risti
Revista Ibérica de Sistemas e Tecnologias de Informação
Revista Ibérica de Sistemas y Tecnologías de Información

Recebido/Submission: 29/07/2020
Aceitação/Acceptance: 17/09/2020

Conversión de eventos desde esquemas preconceptuales en código PL/pgSQL: simulación de software en la cuarta revolución industrial

Carlos Mario Zapata-Jaramillo[1], Juan Sebastián Zapata-Tamayo[1], Paola Andrea Noreña Cardona [1]

cmzapata@unal.edu.co; jzapatat@unal.edu.co; panorenac@unal.edu.co

[1] Universidad Nacional de Colombia, Carrera 80 No. 65-223 Of. M8A-310, 050036, Medellín, Colombia.

DOI: 10.17013/risti.39.18-34

Resumen: La simulación es una de las áeas clave de la Industria 4.0. En la industria de software, la simulación se aplica en la predicción del comportamiento de las aplicaciones a partir del modelado. Una forma de simulación se relaciona con la obtención de código fuente a partir de los modelos, que en el caso de los eventos se suele realizar desde diferentes esquemas conceptuales hacia lenguajes basados en disparadores y viceversa, sin incluir el PL/pgSQL. Por ello, en este artículo se propone un conjunto de reglas heurísticas para obtener código PL/pgSQL a partir de esquemas preconceptuales, como una forma de simular el comportamiento de las aplicaciones que incluyen eventos. Estas reglas se validan mediante un estudio de laboratorio correspondiente a la expansión de una epidemia. Mediante la ejecución de las reglas en el estudio de laboratorio se puede simular el comportamiento futuro de la aplicación y los resultados asociados.

Palabras-clave: Simulación; Industria 4.0; Eventos; Esquemas preconceptuales; PL/pgSQL.

Event conversion from pre-conceptual schemas to PostgreSQL code: software simulation in the fourth industrial revolution

Abstract: Simulation is one of the key areas of Industry 4.0. Simulation in the software industry is applied to the prediction of the application behavior from modeling. One kind of simulation is related to obtaining source code from models; in the case of events, such simulation can be achieved from some conceptual schemas to trigger-based languages and *vice versa*, but excluding PL/pgSQL. For this reason, in this paper, we propose a set of heuristic rules for obtaining PL/pgSQL code from pre-conceptual schemas, as a way to simulate the behavior of applications that include events. Such rules are validated by using a lab study related to the expansion of an epidemic. We can simulate the future behavior of the application and the associated results by executing the rules in the lab study.

Keywords: Simulation; Industry 4.0; Events; Pre-conceptual schemas; PostgreSQL.

1. Introducción

Baygin, Yetis, Karakose, & Akin (2016) presentan nueve áreas clave de la Industria 4.0. La simulación, objeto de este artículo, es una de ellas y se entiende como la reproducción virtual de los procesos que ocurren en las industrias. En la industria de software, y más concretamente en la ingeniería de software, la simulación puede contribuir en la evaluación del impacto de las aplicaciones y también en procesos de rediseño (Tanir, 2017) a partir del modelado.

La generación de código a partir de modelos es una forma de simulación que puede ser manual o automática (Boulanger, 2018) y siempre requiere la implementación de un cierto número de reglas para asegurar que los requisitos definidos en el diseño se consideren. Algunos modelos se traducen en lenguajes como el *Structured Query Language* SQL (Awiti, Vaisman, & Simányi, 2020; Teorey, Lightstone, Nadeau, & Jagadish, 2005; Chaverra, 2011; Chochlik, Kostolny, & Martincova, 2015). En otras propuestas se hace ingeniería inversa a porciones de código en lenguajes basados en disparadores, como el *Programming Language Structured Query Language* PL/SQL (Methakullawat & Limpiyakorn, 2014; Habringer, Moser, & Pichler, 2014; Fernández *et al.*, 2019), para generar modelos. Los eventos, entendidos como fenómenos que ocurren en los procesos de negocios y que permiten iniciar operaciones, se suelen modelar en diferentes esquemas conceptuales como redes de Petri (Narciso & Piera, 2017), *Business Process Modeling and Notation* BPMN (Object Management Group, 2011) y esquemas preconceptuales (Zapata-Jaramillo, 2011), a partir de los cuales se pueden convertir en diferentes lenguajes de programación, como PL/SQL (El-Hayat, Toufik, & Bahaj, 2020; Zapata-Tamayo, 2019). Sin embargo, en ninguna de estas propuestas se toma en consideración el PL/pgSQL, que es un lenguaje de programación basado en disparadores del gestor de bases de datos PostgreSQL.

Por ello, en este artículo se propone un conjunto de reglas heurísticas que posibilita la generación automática de código en PL/pgSQL a partir de esquemas preconceptuales en los que se modelan eventos, de manera que se pueda estudiar el comportamiento de este tipo de aplicaciones en los procesos de simulación que se realizan en la industria de software, en el contexto de la Industria 4.0. Como forma de validación se selecciona un estudio de laboratorio proveniente de la literatura especializada (Noreña, 2018) correspondiente al comportamiento de una epidemia en expansión.

Este artículo se organiza de la siguiente manera: en la Sección 2 se presenta el marco conceptual de la propuesta; en la Sección 3 se presentan los antecedentes; en la Sección 4 se propone la solución y su consecuente validación con un caso de laboratorio; las conclusiones y el trabajo futuro se discuten en la Sección 5.

