EDITORIAL
UNIAUTÓNOMA

MANUAL

PARA LA CREACIÓN DE ESQUEMAS PRECONCEPTUALES

CARLOS MARIO ZAPATA JARAMILLO / PAOLA ANDREA NOREÑA CARDONA
ELIZABETH SUESCÚN MONSALVE

Carlos Mario Zapata Jaramillo
Paola Andrea Noreña Cardona
Elizabeth Suescún Monsalve

Manual para la creación de esquemas preconceptuales

Manual para la creación de esquemas preconceptuales

Grupo de Investigación en Lenguajes Computacionales
Universidad Nacional de Colombia, Medellín, Colombia

Semillero de Ingeniería de Software
GICOMP: Grupo de Investigación en Computación
Universidad EAFIT, Medellín, Colombia

LinkedIn Group

YouTube Channel

ISBN: 978-628-7691-78-0

Diagramación: Paola Noreña y Elizabeth Suescún
Diseño de portada: Mariana Quintero-Ilustradora y Diseñadora gráfica.
La portada simboliza la participación inclusiva de hombres y mujeres en el campo de la computación, donde sus ideas, perspectivas y soluciones toman forma y se hacen comprensibles mediante esquemas preconceptuales.

Sello Editorial Uniautónoma del Cauca
Serie: Investigación
Editor General de Publicaciones: Ramsés López Santamaría
Calle 5 No. 3-85 Popayán, Colombia. Teléfono: PBX: 8213000 - Fax: 8214000
https://www.uniautonoma.edu.co/

Nosotros dedicamos esta obra
A todos nuestros estudiantes.

A Kiki, Sebas y Andy, quienes
me enseñaron el concepto
más importante: el amor.
- Carlos

A Martha, Fer, Jerónimo y Juancho
por su soporte incondicional.
- Paola

A mi hijo Thomas maestro de vida,
A través de ti aprendo, crezco y descubro.
- Elizabeth

Contenido

Lista de Figuras

Lista de Tablas

"Los esquemas preconceptuales dan valor tanto a proyectos ágiles como en cualquier metodología. Los esquemas nos entrega información bastante valiosa para las fases iniciales del proyecto y durante el transcurso de la implementación de estos, porque podemos identificar actores, funcionalidades, procesos, flujos, información relacionada al desarrollo y cálculos para la aplicación que se está construyendo. Nos da una mirada más aterrizada del proyecto de manera visual con el flujo y proceso del negocio y de cómo esto se combina con el resto de las herramientas que se vayan a utilizar. Mi recomendación es crear esquemas por cada feature para una mejor comprensión".

Guillermo Carrillo-Scrum Máster. En Sura

"He empleado esquemas para visualizar y representar el flujo de los procesos que he diseñado en las áreas en las que trabajo. Es fundamental resaltar que, gracias a su notación, todos los involucrados han logrado comprender de manera clara quiénes participan y qué se lleva a cabo en cada etapa del proceso."

Luis Fernando Guarín-Coordinador de negocios. En Essity

"A través de los esquemas preconceptuales se ha facilitado el aprendizaje, la retención de información y la transferencia de conocimientos del modelo del negocio."

Edwin Acosta, Software Engineer I, Despegar Colombia

"En el campo de la simulación de yacimientos de petróleo y gas, los esquemas preconceptuales han sido útiles para abstraer las ecuaciones que relacionan el transporte de los componentes que conforman los hidrocarburos, y su respectivo comportamiento termodinámico. Al plantear las ecuaciones de transporte multicomponente en términos de conceptos que se relacionan entre sí, obtuvimos un mayor entendimiento del panorama general y del flujo de información, permitiendo flexibilizar las herramientas computacionales que posteriormente desarrollamos. Además, al abstraer correctamente los conceptos que eran físicos de los que eran meramente matemáticos, como el método de solución de las ecuaciones, la extensión de la herramienta para considerar nuevos fenómenos físicos fue natural."

Steven Velásquez - Ingeniero de Sistemas.

"Los esquemas preconceptuales permiten una visualización de los datos que se van a integrar y de los criterios de aceptación de cada funcionalidad."

Alecsandra Piedrahita - Ingeniera Biomédica.

