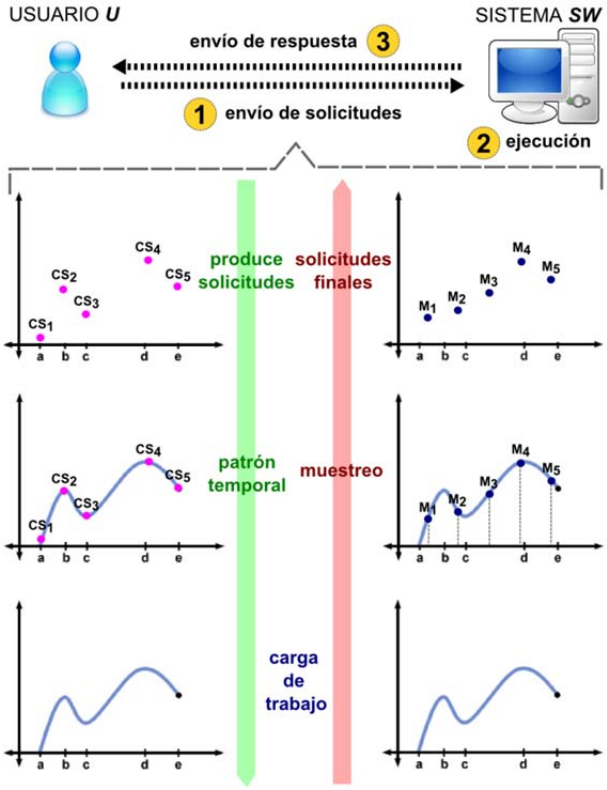


Figura 1. Estrategia utilizada para la conceptualización del esquema base aplicable a los modelos de simulación específicos.

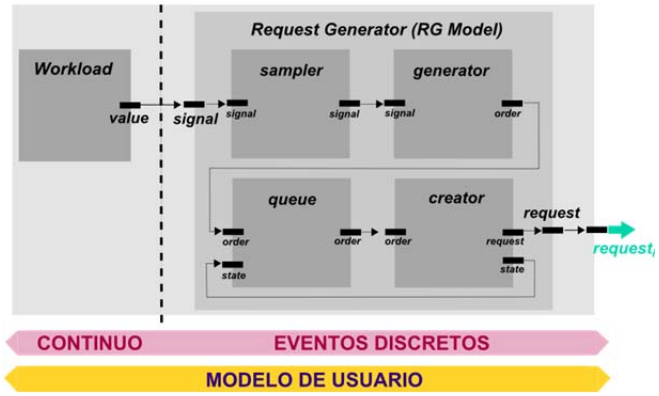


Figura 2. Esquema base genérico del modelo de simulación propuesto.

III. FORMALISMO DE SIMULACIÓN DEVS

DEVS es un formalismo de simulación que establece las bases requeridas para simular sistemas de eventos discretos dentro de un entorno virtual [20]. Permite describir el comportamiento de un sistema de acuerdo a dos niveles de abstracción: *atómico* y *acoplado*. A nivel *atómico* el comportamiento autónomo del sistema es detallado como una

secuencia de transacciones deterministas entre diferentes estados secuenciales. Esta secuencia indica cómo reacciona el sistema ante la ocurrencia de eventos externos y la forma en la cual se generan los eventos de salida. Por su parte, en el nivel *acoplado* el sistema es representado como una red de componentes interconectados en la que cada elemento puede corresponder a un modelo atómico o acoplado.

A. DEVS Clásico con Puertos

La ecuación (1) formaliza la definición de un modelo DEVS clásico con puertos [19].

$$DEVS = (X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta) \quad (1)$$

donde:

$X = \{(p,v) \mid p \in InPorts, v \in X_p\}$ es el conjunto de entradas en el cual *InPorts* es el conjunto de puertos de entrada y X_p es el conjunto de posibles valores para el puerto p.

$Y = \{(p,v) \mid p \in OutPorts, v \in Y_p\}$ es el conjunto de salidas en el cual *OutPorts* es el conjunto de puertos de salida y Y_p es el conjunto de posibles valores de salida para el puerto p.

S es el conjunto de estados.

$\delta_{ext}: Q x X \rightarrow S$ es la función de transición externa en la cual $Q = \{(s,e) \mid s \in S, 0 \leq e \leq ta(s)\}$ es el conjunto de estados totales donde e es el tiempo transcurrido desde la última transición en el estado s.

$\delta_{int}: S \rightarrow S$ es la función de transición interna.

$\lambda: S \rightarrow Y$ es la función de salida.

$ta: S \rightarrow R_0^+$ es la función de avance de tiempo.

B. DEVS Acoplados

La ecuación (2) formaliza la definición de un modelo DEVS acoplado.

$$N = (X, Y, D, \{M_d \mid d \in D\}, EIC, EOC, IC, Select) \quad (2)$$

donde:

X e Y se definen de la misma forma que en la ecuación (1).

D es el conjunto de referencias a componentes.

M_d es el conjunto de modelos que componen al acoplado.