2. Marco conceptual

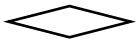
2.1. Simulación en la Industria 4.0

Baygin *et al.* (2016) establecen como uno de los fines de la Industria 4.0 la posibilidad de unir las tecnologías de información con la industria, de modo que se puedan crear fábricas inteligentes. Esa visión se materializa con nueve áreas tecnológicas diferentes:

Big Data y Analítica, Robots Autónomos, Simulación, Integración Horizontal y Vertical de Sistemas, Internet de las Cosas, Ciberseguridad, *Cloud,* Manufactura Aditiva y Realidad Aumentada. Tanir (2017) define la simulación como una técnica de propósito general de gran valor para evaluar espacios de solución de múltiples facetas de manera temprana durante las fases de planeación y diseño sin tener necesidad de un despliegue físico de las posibles alternativas de diseño. Por ello, Tanir (2017) establece que la simulación permite evaluar los impactos que se generen en los procesos de negocios y en las tecnologías de la información, tomando en consideración que se pueden requerir procesos de rediseño. Para Gunal (2019) la simulación puede desempeñar un rol primordial en la realización de la Industria 4.0.

2.2. Representación de eventos en esquemas conceptuales

Luckham (2011) define un evento como algo que ocurre en un sistema o dominio en particular y que sirve para dirigir automáticamente los procesos de negocio. Los eventos se representan en diferentes esquemas conceptuales. Narciso y Piera (2017) emplean transiciones entre los diferentes estados en una red de Petri. En BPMN (*Object Management Group*, 2011), se representan los estados con círculos que encierran diferentes elementos dependiendo de la naturaleza del evento (sobres de carta para eventos de mensaje, relojes para eventos de tiempo, triángulos para eventos tipo señal, etc.). Para efectos de este artículo, se usa la notación de eventos que se define en esquemas preconceptuales (Zapata-Jaramillo, 2011; Noreña (2018), ya que existe ya un conjunto de reglas heurísticas que, a partir de estos esquemas, permiten la traducción a código PL/SQL (Zapata-Tamayo, 2019). Los principales símbolos de los esquemas preconceptuales (incluyendo la notación correspondiente a eventos) se muestran en la Tabla 1.

Símbolo	Explicación
	Concepto: Sustantivo o sintagma nominal del dominio en singular, por ejemplo, *patient*
	Relación estructural: Verbo "ser" o "estar" en tercera persona del singular
	Relación dinámica: Verbo de acción u operación relacionado con el dominio en tercera persona del singular, por ejemplo, *inserts*
	Condicional: Contenedor gráfico de expresiones para evaluar su cumplimiento, por ejemplo, *Time = 0 weeks*
	Instancia: Posible número o texto que se asigna a un concepto, por ejemplo *"stop"*
	Conexión: Flecha para conectar conceptos con relaciones y viceversa
	Implicación: Flecha de tipo causa-efecto para conectar condicionales y relaciones dinámicas
	Relación eventual: Verbo que no necesita agentes, por ejemplo, *arrives*
	Evento: Combinación de conceptos y relaciones eventuales, por ejemplo, *time passes*

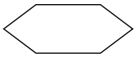
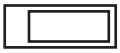
	Variable: Concepto independiente cuyo valor cambia a lo largo del proceso, por ejemplo, *growth rate*
	Parámetro: Concepto independiente con un valor fijo a lo largo de la simulación, por ejemplo, *population*
	Operador: Símbolo que permite incluir operaciones en el esquema preconceptual, por ejemplo +, /, sin, max, etc.
	Línea: Conexión simple para ligar conceptos, operadores e instancias para crear árboles de especificación
	Conjunto concepto-atributo: Nodo para evaluar conjuntos que incluyan un concepto y un atributo particular, por ejemplo, *patient.amount*
	Especificaciones/condiciones iniciales/restricciones: Contenedor que permite incluir expresiones en forma de árbol. Se debe etiquetar como “condiciones iniciales” para expresar valores que se asignan a las variables, conceptos y parámetros al inicio de la simulación. Expresiones rodeadas por llaves son las restricciones que se deben cumplir
	Conexión punteada: Línea para ligar condiciones iniciales o restricciones o especificaciones con los elementos del esquema preconceptual

Tabla 1 – Símbolos de los esquemas preconceptuales (Adaptado de Zapata-Jaramillo, 2011 y Noreña, 2018)

2.3. PL/pgSQL

Worsley y Drake (2001) definen PostgreSQL como un sistema de gestión de bases de datos objeto relacional de código abierto que incluye los comandos típicos del SQL junto con varias características adicionales, como búsquedas complejas, disparadores (*triggers*) y control de concurrencia multiversión. Además, el usuario puede extender PostgreSQL con lenguaje PL/pgSQL, que es el lenguaje de programación de este sistema de gestión de base de datos. Por ser el tema de este artículo, se muestra seguidamente la estructura de las funciones:

```
CREATE FUNCTION function_identifier () RETURNS opaque AS ‘
  DECLARE
    declarations;
    [...]
  BEGIN
    statements;
    [...]
  END;
‘ LANGUAGE ‘plpgsql’;
```

Los *triggers* se usan para invocar las funciones en el caso de que se requiera su ejecución de manera automática ante un evento. La sintaxis, en ese caso, es la siguiente:

```
CREATE TRIGGER function_identifier
  [AFTER/BEFORE] [CREATE/UPDATE/DELETE] OF field ON table
  FOR EACH ROW
  EXECUTE PROCEDURE function_identifier ();
```

3. Antecedentes

Para Boulanger (2018), una élite de expertos realiza la programación de una aplicación de software, aunque ese proceso se puede automatizar con la asistencia manual de un conjunto de reglas heurísticas o incorporando esas reglas en una herramienta con fines de automatización. Dado que el tema central de este artículo es la generación de código PL/pgSQL desde modelos que incluyan eventos, una revisión sistemática de literatura arroja que no existen propuestas asociadas con ese tópico en particular, pero hay similitudes que son de utilidad para el proceso.