"Los esquemas preconceptuales ayudan a enriquecer la experiencia del usuario, ya que facilitan la comprensión del producto de software, sus características y funcionalidades. Y a su vez, son esenciales para la elaboración de pruebas de aceptación efectivas, tanto de las funcionalidades como del manejo de datos."

Julieth Jiménez - Ingeniera Matemática.

"Al permitir una interacción temprana y clara entre los interesados y el equipo de desarrollo, los esquemas preconceptuales contribuyen a la creación de productos de software sostenibles."

Evelyn Zapata, estudiante de Ingeniería de Sistemas.

"... Continuaré con los estudios y colocaré los esquemas preconceptuales como parte indispensable de la documentación que produciré. Creo que con los EP es posible llegar exactamente a donde queremos ir, en la necesidad real del cliente. Es un modelo que ambos stakeholders *y el equipo pueden entender porque representa sus procesos y puede ser comparable al inglés, como un idioma internacional. La visión que los EP dan a los escenarios que deben ser analizados y discutidos es asombrosa."*

Daniel Almeida - Desarrollador Java. En Brasil.

Prefacio

Querido lector,

Es un honor y un placer darte la bienvenida a esta obra, resultado de más de una década de dedicación y profundo aprendizaje acerca de los esquemas preconceptuales (EP). Desde el año 2007, se ha trabajado en la mejora y comprensión de estos esquemas, buscando siempre avanzar en su desarrollo y aplicación en áreas de computación como la ingeniería de software y específicamente para apoyar el diseño en software, el cual ha evolucionado significativamente en las últimas décadas, convirtiéndose en una etapa crucial en el ciclo de vida del desarrollo de software que actúa como puente entre la definición de los requisitos y la implementación. Cabe resaltar que el diseño es esencial para dar claridad sobre la solución, facilita la evolución, promueve la reutilización, permitiendo que las soluciones de software sean efectivas y escalables. En ese sentido, esta obra muestra los EP como una herramienta gráfica que apoya las etapas iniciales del diseño y se utiliza generalmente para representar conceptos y relaciones, antes de profundizar en el diseño de software formal. Los mismos son una representación sencilla por su cercanía con el lenguaje natural, facilitando el entendimiento de un dominio y es aquí donde se encuentra su gran potencial.

Este libro es un compendio de esfuerzos de diferentes aportes de la comunidad que tiene interés en EP, una travesía intelectual y emocional por sus fundamentos filosóficos, ontológicos y sistémicos producto de conversaciones, consensos y una profunda convicción de estar presentando una obra significativa para usar y profundizar en este tema. Por tanto, nuestro objetivo es proporcionarte una visión completa y detallada de los esquemas preconceptuales, ofreciendo la notación, reglas de equivalencia y ejemplos que puedan servirte como herramientas para aplicar conocimientos que puedan ser útiles tanto en el ámbito académico como en el práctico. A lo largo de estas páginas, te invitamos a explorar con curiosidad las características que componen los EP. Nuestro deseo es que cada capítulo te brinde nuevas perspectivas, te inspire y te incite a una reflexión más profunda sobre la naturaleza y propósito de los EP.

Este libro ha sido posible gracias a la colaboración y aporte valioso de personas

a lo largo de años que nos han inspirado a profundizar en este tema, por lo que agradecemos profundamente a nuestros colegas, por sus invaluables aportes y debates que enriquecieron nuestra comprensión; a los estudiantes, por inquietarnos con sus preguntas; agradecemos a Dios por ser fuente de inspiración y conocimiento; a nuestras familias y amigos, por su constante apoyo y paciencia durante los largos años de investigación y redacción; y, sobre todo, a tí, lector, por tomar este libro en tus manos y darle vida a través de tu lectura.

Queremos que esta obra sea una fuente de inspiración y una opción en el diseño en software para:

- *Estudiantes y docentes* de programas de ingeniería, ciencias de la computación y disciplinas afines que busquen herramientas metodológicas para representar y transformar el conocimiento en soluciones tecnológicas.
- *Profesionales y desarrolladores de software*, interesados en metodologías de análisis conceptual, diseño centrado en el dominio y generación de artefactos técnicos y de código.
- *Investigadores y científicos* en áreas como biología, física, medicina, ingeniería, ciencias sociales, entre otras; que requieren traducir conocimientos especializados en visualizaciones, simulaciones y herramientas digitales.