$EIC \in \{((N,ip_N),(d,ip_d)) \mid ip_N \in InPorts, d \in D, ip_d \in InPorts_d\}$ son los acoplamientos externos de entrada.

$EOC \in \{((d,op_d),(N,op_N)) \mid op_N \in OutPorts, d \in D, op_d \in OutPorts_d\}$ son los acoplamientos externos de salida.

$IC \in \{((a, op_a),(b, ip_b)) \mid a,b \in D , op_a \in OutPorts_a, ip_b \in InPorts_b\}$ es el conjunto de acoplamientos internos.

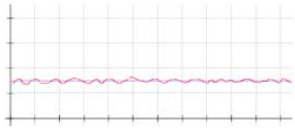
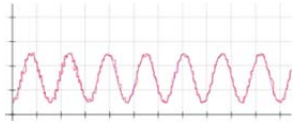
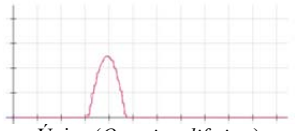
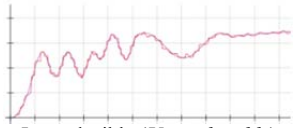
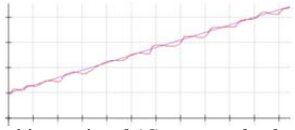
Select es la función desempate (sólo para DEVS clásico).

IV. MODELOS DE SIMULACIÓN

A. Modelos de Cargas de Trabajo: Workload

De acuerdo con Fehling et al. [11], existen cinco tipos de CT que pueden ser utilizados para representar comportamientos temporales de usuario (Tabla I). Debido a que cada una de estas representaciones evidencia una conducta real, existen ligeras variaciones (propias del comportamiento humano) sobre el patrón temporal predominante.

TABLA I
CARGAS DE TRABAJO APLICABLES A USUARIOS

Tipo de CT	***Descripción***
 Estática (*Static*)	Perfil de utilización más o menos homogéneo a lo largo del período de tiempo en estudio.
 Periódica (*Periodic*)	Perfil de utilización recurrente en horas pico a intervalos de tiempo regulares.
 Única (*Once in a lifetime*)	Caso especial de CT periódica donde el pico de utilización se da una única vez en el tiempo.
 Impredecible (*Unpredictable*)	Generalización de CT periódica en la que se dan picos/valles de forma no predecible.
 De cambio continuo[a] (*Continuously changing*)	El perfil de utilización aumenta o disminuye constantemente en el período de tiempo.

[a] La figura corresponde a una CT continuamente creciente.

A fin de generar modelos de simulación que permitan reproducir los diferentes tipos de CT esquematizados y, teniendo en cuenta que es común utilizar abstracciones para simplificar la construcción de este tipo de modelos [21] , se optó por idealizar cada una de las representaciones en una función continua. Bajo esta idealización, cada tipo de CT se simplificó en una única función *f(t)* definida en base a un conjunto de parámetros específicos. Estos parámetros permiten adaptar las funciones para reflejar, en base al mismo tipo de CT, diferentes resultados (esto es, distintos valores de salida). De esta manera, en cada caso concreto sobre el que se aplique un modelo de CT se deberá configurar la *f(t)* haciendo uso de valores de parametrización que reflejen el comportamiento deseado.

Las funciones definidas para representar las CT propuestas son las siguientes:

(i) *CT estática*: Función constante $f(t)=c$ cuyo valor de salida c se mantiene para cualquier valor de la variable independiente t. Requiere la configuración del parámetro entero c (con $c > 0$), el cual representa la cantidad de solicitudes, es decir, el valor de salida de la función.

(ii) *CT periódica*: Función periódica y positiva *f(t)* que representa un comportamiento sinusoidal. Requiere la configuración de los parámetros *amplitude* y *frequency*. El parámetro entero *amplitude* refiere al máximo alejamiento (en el valor absoluto) de la curva respecto a

su línea base. El parámetro real *frequency* indica la frecuencia de oscilación (número de repeticiones por unidad de tiempo), siendo su valor el inverso al período de la función.

(iii) *CT única*: Función *f(t)* definida en tramos -conforme lo expresado en la ecuación (3)-, en la cual existe un instante de tiempo *tMax* en el que la curva toma el máximo valor de utilización *vMax*. Requiere de la configuración de los parámetros reales *tMax* y *vMax*. Por otra parte, los valores de $sqrt_1$ y $sqrt_2$ quedan determinados por las raíces reales de la función cuadrática definida en base a los valores especificados para cada uno de los parámetros previamente identificados.

(iv) *CT impredecible*: Función continua *f(t)* que define un comportamiento aleatorio (acotado en el extremo superior) a lo largo del tiempo. Requiere la configuración del parámetro *vMax* que indica el máximo valor de salida admitido. De esta manera, los valores resultantes de la función oscilarán en el rango [0; *vMax*] (restringiendo la posibilidad de una CT infinita).