3.1. Desde modelos hacia SQL

Awiti *et al.* (2020) parten de BPMN para obtener expresiones SQL. Teorey *et al.* (2005) y Chochlik *et al.* (2015) parten del diagrama entidad-relación para generar SQL. De manera similar, Chaverra (2011) emplea el esquema preconceptual para generar el diagrama entidad-relación correspondiente y, simultáneamente, la estructura de las tablas en SQL. El diagrama entidad-relación es un esquema conceptual que se usa para representar la estructura del dominio, por lo cual los eventos no se pueden representar en él. En los esquemas preconceptuales se pueden representar los eventos, pero esta porción del esquema no es la que emplea Chaverra (2011) para su proceso de generación de código. Además, si bien SQL es el punto de partida para PL/pgSQL, el SQL tampoco posee sintaxis que permita representar los eventos. Se resalta de estas propuestas la adición de reglas heurísticas para transformar esquemas preconceptuales en sentencias SQL.

3.2. Desde PL/SQL hacia esquemas conceptuales y viceversa

La ingeniería inversa se usa para recuperar esquemas conceptuales cuando existe alguna porción de código legado. Methakullawat y Limpiyakorn (2014) utilizan código legado PL/SQL para generar el diagrama de clases de UML (*Unified Modeling Language*). Las sentencias que incluyen procedimientos sólo las utilizan para generar la parte estructural del diagrama de clases. De forma similar, Habringer *et al.* (2014) usan procedimientos almacenados en PL/SQL para generar el diagrama de flujo de datos, que en cierta forma es también estructural. Ni en el diagrama de clases de UML ni en el diagrama de flujo de datos se toman en cuenta los eventos, aunque el lenguaje PL/SQL sí los soporta. Fernández *et al.* (2019) usan *triggers* de PL/SQL para obtener código Java usando diagramas de clases.

El-Hayat *et al.* (2020) parten de un diagrama de clases enriquecido con OCL para definir *triggers* en PL/SQL. Zapata-Tamayo (2019) parte de esquemas preconceptuales que incluyen eventos y, con base en un conjunto de reglas heurísticas, genera sentencias PL/

SQL con la sintaxis completa, de forma tal que se puede ejecutar en un gestor de Oracle™. Sin embargo, Oracle™ no es un gestor de código abierto y requiere licenciamiento para su uso. En esta propuesta, sin embargo, no se considera la traducción de las reglas a código PL/pgSQL, que sí es de código abierto. Al igual que el PL/SQL, su homólogo de código abierto PL/pgSQL también tiene una sintaxis apropiada para soportar eventos.

4. Solución y validación

Gunal (2019) define para la simulación en Industria 4.0 un "gemelo digital", que es un software para controlar elementos físicos de una plataforma colaborativa de procesos industriales de negocios y para recolectar datos de esa plataforma que permitan predecir y mejorar su funcionamiento. En las empresas de software no se desarrolla un producto tangible y, por ello, el gemelo digital de Gunal (2019) adquiere su sentido cuando se emplean modelos para simular las aplicaciones definitivas y su comportamiento, como es el caso de la epidemia de influenza que se estudia en este artículo. Por esta razón, el tipo de simulación que se propone con la conversión de un modelo basado en eventos (en este caso el esquema preconceptual) en *triggers* ejecutables en PL/pgSQL posibilitan la adquisición de datos desde etapas preliminares del diseño, actuando como gemelo digital de una aplicación definitiva que permita estudiar un fenómeno desde el punto de vista del modelado.

Zapata-Tamayo (2019) usa las reglas heurísticas de Chaverra (2011) para generar las sentencias SQL correspondientes a la creación de tablas, especialmente aquellas en las cuales se van a almacenar los datos. Además de esas reglas, en este artículo se aprovechan las reglas 1 y 2 de Zapata-Tamayo (2019), en las cuales se generan las tablas correspondientes a las variables (regla 1) y parámetros (regla 2), pues su funcionamiento es idéntico en su traducción a lo que allí se presenta para el lenguaje SQL. Finalmente, se considera el caso de los nombres de variables y parámetros que coinciden con palabras reservadas del lenguaje, adicionando un guión bajo (_) antes del nombre. Adicionalmente, se consideran algunas variaciones a las reglas de Zapata-Tamayo (2019):

- **Sintaxis del PL/pgSQL.** La sintaxis específica del PL/pgSQL tiene algunas diferencias con el PL/SQL. La diferencia fundamental radica en el hecho de que los *triggers* se deben programar como funciones y el código de creación del *trigger* sólo se utiliza para llamar la función correspondiente. En la regla 3 de Zapata-Tamayo (2019) hay que adicionar algunas líneas de la sintaxis específica, como se muestra en la Tabla 2.

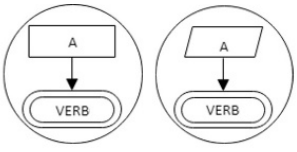
Precondición	Postcondición
	CREATE TRIGGER -"A_VERB" EXECUTE PROCEDURE "A_VERB" () CREATE FUNCTION -"A_VERB" () RETURNS TRIGGER AS $"A_VERB" ()$ END $"A_VERB" ()$ LANGUAGE plpgsql
Nota: La tercera, quinta y sexta línea de la expresión CREATE FUNCTION son necesarias para la ejecución.	

Tabla 2 – Modificación a la regla heurística 3 de Zapata-Tamayo (2019).

- **Efecto de dividir un evento en el modelo en dos *triggers* en lenguaje de programación para evitar los problemas de las tablas mutantes.** Se deben incluir todos los elementos de la condición inicial del evento del modelo en cada uno de los *triggers* en lenguaje de programación, dado que es posible que se presenten problemas de simultaneidad que hagan disparar otro *trigger* con la misma condición final. Ésta es una variación de la Regla 7 de Zapata-Tamayo (2019), tal como se muestra en la Tabla 3.
- **Presencia de dos o más condicionales en paralelo en una especificación de evento.** Hay que adicionar un bloque de código por cada condicional en paralelo que se tenga, tomando la precaución de incluir en el bloque DECLARE todas las variables y parámetros que se requieran en la evaluación de los condicionales. Ésta es una variación a la regla 10 de Zapata-Tamayo (2019), como se ilustra en la Tabla 4.