Este manual puede ser utilizado como:

- *Guía práctica y académica* para cursos de ingeniería de software, modelado conceptual, análisis de requisitos y diseño orientado a modelos.
- *Referencia metodológica* para conectar conocimiento del dominio con soluciones computacionales en proyectos interdisciplinarios.
- *Base para la generación* de artefactos ágiles, esquemas conceptuales y código fuente a partir de EP.

Esperamos que encuentres en estas páginas algo que resuene contigo, que te haga pensar, sentir y, quizás, cambiar. Este libro es tanto tuyo como nuestro, y en cada palabra late el deseo de compartir algo significativo, algo que vaya más allá de lo ordinario y te acompañe en tu propio viaje de descubrimiento. Haciendo que el interés y aplicación de los EP continue creciendo.

Con recomendación y entusiasmo,

Los autores

Capítulo 1

Introducción

Los esquemas preconceptuales (EP) son representaciones del conocimiento a partir del discurso del interesado, los cuales tienen una naturaleza intuitiva que facilita la comprensión del conocimiento tácito que emerge sobre el dominio de un sistema de software (Zapata-Tamayo y Zapata-Jaramillo, 2018). Los esquemas involucran características estructurales, dinámicas, télicas y eventuales; cuyo fundamento proporciona sincronía léxica, sintáctica y semántica entre las características. Las características de los EP se definen con el propósito de disminuir la brecha entre el discurso del interesado y el código fuente del sistema de software en el proceso de educción , en el cual los EP representan la captura y descubrimiento de la información relevante para la conformación de los requisitos del software que se debaten entre los interlocutores (Zapata, 2007, 2012). En suma, los esquemas preconceptuales también permiten la representación de procesos y conceptos del dominio en el diseño de productos de software.

Desde la propuesta original de los EP, Zapata (2007), al momento de la publicación de este libro, se han presentado algunas propuestas que contribuyen al mejoramiento de las características de los EP descritos a seguir:

- **Características Estructurales**: Zapata (2007) propone relaciones estructurales en el EP para representar conceptos clase y conceptos hoja. Manjarrés (2010) presenta una especificación para restricciones en conceptos hoja. Chaverra (2011) propone la representación de tipos de datos en EP y los atributos derivados.

- **Características Dinámicas**: Zapata (2007) define las relaciones dinámicas en el EP para representar procesos del sistema y Chaverra (2011) propone su especificación con notación matemática. Zapata *et al.* (2011a) exploran los EP como esquemas ejecutables.

- **Características Télicas**: Zapata *et al.* (2007b) proponen las relaciones

de logro. Vargas (2016) propone reglas de representación por su parte para relaciones de logro y problemas. Zapata y Castro (2017) presentan indicadores claves de rendimiento (KPI por sus siglas en inglés) como parte de la especificación de estas relaciones.

- **Características Eventuales**: Zapata (2012) propone una representación gráfica para los eventos del sistema. Noreña-Cardona *et al.* (2019) y Noreña-Cardona (2020) extienden la notación matemática de los EP para la especificación de eventos.

Algunas de estas propuestas se enfocan en las siguientes equivalencias para las características de los EP:

- **Equivalencias de EP a esquemas conceptuales:** Zapata (2007) define reglas heurísticas para traducir la notación de EP a esquemas conceptuales del lenguaje unificado de modelado (UML, por sus siglas en inglés). Zapata *et al.* (2007b) plantean una traducción de EP a diagramas de casos de uso. Chaverra (2011) y Zapata *et al.* (2011b) proponen la generación automática del diagrama entidad-relación desde EP. Zapata *et al.* (2011d) presentan una traducción entre el diagrama de objetivos y los EP. Zapata (2012) presenta el uso de esquemas conceptuales con su traducción desde EP. Zapata y Garcés (2008) generan una traducción desde los EP a diagramas de secuencias.

- **Equivalencias entre EP y artefactos ágiles:** Villamizar (2019) presenta equivalencias entre EP y otros artefactos como historias de usuario y *mock-up*.