(v) *CT de cambio continuo*: Función *f(t)* que evidencia una evolución temporal de incremento/decremento lineal a partir de un valor de referencia inicial. Requiere la configuración de los parámetros *vInit* y *gradient*. El parámetro *vInit* refiere al valor de inicio de la función, mientras que el parámetro *gradient* representa la pendiente de la recta que esquematiza la evolución temporal.

$$f(t)=\begin{cases} -\left((t-tMax)^2\right)+vMax & si\ sqrt_1 \le t \le sqrt_2 \\ 0 & en\ otro\ caso \end{cases} \quad (3)$$

$$f(t)=\begin{cases} gradient \times t + vInit & si\ f(t) > 0 \\ 0 & en\ otro\ caso \end{cases} \quad (4)$$

Para cada una de estas funciones, se desarrolló un modelo de simulación del tipo $y = f(t)$ en el cual el valor de salida y_i en un instante de tiempo t_i es el valor de la función *f(t)* especificada cuando $t=t_i$.

B. Modelo de Generación de Solicitudes: RG Model

La ecuación (5) formaliza la definición del modelo DEVS acoplado que representa el proceso de generación de solicitudes esquematizado en la Fig. 2. Como puede observarse este modelo se encuentra formado por cuatro modelos atómicos, a saber: *Sampler*, *Order Generator*, *Order Queue* y *Request Creator*. La definición de cada uno de estos modelos se detalla en las siguientes secciones.

$$RG=(X_{RG},Y_{RG},D_{RG},\{M_{RG,d}|d\in D_{RG}\},EIC_{RG},EOC_{RG},IC_{RG}) \quad (5)$$

donde:

$X_{RG} = \{(p,v) \mid p \in RGIP,\ v \in X_{RG,p}\}$ con $RGIP=\{ signal \}$ y $X_{RG,signal} = R_0^+$

$Y_{RG}=\{(p,v) \mid p \in RGOP,\ v \in Y_{RG,p}\}$ con $RGOP=\{ request \}$ y $X_{RG,request} = \{ request_1, request_2,..., request_n \}$

$D_{RG} = \{sampler, generator, queue, creator\}$

$$M_{RG,d} = \begin{cases} M_{RG,sampler} = S \\ M_{RG,generator} = OG \\ M_{RG,queue} = OQ \\ M_{RG,creator} = RC \end{cases}$$

$EIC_{RG} = \{ ((RG,signal),(sampler,signal)) \}$

$EOC_{RG} = \{ ((creator,request),(RG,request)) \}$

$IC_{RG} = \{((sampler,signal),(generator,signal)) , ((generator, order),(queue,order)) , ((queue,order),(creator,order)), ((creator,request),(queue,state)) \}$

1) Sampler

El modelo *Sampler* (S) se encuentra formado por un puerto de entrada y un puerto de salida. Mientras que por el puerto de entrada recibe eventos que corresponden a valores numéricos que representan una señal continua, por el puerto de salida emite la última entrada recibida redondeada al valor entero más cercano a intervalos variables de tiempo. Al recibir un evento, el S guarda el valor como parte de su estado interno dentro de la variable *value*. Cada *T* unidades de tiempo, emite un evento de salida que contiene el valor entero más próximo asociado a la variable *value*. El valor de *T* es obtenido por medio de una función de distribución uniforme cuyo rango de valores está comprendido entre $[T_{min};T_{max}]$ (donde T_{min} y T_{max} son parámetros configurables de S).

La ecuación (6) formaliza la definición del modelo S, donde *round()* refiere a la función matemática que aproxima un valor real al valor entero más próximo.

$$S = (X_S, Y_S, S_S, \delta_{S,ext}, \delta_{S,int}, \lambda_S, ta_S) \quad (6)$$

donde:

$X_S = \{(p,v) \mid p \in SIP,\ v \in X_{S,p}\}$ con $SIP = \{ signal \}$ y $X_{S,signal} = R_0^+$

$Y_S = \{(p,v) \mid p \in SOP,\ v \in Y_{S,p}\}$ con $SOP = \{ signal \}$ y $Y_{S,signal} = N_0$

$S_S = (\sigma \times value)$ donde $\sigma \in R_0^+$ y $value \in R_0^+$

$\delta_{S,ext}(s, e,(x, port)) = \delta_{S,ext}((\sigma, value),e,(input, signal)) = = (\sigma-e, input)$

$\delta_{S,int}(s) = \delta_{S,int}(0, value) = (T, value)$

$\lambda_{S,int}(s) = \lambda_{S,int}(0, value) = (signal, round(value))$

$ta_S = \sigma$

2) Order Generator

El modelo *Order Generator* (OG) al recibir un evento de entrada emite tantas salidas como el valor entero indicado en dicho evento. Cada evento de salida lleva en su interior el identificador del tipo de solicitud a generar. Así, mientras que el valor de entrada representa la cantidad de solicitudes que deben ser creadas, los eventos de salida especifican (para cada una de las solicitudes a crear) el *requestType* asociado. De esta manera, se asume que las solicitudes que ingresarán al sistema de software son identificadas por medio de un valor entero (secuencial que inicia en 1) denominado *requestType*.