Precondición	Postcondición

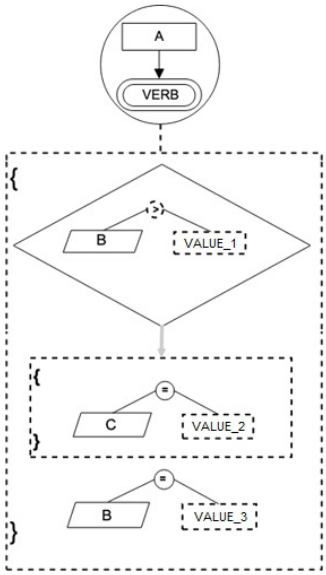

Nota: Se requiere duplicar el *trigger* porque, como se nota en el modelo, la variable B se invoca tanto para iniciar el evento como para finalizarlo, generando un ciclo infinito. La sintaxis también es similar si se trata del campo de una tabla, pero en ese caso se invocaría en el SELECT la tabla correspondiente y no la tabla *variable*.
	CREATE TRIGGER “A_VERB”
	AFTER INSERT OF “B” ON “VARIABLE”
	FOR EACH ROW
	EXECUTE PROCEDURE “A_VERB” ()
	
	CREATE FUNCTION -”A_VERB” ()
	RETURNS TRIGGER
	AS $”A_VERB” ()$
	DECLARE
	“v_” + B B.TYPE;
	BEGIN
	SELECT B FROM [VARIABLE] INTO “v_” + B;
	IF “v_” + B [CONDICION] VALUE_1 THEN
	UPDATE [VARIABLE] SET C=’VALUE_2’;
	END IF;
	RETURN NULL;
	END;
	$”A_VERB” ()$
	LANGUAGE plpgsql
	
	CREATE TRIGGER “A_VERB” + “_1”
	AFTER INSERT OF “C” ON “VARIABLE”
	FOR EACH ROW
	EXECUTE PROCEDURE “A_VERB” + “_1” ()
	
	CREATE FUNCTION “A_VERB” + “_1” ()
	RETURNS TRIGGER
	AS $”A_VERB” + “_1” ()$
	DECLARE
	“v_” + B B.TYPE;
	BEGIN
	SELECT B FROM [VARIABLE] INTO “v_” + B;
	IF “v_” + B [CONDICION] VALUE_1 THEN
	UPDATE [VARIABLE] SET B=’VALUE_3’;
	END IF;
	RETURN NULL;
	END;
	$”A_VERB” ()$
	LANGUAGE plpgsql

Tabla 3 – Modificación a la regla heurística 7 de Zapata-Tamayo (2019).

Con el fin de validar el efecto de las modificaciones a las reglas, se selecciona un caso de laboratorio de la literatura especializada. Noreña (2018) presenta un esquema preconceptual correspondiente al modelo epidémico de una ciudad infectada con influenza. Ese esquema preconceptual incluye los diferentes eventos que controlan el incremento en la cantidad de pacientes que sufren la enfermedad. Como el esquema se encuentra originalmente en el idioma inglés, la aplicación de las reglas se realiza respetando este lenguaje. En el esquema se hacen algunas modificaciones en relación con el original:

Precondición	**Postcondición**

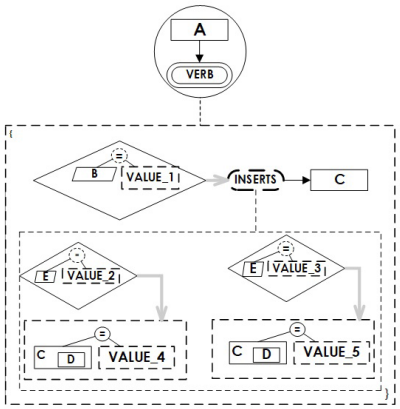

Nota: Si hubiera más ciclos paralelos, se deberían agregar más sentencias IF después de la línea 18 de la creación de la función. Los árboles de asignación pueden ser tan complejos como se requiera, pero hay que definir variables para cada uno de los elementos (variables, parámetros o atributos) que participen en la asignación.
	CREATE TRIGGER “A_VERB”
	AFTER INSERT OF “B” ON “VARIABLE”
	FOR EACH ROW
	EXECUTE PROCEDURE “A_VERB” ()
	
	CREATE FUNCTION -”A_VERB” ()
	RETURNS TRIGGER
	AS $”A_VERB” ()$
	DECLARE
	“v_” + B B.TYPE;
	“v_” + E E.TYPE;
	BEGIN
	SELECT B FROM [VARIABLE] INTO “v_” + B;
	SELECT E FROM [VARIABLE] INTO “v_” + E;
	IF “v_” + B [CONDICION] VALUE_1 THEN
	IF “v_” + E [CONDICION] VALUE_2 THEN
	INSERT INTO C (D)
	VALUES (VALUE_4)
	END IF
	IF “v_” + E [CONDICION] VALUE_3 THEN
	INSERT INTO C (D)
	VALUES (VALUE_5)
	END IF
	END IF;
	RETURN NULL;
	END;
	$”A_VERB” ()$
	LANGUAGE plpgsql

Tabla 4 – Modificación a la regla heurística 10 de Zapata-Tamayo (2019).

- La constante de integración, que en el modelo de Noreña (2018) se consideraba un parámetro, se considera ahora una variable, pues depende del número de pacientes infectados en la semana cero. El valor inicial de esta variable es cero.
- La tasa de crecimiento, que también se consideraba un parámetro, también se considera ahora una variable, pues depende del número de pacientes infectados en la semana 1 y de la constante de integración. El valor inicial de esta variable es cero.
- Se excluyen de las condiciones iniciales los vectores de tiempo y número de pacientes que se consideran registros iniciales en la tabla paciente, necesarios para el cálculo de la constante de integración y la tasa de crecimiento. En su lugar, se modela un evento denominado *patient amount arrives* (número de

pacientes llega), que sirve para actualizar la tabla de pacientes con los valores correspondientes a los infectados en las semanas 0 y 1, valores que se requieren para el cálculo de las variables *constante de integración* y *tasa de crecimiento* y su actualización con los valores definitivos en dicha tabla.