- **Equivalencias de EP a Código fuente:** Zapata y Cardona (2008) implementan las reglas heurísticas de Zapata (2007). Manjarrés (2010) propone el código para las restricciones de conceptos hoja. Chaverra (2011) y Zapata *et al.* (2011b) presentan una traducción automática de las relaciones dinámicas en EP a código. Zapata *et al.* (2011c) proponen la generación automática del diagrama entidad-relación desde EP con reglas en UN-Lencep (lenguaje controlado) y SQL. Zapata (2012) ejemplifica un caso de estudio que incluye el EP y el código fuente. Calle (2016) propone patrones de diseño desde EP. Zapata y Castro (2017) definen el código de KPI para relaciones de logro. Zapata-Tamayo (2019) propone una generación semiautomática de código PL/SQL para la especificación de eventos. Velásquez (2019) y Noreña-Cardona

(2020) presentan código C++ y código Python respectivamente desde EP para eventos. Mosquera (2021) propone un conjunto de reglas heurísticas para aplicaciones móviles.

Las propuestas previas reflejan el esfuerzo por mejorar las características de los esquemas. Sin embargo, las características estructurales, dinámicas, télicas y eventuales se tratan de forma aislada en las diferentes propuestas. Ese tratamiento aislado dificulta la representación en un EP y la implementación en código fuente de cada una de las porciones de un sistema de software. Consecuentemente, puede surgir ambigüedad en el uso de los esquemas preconceptuales. Por este motivo se hace necesario presentar un manual guía homogéneo que sirva de referencia para la creación de EP y las mejoras eventualmente incorporadas.

En este libro se propone un manual para la creación de EP, como una especificación revisada de las características estructurales, dinámicas, télicas y eventuales de los EP. En este manual se integran todas las características de los EP que se recopilan desde la literatura y se presenta como un documento guía. Además, se genera el código fuente del sistema de software correspondiente a cada porción del esquema que permite cerrar el ciclo desde la especificación hasta la implementación. La especificación se valida en un caso de laboratorio: un sistema de salud para la evaluación del riesgo de fragilidad en adultos mayores con una arquitectura de microservicios dirigida por eventos. En ese caso de laboratorio se ejemplifica cada porción del EP y el código fuente correspondiente.

Con el manual para la creación de EP, se pretenden lograr las siguientes contribuciones:

(i) Eliminación de la ambigüedad en el uso de los EP, al incluir todas sus características en una misma fuente.

(ii) Definición de lineamientos para los EP en el proceso de educción de requisitos de un sistema de software.

(iii) Descripción de lineamientos para la automatización del proceso. En si, una visión homogénea de uso de los EP.

La estructura de este libro es la siguiente:

En el Capítulo 2 se definen los esquemas preconceptuales.

En el Capítulo 3 se relacionan los antecedentes y se analiza el problema.

En el Capítulo 4 se presenta la especificación de las características de los EP.

En el Capítulo 5 se ejemplifica la especificación de los EP en un caso de laboratorio.

En el Capítulo 6 se describen las conclusiones y los desafíos futuros para los EP.

Capítulo 2

Esquemas preconceptuales

Los esquemas preconceptuales (EP) son representaciones intermedias entre el lenguaje natural (conjunto de elementos de comunicación del ser humano, por ejemplo, un interesado que necesita un sistema de pedido) y un esquema conceptual (representación formal de un dominio, por ejemplo los diagramas UML), o un lenguaje formal (precisa símbolos, estructura y reglas basadas en matemáticas, lógica y computación, por ejemplo, lenguajes de programación, lenguajes de bases de datos, lógica de predicados, gramáticas formales de lingüística computacional, entre otros; Zapata 2007).

Un analista/científico puede usar el EP para reconocer el dominio de conocimiento y luego, el desarrollador puede desarrollar el sistema de software con base en el EP, como se resume en la Figura 2.1.