El proceso de selección de este identificador es configurado en los parámetros *randomRequestTypeId*, *fixedRequestTypeId*

y *requestTypeQuantity* del modelo. El primer parámetro indica si el proceso de selección debe (o no) generar solicitudes al azar. En caso afirmativo, para cada evento de salida, el OG genera un identificador aleatorio comprendido entre 1 y *requestTypeQuantity* (esto es, la cantidad de tipos de solicitudes que existen en el sistema). En caso negativo, el modelo utiliza el parámetro *fixedRequestTypeId* como único identificador de los eventos de salida.

La ecuación (7) formaliza la definición del modelo OG, donde *getRequestType()* es la función que ejecuta el proceso de selección del identificador de solicitud a ser enviado en el evento de salida. La configuración de esta función queda definida por los parámetros detallados con anterioridad.

$$OG = (X_{OG}, Y_{OG}, S_{OG}, \delta_{OG,ext}, \delta_{OG,int}, \lambda_{OG}, ta_{OG}) \quad (7)$$

donde:

$X_{OG} = \{(p,v) \mid p \in OGIP, v \in X_{OG,p}\}$ con $OGIP = \{ signal \}$ y $X_{OG,signal} = N_0$

$Y_{OG} = \{(p,v) \mid p \in OGOP, v \in Y_{OG,p}\}$ con $OGOP = \{ order \}$ y $Y_{OG,order} = N$

$S_{OG} = (\sigma \times value)$ donde $\sigma \in R_0^+$ y $value \in N_0$

$\delta_{OG,ext}(s, e, (x, port)) = \delta_{OG,ext}((\sigma, value), e, (input, signal)) = (0, input)$

$$\delta_{OG,int}(s) = \begin{cases} \delta_{OG,int}(0, value) = (0, value-1) & \text{si } value > 1 \\ \delta_{OG,int}(0, value) = (\infty, value-1) & \text{si } value = 1 \end{cases}$$

$\lambda_{OG,int}(s) = \lambda_{OG,int}(0, value) = (order, requestType)$ donde $requestType = getRequestType()$

$ta_{OG} = \sigma$

3) Order Queue

El modelo *Order Queue* (OQ) simula el comportamiento de una cola de trabajos que administra la entrega de los identificadores de solicitudes. Consta de dos puertos de entrada (*order* y *state*) y un puerto de salida (*order*). Por el puerto de entrada *state* recibe el estado del procesador asociado a la cola (esto es, una señal que indica si el modelo está o no en condiciones de recibir un nuevo evento para ser procesado). Por otra parte, por el puerto de entrada *order* recibe valores enteros que corresponden a los identificadores de las solicitudes a crear. Cada vez que arriba un evento de entrada, el identificador asociado es encolado como último elemento de una estructura de datos. Dicha estructura respeta el formato FIFO (first in, first out). Sólo cuando el procesador indica que se encuentra disponible, el primer identificador de la estructura es enviado dentro de un nuevo evento por el puerto de salida *order*. Debido a que el envío de este evento implica su procesamiento, el OQ cambia de estado registrando que el procesador ha dejado de estar disponible en consecuencia del evento enviado. Esto se debe a que, si el procesador estaba disponible (motivo por el cual se envió un identificador en el evento de salida), al recibir la orden enviada deberá inmediatamente comenzar su tratamiento (por lo que el procesador pasará a estar ocupado).

La ecuación (8) formaliza la definición del OQ, donde el operador • es utilizado para expresar la concatenación entre un elemento y una estructura de datos (y viceversa).

$$OQ = (X_{OQ}, Y_{OQ}, S_{OQ}, \delta_{OQ,ext}, \delta_{OQ,int}, \lambda_{OQ}, ta_{OQ}) \quad (8)$$

donde:

$X_{OQ} = \{(p,v) \mid p \in OQIP, v \in X_{OQ,p}\}$ con $OQIP = \{ order , state \}$, $X_{OQ,order} = N$ y $X_{OQ,state} = \{ available \}$

$Y_{OQ} = \{(p,v) \mid p \in OQOP, v \in Y_{OQ,p}\}$ con $OQOP = \{ order \}$ y $Y_{OQ,order} = N$

$S_{OQ} = (\sigma \times q \times proc)$ donde $\sigma \in R_0^+$, *q es una cola de enteros* y $proc \in \{available , unavailable \}$

$$\delta_{OQ,ext}(s,e,(x,port)) = \begin{cases} \delta_{OQ,ext}((\sigma,q,proc),e,(id,order)) = \\ = (\infty, q \bullet id, proc) \\ \delta_{OQ,ext}((\sigma,q,proc),e,(available,state)) = \\ = (0,q,available) \end{cases}$$

$\delta_{OQ,int}(s) = \delta_{OQ,int}(0, id \bullet q, available) = (\infty, q, unavailable)$

$\lambda_{OQ,int}(s) = \lambda_{OQ,int}(0, id \bullet q, available) = (order, id)$

$ta_{OQ} = \sigma$

4) Request Creator

El modelo *Request Creator* (RC) consta de un puerto de entrada y dos puertos de salida. Por el puerto de entrada *order* recibe el identificador de la solicitud a crear, mientras que por el puerto de salida *request* envía un evento que encapsula la solicitud creada en base al identificador recibido. Por otra parte, por el puerto de salida *state* el modelo envía un evento de estado que indica su disponibilidad. De esta manera, el modelo garantiza que en un momento dado sólo se estará procesando un trabajo.