- Se incluyen en las condiciones iniciales el parámetro *end time* para ponerle fin a la simulación en un tiempo de 6 semanas y la variable *timestamp*, que puede tomar los valores *next* y *stop*, con el fin de controlar la ejecución de los eventos en la simulación. Nótese que al final de cada uno de los eventos hay una asignación diferente del valor de *timestamp*.
- Se reemplazan los vectores tiempo y paciente por los atributos *local time* y *amount*, correspondientes a la tabla *patient*.

El modelo completo, modificado según las consideraciones anotadas, se presenta en la Figura 1. Las flechas de conexión posibilitan la lectura entre los conceptos y las relaciones (por ejemplo "paciente tiene hora local y cantidad"). Las flechas de implicación generan relación causa-efecto (por ejemplo, "cuando el tiempo pasa, la cantidad de pacientes crece"). Las conexiones punteadas permiten detallar los eventos y los operadores permiten complementar las fórmulas (por ejemplo, la parte inicial de "llega cantidad de pacientes" se lee "si la marca de tiempo es igual a pare y el tiempo es menor a dos semanas, se inserta un paciente"). Con base en el modelo de la Figura 1, se inicia la aplicación de las reglas 1 y 2, correspondientes a la creación de las tablas *variable* y *parameter*, como se muestra en la Tabla 5. Además, aplicando las reglas de Chaverra (2011) se crea la tabla *patient*, que sirve para almacenar los datos correspondientes a la cantidad de personas infectadas en la población en cada semana.

El evento *time passes* requiere la variable *timestamp* tanto en el condicional como en el final del evento, por lo cual requiere la aplicación de la regla 7, generando dos *triggers* diferentes tal como se muestra en las Tablas 6 y 7. Además, con la aplicación de la regla 3 se crea el código tanto de cada uno de los *triggers* como de las funciones correspondientes. Ahora, en el caso del evento *patient amount arrives* se debe aplicar la regla 10 para generar las sentencias IF paralelas que se dan en el cuerpo del *trigger*, como se muestra en las Tablas 8 y 9. Dado que el evento termina con la actualización de la variable *timestamp*, que también se necesita en el condicional, se aplica también la regla 7 para generar un segundo *trigger*, pero, por efectos de espacio, ese segundo *trigger* no se incluye en este artículo, al igual que los que corresponden al evento *patient amount grows*, pero su código es similar al que se muestra en los otros eventos.

Una vez se genera la totalidad del código PL/pgSQL mediante la utilización de las reglas, es necesario cambiar los valores en el PostgreSQL para que se inicie la ejecución de los diferentes *triggers*.Como se aprecia en la Tabla 5, antes de iniciar la simulación el valor de *timestamp* está vacío, por lo cual se debe colocar en el valor *stop*, que es la primera condición para ejecutar el proceso. Eso se logra con el siguiente comando en PostgreSQL:

```
UPDATE “variable” SET _timestamp = ‘stop’;
```