Debido a que el contenido de una solicitud varía de acuerdo al sistema de software sobre el cual debe actuar el modelo de usuario, la conformación de las solicitudes debe especificarse en un archivo externo. La ubicación física de este archivo debe indicarse en el parámetro *requestsFile* del modelo.

La ecuación (9) formaliza la definición del RC, donde la función *createRequest(type)* es la encargada de crear una solicitud del tipo *type* que respete la estructura requerida. Esta estructura se obtiene por medio del acceso al contenido del archivo indicado en el parámetro *requestsFile*.

$$RC = (X_{RC}, Y_{RC}, S_{RC}, \delta_{RC,ext}, \delta_{RC,int}, \lambda_{RC}, ta_{RC}) \quad (9)$$

donde:

$X_{RC} = \{(p,v) \mid p \in RCIP, v \in X_{RC,p}\}$ con $RCIP = \{ order \}$ y $X_{RC,order} = N$

$Y_{RC} = \{ (p,v) \mid p \in RCOP, v \in Y_{RC,p} \}$ con $RCOP = \{ request, state \}$, $Y_{RC,request} = \{ request_1, request_2, ..., request_n \}$ y $Y_{RC,state} = \{ available \}$

$S_{RC} = (\sigma \times req)$ donde $\sigma \in R_0^+$ y $req \in \{ request_1, ..., request_n \}$

$\delta_{RC,ext}(s,e,(x,port)) = \delta_{RC,ext}((\sigma,req),e,(id,order)) = (0,newRequest)$ donde $newRequest = createRequest(id)$

$$\delta_{RC,int}(s) = \begin{cases} \delta_{RC,int}(0, req) = (0, \phi) \\ \delta_{RC,int}(0, \phi) = (\infty, \phi) \end{cases}$$

$$\lambda_{RC,int}(s) = \begin{cases} \lambda_{RC,int}(0, req) = (request , req) \\ \lambda_{RC,int}(0, \phi) = (state , available) \end{cases}$$

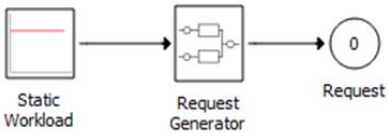
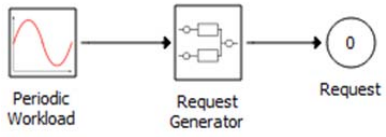
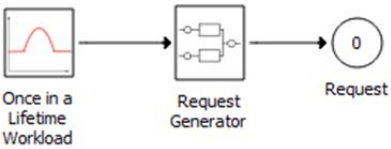
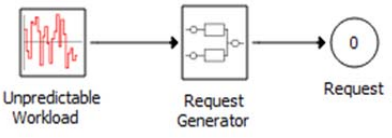
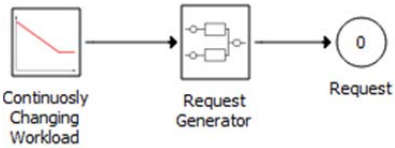
$ta_{RC} = \sigma$

C. Modelos Específicos de Comportamientos de Usuarios

Con el objetivo de representar distintos tipos de usuarios, se diseñaron e implementaron cinco modelos acoplados que respetan el esquema planteado en la Fig. 2, a saber: *static user, periodic user, once in a lifetime user, unpredictable user, continuously changing user.* Cada modelo específico utiliza como componente *Workload* una función diferente, lo que da como resultado un conjunto de comportamientos de usuario que varía de acuerdo al tipo de señal elegida para la generación de la CT.

La implementación de estos modelos se realizó haciendo uso de la herramienta PowerDEVS [22]. PowerDEVS es una herramienta de desarrollo de modelos de simulación en C++ especialmente destinada al diseño y construcción de modelos de simulación DEVS. Sin embargo, también proporciona librerías que posibilitan el desarrollo de trabajos basados en modelos híbridos (similares al esquema que conceptualiza a los modelos de simulación propuestos). La Tabla II resume el conjunto de modelos de usuario implementados en esta herramienta (haciendo uso de una visualización en términos de componentes y conectores, dentro de la cual los puertos se indican explícitamente por medio de círculos numerados).

TABLA II
MODELOS DE COMPORTAMIENTOS DE USUARIOS IMPLEMENTADOS EN POWERDEVS

Nombre	***Implementación***
Static user	
Periodic user	
Once in a lifetime user	
Unpredictable user	
Continuously changing user	

V. PRUEBAS Y RESULTADOS

Cada una de las implementaciones de los modelos de comportamiento de usuario descriptos en la sección precedente se probó a fin de evaluar su adecuación a las CT pretendidas. A continuación se detallan dos de las pruebas realizadas sobre los modelos *Periodic User* y *Continuously Changing User*.