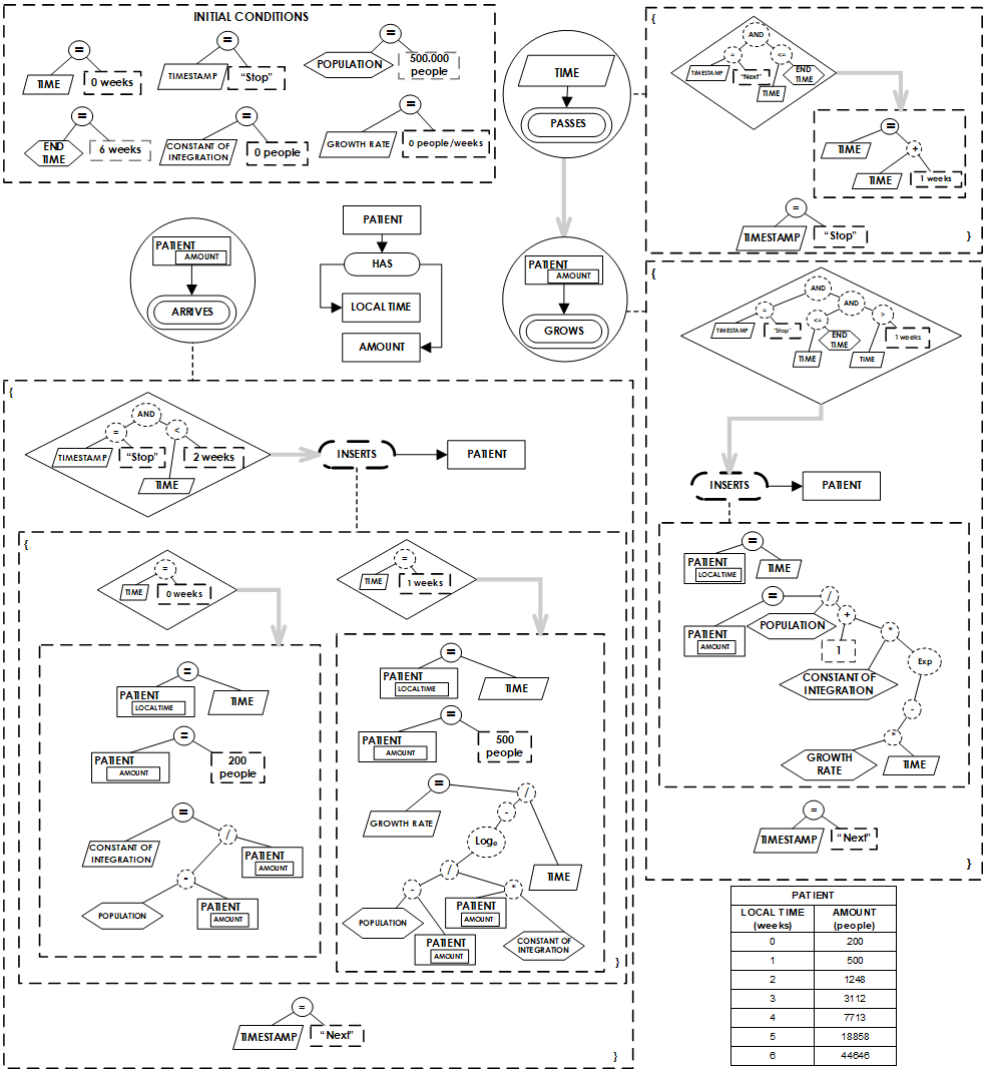


Figura 1 – Adaptación del modelo epidemiológico de Noreña (2018)

Cuando se inicia ese proceso y con los valores que ya se tienen en las tablas *variable*, *parameter* y *patient* se generan los siete valores que se muestran en la Figura 1 para la tabla *patient* en *local_time* 0 a 6.

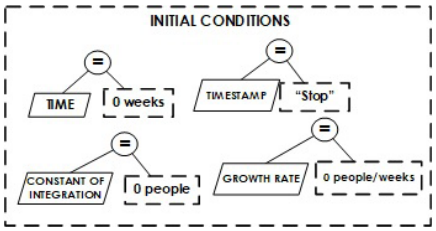
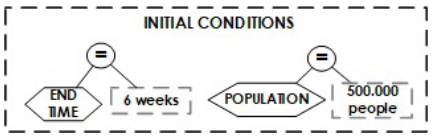
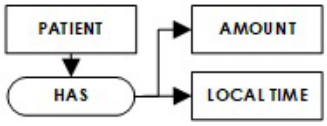
Esquema preconceptual	Código generado PL/pgSQL
 Nota: a las variables *time* y *timestamp* se les antepone un guión bajo, por ser palabras reservadas del lenguaje	CREATE TABLE variable (id serial PRIMARY KEY, _timestamp text NOT NULL, _time float NOT NULL, growth_rate float NOT NULL, constant_of_integration float NOT NULL); INSERT INTO variable (_timestamp, _time, growth_rate, constant_of_integration) VALUES (‘’, 0, 0, 0);
 Nota: al parámetro *population* se le antepone un guión bajo, por ser palabra reservada del lenguaje	CREATE TABLE parameter (id serial PRIMARY KEY, _population float NOT NULL, end_time float NOT NULL); INSERT INTO parameter (_population, end_time) VALUES (500000, 6);
	CREATE TABLE patient (id serial PRIMARY KEY, local_time float NOT NULL DEFAULT 0, amount float NOT NULL DEFAULT 0);

Tabla 5 – Aplicación de las reglas 1 y 2 para las tablas *variable* y *parameter* y de las reglas de Chaverra (2011) para la creación de la tabla *patient*.

Ahora, si se hace un UPDATE suficientemente grande del parámetro *end_time*, es posible determinar cuándo la epidemia alcanzará toda la población en la ciudad. Para este caso en particular, se encuentra que en la semana 23 se alcanza un *amount* en *patient* de 499999.1327998074, un número suficientemente cercano al de la población total de la ciudad, que es de 500000. Si se quiere ser suficientemente preciso, en la semana 49 se llega a ese número exactamente.

5. Conclusiones y trabajo futuro

En este artículo se propuso un conjunto de reglas heurísticas que posibilitan la generación de código PL/pgSQL a partir de modelos que incluyen eventos representados en esquemas preconceptuales.

Estas reglas constituyen un ejemplo de la manera en que la industria del software emplea el área clave de simulación correspondiente a la Industria 4.0 para predecir el comportamiento de sistemas complejos, lo cual se pudo comprobar mediante un caso de laboratorio correspondiente a la literatura especializada y en el cual se ejemplifica

la expansión de una epidemia de influenza en una ciudad. Con el código generado fue posible la generación automática de los valores que se predijeron para el modelo. El código se generó completamente a partir de las reglas y fue completamente funcional en el gestor de bases de datos PostgreSQL, de código abierto.

Esquema preconceptual	**Código generado PL/pgSQL**

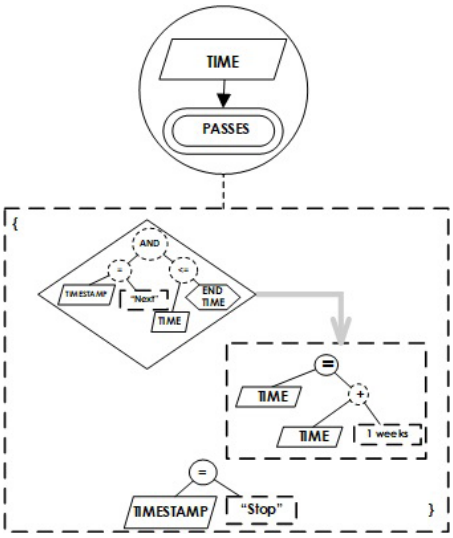

Nota: en esta parte de la Tabla se puede apreciar la generación del código de los dos *triggers* y el código de la primera función, correspondiente a la actualización de la variable *time* con el incremento de una semana.
	CREATE TRIGGER time_passes
	AFTER UPDATE OF _timestamp ON variable
	FOR EACH ROW
	EXECUTE PROCEDURE time_passes ();
	
	CREATE TRIGGER time_passes_1
	AFTER UPDATE OF _time ON variable
	FOR EACH ROW
	EXECUTE PROCEDURE time_passes_1 ();
	
	CREATE FUNCTION time_passes ()
	RETURNS TRIGGER
	AS $time_passes$
	DECLARE
	v_timestamp text;
	v_time float;
	v_end_time float;
	BEGIN
	SELECT
	_timestamp
	FROM
	variable INTO v_timestamp;
	SELECT
	_time
	FROM
	variable INTO v_time;
	SELECT
	end_time
	FROM
	parameter INTO v_end_time;
	IF (v_timestamp = 'next') THEN
	IF (v_time <= v_end_time) THEN
	UPDATE
	variable
	SET
	_time = v_time + 1;
	END IF;
	END IF;
	RETURN NULL;
	END;
	$time_passes$
	LANGUAGE plpgsql;

Tabla 6 – Aplicación de la regla 7 para la creación de los *triggers* correspondientes al evento *time passes*.