Como se mencionó con anterioridad, cada uno de los modelos de simulación sintetiza un modelo genérico susceptible de ser aplicado en diferentes contextos de acuerdo a la configuración de sus parámetros. Entonces, teniendo en cuenta que las CT resultantes dependerán de los valores asociados a los parámetros y, a fin de probar la adecuación de los modelos sobre diferentes escenarios, se presentan a continuación dos posibles configuraciones para cada caso. Cada una de las configuraciones propuestas se corresponde con un escenario, a saber:

(i) *Periodic User*:
   a. Escenario #1: Un usuario genera de 0 a 5 solicitudes de forma periódica, repitiendo el patrón de ocurrencia aproximadamente cada 15 minutos (900 segundos).
   b. Escenario #2: Un usuario genera de 0 a 20 solicitudes de forma periódica, repitiendo el patrón de ocurrencia aproximadamente cada 1 hora (3600 segundos).

(ii) *Continuously Changing User*:
   a. Escenario #1: Un usuario genera (inicialmente) 15 solicitudes, variando su comportamiento linealmente de forma tal que, luego de transcurrida una hora y media (5400 segundos), no genera más solicitudes.
   b. Escenario #2: Un usuario inactivo comienza a generar solicitudes linealmente de forma tal que, luego de 1 hora (3600 segundos), emite 7 solicitudes.

En base a las descripciones precedentes, se establecieron los valores de los parámetros requeridos para cada modelo (Tabla III). Estos escenarios se ejecutaron haciendo uso de un tiempo de simulación equivalente a 2 horas (7200 segundos – unidades de tiempo-).

Las Fig. 3 y 4 visualizan los resultados obtenidos para cada uno de los modelos configurados en términos de cantidad de solicitudes producidas a lo largo del tiempo. Como puede observarse en la Fig. 3, las salidas de los escenarios propuestos reflejan el comportamiento de una CT periódica visualizándose un comportamiento sinusoidal a lo largo del tiempo. En este caso, ambos escenarios respetan el comportamiento esperado. Por su parte, en la Fig. 4 se observa que la CT del primer escenario sufre un decremento lineal (partiendo del valor inicial) hasta llegar a cero, mientras que, en la CT del segundo escenario se observa un crecimiento en el tiempo. Nuevamente, al igual que en el caso previo, los comportamientos resultantes del proceso de simulación se ajustan a lo esperado. En este sentido, es importante resaltar que en ambos casos el uso de diferentes valores en los parámetros de configuración de cada escenario dio como resultado la adaptación de modelos genéricos a distintas situaciones. De esta manera, es posible concluir que la especificación de los modelos de simulación desarrollados es correcta.

Los modelos restantes se probaron de forma análoga a los casos precedentes, habiéndose obtenido resultados similares a los expuestos con anterioridad en cada una de las experiencias realizadas. Por motivos de espacio, los resultados de estas pruebas no han sido incluidos en este trabajo.

TABLA III
CONFIGURACIONES DE LAS PRUEBAS REALIZADAS EN LOS MODELOS PERIODIC USER Y CONTINUOUSLY CHANGING USER

Periodic User			
Parámetros del Modelo		*Valor*	
Workload	*Request Generator*	*Escenario #1*	*Escenario #2*
amplitude[b]	-	2.5	10
frequency[c]	-	0.0011	0.00027
-	Tmin	1	
-	Tmax	30	
-	requestsFile	requestsexample.txt	
-	requestTypeQuantity	5	
-	randomRequestTypeId	no	
-	fixedRequestTypeId	1	
Continuously Changing User			
Parámetros del Modelo		*Valor*	
Workload	*Request Generator*	*Escenario #1*	*Escenario #2*
vInit	-	15	0
gradient[d]	-	-0.0027	0.0019
-	Tmin	1	
-	Tmax	30	
-	requestsFile	requestsexample.txt	
-	requestTypeQuantity	5	
-	randomRequestTypeId	no	
-	fixedRequestTypeId	1	

[b] Los valores corresponden a la mitad del rango de oscilación definido en cada escenario.
[c] Los valores corresponden a *1/T*, donde *T* representa el período de la señal en segundos. Para los escenarios propuestos se tiene que: *T=900* (Escenario#1) y *T=3600* (Escenario #2).
[d] Los valores corresponden a *G=valor de referencia/tiempo de transición en segundos*. Para los escenarios propuestos se tiene que: *G=-15/5400* (Escenario #1) y *G=7/3600* (Escenario #2).

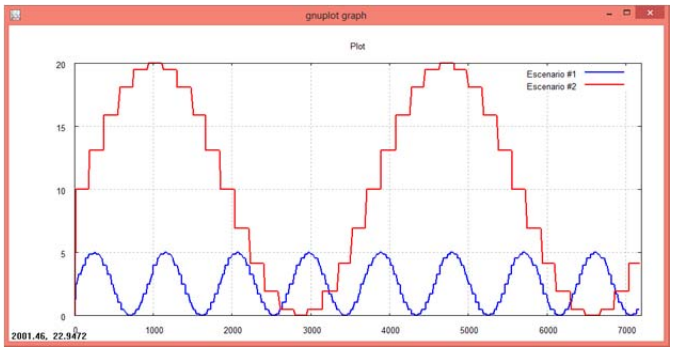


Figura 3. Evolución temporal de la cantidad de solicitudes producidas en el modelo *Periodic User* para cada uno de los escenarios propuestos.