Esquema preconceptual	Código generado PL/pgSQL

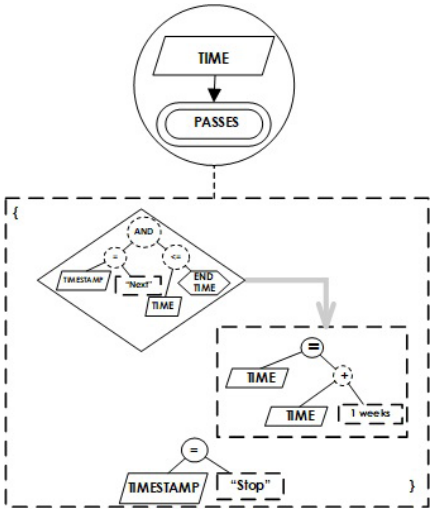

Nota: en esta parte de la tabla se muestra la aplicación de la regla 7 para generar la segunda función, en la cual se actualiza la variable *timestamp* para que tenga el valor *stop*, que es el que dispara la aplicación de otros *triggers*.
	CREATE FUNCTION time_passes_1 ()
	RETURNS TRIGGER
	AS $time_passes_1$
	DECLARE
	v_timestamp text;
	v_time float;
	v_end_time float;
	BEGIN
	SELECT
	_timestamp
	FROM
	variable INTO v_timestamp;
	SELECT
	_time
	FROM
	variable INTO v_time;
	SELECT
	end_time
	FROM
	parameter INTO v_end_time;
	IF (v_timestamp = 'next') THEN
	IF (v_time <= v_end_time) THEN
	UPDATE
	variable
	SET
	_timestamp = 'stop';
	END IF;
	END IF;
	RETURN NULL;
	END;
	$time_passes_1$
	LANGUAGE plpgsql;

Tabla 7 – Aplicación de la regla 7 para la creación de los *triggers* correspondientes al evento *time passes* en su segunda función.

Como líneas de trabajo futuro, se proponen las siguientes:

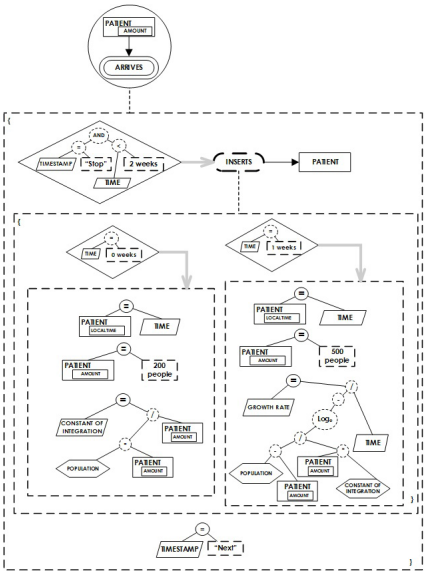
Esquema preconceptual	**Código generado PL/pgSQL**
 Nota: en esta parte de la Tabla se presenta la aplicación de la regla 3 para generar el código del *trigger* y el inicio del código de la función. Acá se muestra la declaración de las variables y los comandos SELECT para obtener los diferentes valores de las tablas correspondientes, que se usarán luego en el cálculo.	CREATE TRIGGER patient_amount_arrives AFTER UPDATE OF _timestamp ON variable FOR EACH ROW EXECUTE PROCEDURE patient_amount_arrives (); CREATE FUNCTION patient_amount_arrives () RETURNS TRIGGER AS $patient_amount_arrives$ DECLARE v_timestamp text; v_time float; v_population float; v_constant_of_integration float; v_growth_rate float; v_patient_local_time float; v_patient_amount float; BEGIN SELECT _timestamp FROM variable INTO v_timestamp; SELECT _time FROM variable INTO v_time; SELECT _population FROM parameter INTO v_population; SELECT constant_of_integration FROM variable INTO v_constant_of_integration; SELECT growth_rate FROM variable INTO v_growth_rate; SELECT local_time FROM patient INTO v_patient_local_time; SELECT amount FROM patient INTO v_patient_amount;

Tabla 8 – Aplicación de la regla 3 para la creación de uno de los *triggers* correspondiente al evento *patient amount arrives*.

- Elaboración de una herramienta que incluya las reglas que se definen en este artículo y las que define Zapata-Tamayo (2019) para la generación automática de código fuente, tanto en PL/SQL como en PL/pgSQL a partir de las representaciones en esquemas preconceptuales.

- Estudio de otros fenómenos relacionados con eventos complejos para determinar reglas heurísticas adicionales en situaciones que así lo requieran para el funcionamiento autónomo de los eventos en este tipo de simulación.
- Exploración de otras maneras en que la simulación basada en eventos pueda contribuir en la Industria 4.0.

Esquema preconceptual	**Código generado PL/pgSQL**

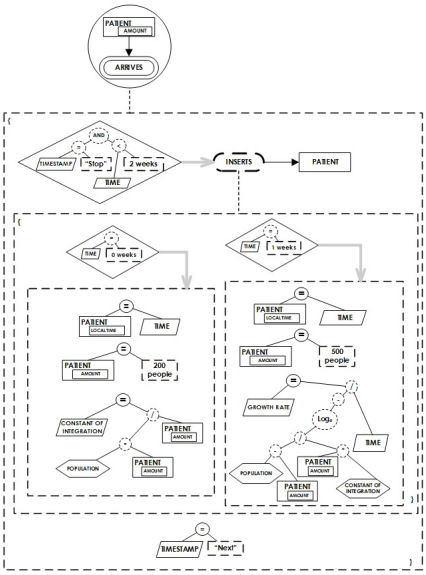

Nota: en esta parte de la tabla se aprecia la aplicación de la regla 10 para los dos ciclos IF que se generan. Inicialmente, se hace el cálculo de *constant_of_integration* en la semana 0 y luego se calcula *growth_rate* en la semana 1.
	IF (v_timestamp = 'stop') THEN
	IF (v_time < 2) THEN
	IF (v_time = 0) THEN
	INSERT INTO patient (local_time, amount)
	VALUES (v_time, 200);
	SELECT amount
	FROM patient
	INTO v_patient_amount
	ORDER BY
	id DESC
	LIMIT 1;
	UPDATE variable
	SET
	constant_of_integration = ((v_population - v_patient_amount) / v_patient_amount);
	END IF;
	IF (v_time = 1) THEN
	INSERT INTO patient (local_time, amount)
	VALUES (v_time, 500);
	SELECT amount
	FROM patient
	INTO v_patient_amount
	ORDER BY
	id DESC
	LIMIT 1;
	UPDATE variable
	SET
	growth_rate = (- ln(((v_population - v_patient_amount) / (v_patient_amount * v_constant_of_integration)))) / v_time;
	END IF;
	END IF;
	END IF;
	RETURN NULL;
	END;
	$patient_amount_arrives$
	LANGUAGE plpgsql;

Tabla 9 – Aplicación de la regla 10 para la creación de uno de los *triggers* correspondiente al evento *patient amount arrives*.

Agradecimiento

Este artículo es parte del proyecto de Maestría titulado "Generación Semi-automática de Código PL/SQL a partir de Representaciones de Eventos Basadas en Esquemas Preconceptuales", patrocinado por la Facultad de Minas de la Universidad Nacional de Colombia, sede Medellín.

Referencias

Awiti, J., Vaisman, A., & Simányi, E. (2020). Design and implementation of ETL processes using BPMN and relational algebra. *Data and knowledge engineering, 129,1-14*. 10.1016/j.datak.2020.101837

Baygin, M., Yetis, H., Karakose, M., & Akin. E. (2016). An Effect Analysis of Industry 4.0 to Higher Education. In *15th International Conference on Information Technology Based Higher Education and Training (ITHET)*. Istanbul.

Boulanger, J. (2018). *Certifiable software applications 3: Downward cycle*. Elsevier Ltd.

Chaverra, J. J. (2011). *Generación Automática de Prototipos Funcionales a Partir de Esquemas Preconceptuales.* (Tesis de Maestría). Departamento de Ciencias de la Computación y de la Decisión, Universidad Nacional de Colombia, Medellín.

Chochlik, M., Kostolny, J., & Martincova, P. (2015). Metamodel describing a relational database schema. *Central European Researchers Journal*, *1*(1), 94–102.

El-Hayat, S., Toufik, F., & Bahaj, M. (2020). UML/OCL based design and the transition towards temporal object relational database with bitemporal data. *Journal of King Saud University—Computer and information sciences,* 32, 398–407.

Fernández, C., Molina, J., Bermúdez, F., Hoyos, J., Sevilla, D., & Cuesta, B. (2019). Developing a model-driven reengineering approach for migrating PL/SQL triggers to Java: a practical experience. *The journal of systems and software,* 151, 38–64.

Gunal, M. (2019). Simulation and the Fourth Industrial Revolution. In *Simulation for Industry 4.0: past, present, and future* (pp. 1–17). Springer.

Habringer, M., Moser, M., & Pichler, J. (2014). Reverse Engineering PL/SQL Legacy Code: An Experience Report. In *IEEE International Conference on Software Maintenance and Evolution*. Victoria, Canada.

Luckham, D. (2011). *Event Processing for Business: Organizing the Real-time Enterprise*. John Wiley & Sons.

Methakullawat, N., & Limpiyakorn, Y. (2014). Reengineering Legacy Code with Model Transformation. *International Journal of Software Engineering and Its Applications*, *8*(3), 97–110.

Narciso, M., & Piera, M. A. (2017). Modeling Discrete Event Systems Using Petri Nets, in *Robust Modelling and Simulation* (pp. 49–85). Springer.

Noreña, P. (2018). An Extension to Pre-conceptual Schemas for Refining Event Representation and Mathematical Notation. In *21st Conferencia Iberoamericana en Software Engineering (CIbSE)* (pp. 589–596). Bogotá, Colombia.

Object Management Group (2011). About the business process modelling and notation specification version 2.0 [Internet]. http://www.omg.org/spec/BPMN/2.0

Tanir O. (2017). Simulation-Based Software Engineering. In *Guide to Simulation-Based Disciplines. Simulation Foundations, Methods and Applications*. Springer.

Teorey, T., Lightstone, S., Nadeau, T., & Jagadish, H. V. (2005). *Database Modeling and Design (4th Edition)*. Morgan Kaufmann.

Worsley, J. & Drake, J. (2001). *Practical PostgreSQL*. O'Reilly.

Zapata-Jaramillo, C. M. (2011). *The UNC-Method revisited: elements of the new approach*. Lambert Academic Publishing.

Zapata-Tamayo, J. S. (2019). *Generación Semiautomática de Código PL/SQL a partir de Representaciones de Eventos Basadas en Esquemas Preconceptuales*. (Tesis de Maestría). Departamento de Ciencias de la Computación y de la Decisión, Universidad Nacional de Colombia, Medellín.