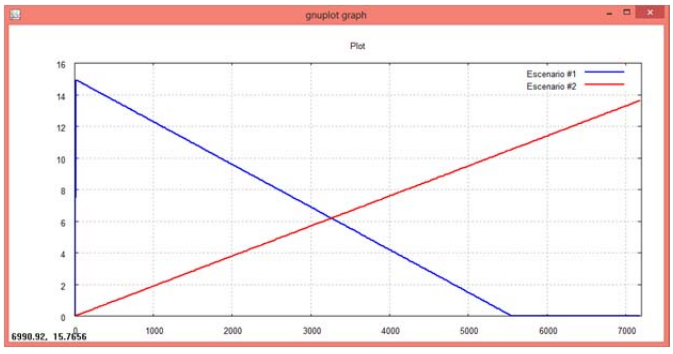


Figura 4. Evolución temporal de la cantidad de solicitudes producidas en el modelo *Continuously Changing User* para cada uno de los escenarios propuestos.

Es importante destacar que la utilización de este tipo de modelos como generadores de entrada para modelos de simulación específicos presupone la existencia de múltiples componentes que procesen las solicitudes creadas y/o de la existencia de una cola de trabajos en espera dentro del sistema. Esto se debe a que, en un momento dado, el modelo de comportamiento de usuario produce múltiples solicitudes de entrada. Si no existe algún mecanismo que gestione su ingreso, sólo la primera solicitud será atendida y el resto se perderá.

VI. CONCLUSIONES

En la actualidad, el uso de técnicas de simulación para estimar y predecir el comportamiento de sistemas de software (o de algún producto y/o parte derivada de estos) se encuentra ampliamente difundido. En la mayoría de los casos estudiados, la aplicación de estas técnicas requiere de la conceptualización y estructuración de modelos de entrada; los cuales suelen corresponderse con modelos de comportamientos de usuarios. De esta manera, se manifiesta una clara necesidad de disponer de mecanismos de entrada apropiados para la generación temporal de solicitudes.

Se ha presentado un esquema de simulación genérico basado en la combinación de modelos continuos y modelos de eventos discretos que permite definir tipos de comportamientos de usuario en base a distintos patrones de carga de trabajo. A partir de este esquema, cinco modelos específicos han sido definidos e implementados; dando como resultado un conjunto configurable de modelos de comportamientos de usuarios. La aplicabilidad de estos modelos en diferentes ámbitos es alta, ya que han sido diseñados de forma tal que su parametrización garantice independencia del contexto. De esta manera, el esquema base no sólo se ha utilizado para detallar modelos específicos sino que además puede aplicarse a otros tipos de cargas de trabajo no contempladas en los modelos desarrollados. Es decir, pueden generarse nuevos modelos de usuarios haciendo uso de otro tipo de señales de entrada como parte del modelo *Workload*. La extensibilidad del esquema propuesto es una de las principales ventajas del enfoque.

Los modelos de usuario implementados en este trabajo han sido utilizados como soporte en la línea de investigación actual del grupo de trabajo, la cual propone evaluar la calidad de arquitecturas web a fin de predecir su adecuación a los requerimientos no funcionales. Específicamente, los modelos de comportamiento de usuario han sido utilizados para evaluar el dinamismo de la arquitectura en estudio para adaptarse a cargas de trabajo variables.

Aunque la propuesta de este trabajo constituye una parte del desarrollo actual, la posibilidad de aplicarla en otros ámbitos es el motor que impulsa su difusión en el contexto de la ingeniería de software.

AGRADECIMENTOS

Los autores agradecen el apoyo financiero otorgado por CONICET (PIP 112-20110100906) y por la Universidad Tecnológica Nacional – Facultad Regional Santa Fe (UTI3803TC and UTI3804TC).

REFERENCIAS

[1] R. Pressman, "Software Engineering: A Practitioner´s Approach", 7th ed. McGraw-Hill. 2010.

[2] R. Buyya, R. Ranjan, R. N. Calheiros, "Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities", *International Conference on High Performance Computing & Simulation*, pp. 1-11, HPCS'09. 2009.

[3] A. Law, M. McComas, "Simulation software for communications networks: the state of the art". *IEEE Communications Magazine*, vol. 32, no 3, p. 44-50. 1994.

[4] F. J. Lerch, D. J. Ballou, D. E. Harter, "Using simulation-based experiments for software requirements engineering". *Annals of Software Engineering*, vol. 3, no 1, p. 345-366. 1997.

[5] V. Bogado, S. Gonnet, H. Leone, "Modeling and simulation of software architecture in discrete event system specification for quality evaluation", *Simulation*, vol. 90, no. 3, p. 290. 2014.

[6] J. Rech, C. Bunse, "Evaluating Performance of Software Architecture Models with the Palladio Component Model". *Model-Driven Software Development: Integrating Quality Assurance,* pp. 95-118. 2008.

[7] A. Drappa, J. Ludewig, "Simulation in Software Engineering Training", *Proceedings of the 22Nd International Conference on Software Engineering*, New York, pp. 199–208. 2000.

[8] D. Pfahl, K. Lebsanft, "Using simulation to analyse the impact of software requirement volatility on project performance", *Information and Software Technology*, vol. 42, no 14, pp. 1001-1008. 2000.

[9] I. Rus, J. Collofello, P. Lakey, "Software process simulation for reliability management". *Journal of Systems and Software*, vol. 46, no 2, pp. 173-182. 1999.

[10] D. Raffo, J. Vandeville, R. Martin, "Software process simulation to achieve higher CMM levels", *Journal of Systems and Software*, vol. 46, no 2, pp. 163-172. 1999.

[11] C. Fehling, F. Leymann, R. Retter, W. Schupeck, P. Arbitter, "Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications". *Springer Science & Business Media*, 2014.

[12] H. Hlavacs, E. Hotop, G. Kotsis, "Workload generation by modeling user behavior", Proceedings *of OPNETWORKS 2000*, 2000.

[13] C. Kurz, H. Hlavacs, G. Kotsis, "Workload Generation by Modelling User Behavior in an ISP Subnet". *Proceedings of the International Symposium on Telecommunications*, 2001.

[14] H. Hlavacs, G. Kotsis, "Modeling user behavior: a layered approach", *7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pp. 218–225. 1999.

[15] D. Zehe, W. Cai, A. Knoll, H. Aydt, "Tutorial on a Modeling and Simulation Cloud Service", *Proceedings of the 2015 Winter Simulation Conference*, Piscataway, NJ, USA, pp. 103–114. 2015.

[16] R. Calheiros, R. Ranjan, A. Beloglazov, C. De Rose, R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms", *Software: Practice and Experience*, vol. 41, no 1, p. 23-50. 2011.

[17] A. Nuñez, J. Vázquez-Poletti, A. C. Caminero, J. Carretero, I. Llorente, "Design of a New Cloud Computing Simulation Platform". *Computational Science and Its Applications - ICCSA 2011*, Eds. Springer, pp. 582–593. 2011.

[18] N. Hurtado, M. Ruiz, E. Orta, J. Torres, "Using simulation to aid decision making in managing the usability evaluation process", *Information and Software Technology*, vol. 57, p. 509-526. 2015.

[19] B. Zeigler, H. Praehofer, T. Kim, "Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems", *Academic Press,* 2nd edition. 2000.

[20] B. P. Zeigler, H. S. Sarjoughian, R. Duboz, J. C. Soulie, "Guide to Modeling and Simulation of Systems of Systems", *Springer Science & Business Media*, London. 2013.

[21] A. M. Law, W. D. Kelton, "Simulation Modeling and Analysis". *McGraw-Hill*, 1991.

[22] F. Bergero, E. Kofman, "PowerDEVS: a tool for hybrid system modeling and realtime simulation". *Simulation*, vol. 87, no. 1–2, pp. 113–132. Jan, 2011.

María Julia Blas recibió el título de Ingeniera en Sistemas de Información de la Facultad Regional Santa Fe de la Universidad Tecnológica Nacional (Argentina) en el año 2014. Actualmente se desempeña como becaria doctoral del Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET) en el Instituto de Desarrollo y Diseño INGAR ubicado en la ciudad de Santa Fe, Argentina (el cual depende del Consejo Nacional de Investigaciones Científicas y Técnicas y de la Universidad Tecnológica Nacional). Además, se encuentra desarrollando el Doctorado en Ingeniería mención Sistemas de Información en la Universidad Tecnológica Nacional (UTN). Las principales áreas sobre las cuales desarrolla su trabajo de investigación incluyen simulación de arquitecturas de software y evaluación de la calidad.

Silvio Gonnet recibió el título de Ingeniero en Sistemas de Información de la Facultad Regional Santa Fe de la Universidad Tecnológica Nacional (Argentina) en el año 1998 y obtuvo el Doctorado en Ingeniería de la Universidad Nacional del Litoral (UNL) en el año 2003. Actualmente posee el cargo de Investigador Adjunto del Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), desempeñando funciones en el Instituto de Desarrollo y Diseño INGAR (CONICET-UTN). Además, es profesor del Departamento Ingeniería en Sistemas de Información de la Facultad Regional Santa Fe de la Universidad Tecnológica Nacional. Sus áreas de interés son modelos de soporte al proceso de diseño, arquitecturas de software y web semántica.

Horacio Leone recibió el Doctorado en Ingeniería Química de la Universidad Nacional del Litoral en el año 1986. Actualmente se desempeña como profesor en el Departamento Ingeniería en Sistemas de Información de la Facultad Regional Santa Fe de la Universidad Tecnológica Nacional. Además, posee el cargo de Investigador Principal del Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), desempeñando funciones en el Instituto de Desarrollo y Diseño INGAR (CONICET-UTN). Sus temas de interés incluyen modelado empresarial, web semántica para sistemas de información de cadenas de suministro y arquitecturas de software.