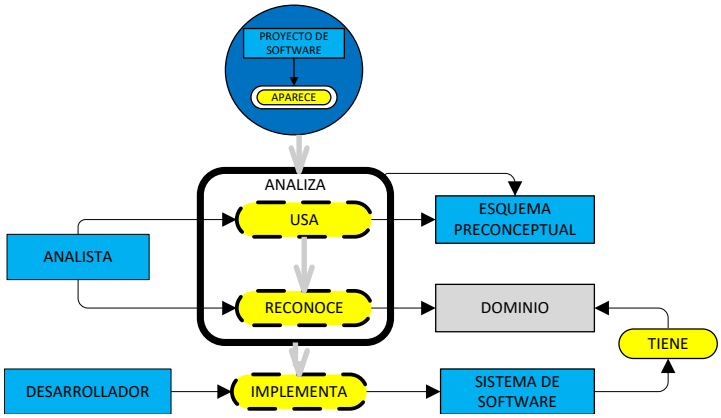


Figura 2.1. Uso de los EP. Los autores

Los EP se clasifican como modelos independientes (se refiere a aquellos modelos conceptuales que son independientes de una implementación, tecnología o plataforma) con una vista múltiple, es decir, una vista estructural y dinámica, según la arquitectura dirigida por modelos (*Model Driven Architecture*; MDA

en inglés; OMG, 2003), *estructural* porque representa elementos estáticos, sus relaciones y su organización, por otro lado, *dinámico* porque representa la interacción y los cambios en el tiempo de los elementos.

2.1. Notación

La notación de los esquemas preconceptuales se divide en cuatro grupos: nodos, relaciones, enlaces y aglutinadores (véase la Figura 2.2).

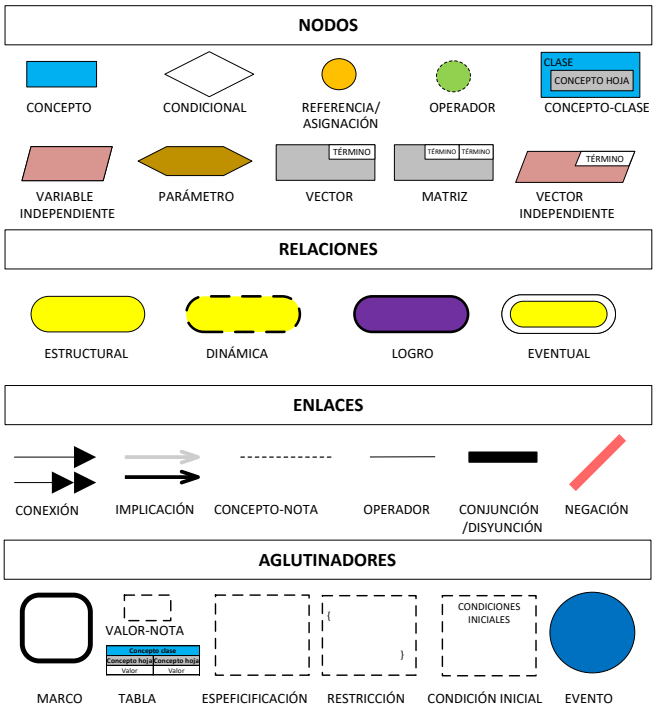


Figura 2.2. Notación de EP. (Zapata, 2012; Noreña-Cardona, 2020)

Ingresar al siguiente enlace para descargar plantillas de los EP y trabajar en herramientas de modelado.

Los colores estándar para representar la notación de los EP se presentan en la Figura 2.3.

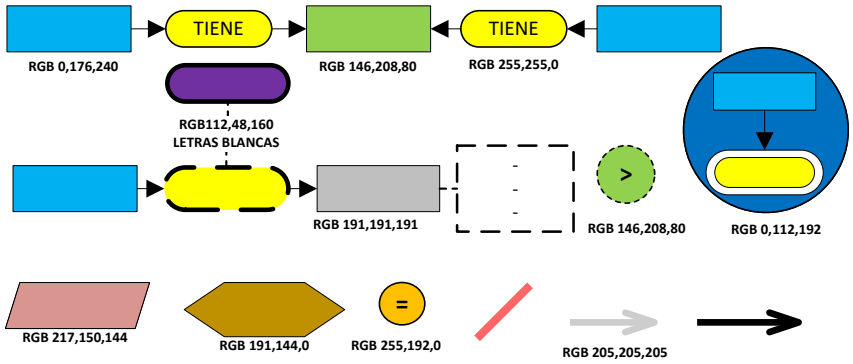


Figura 2.3. Colores estándar. Los autores

2.1.1. Nodos

Concepto: Se define como un sustantivo o un sintagma nominal en singular, que representa un actor u objeto dentro del dominio del interesado (Zapata, 2007, 2012). Los conceptos se pueden clasificar en dos tipos principales: (i) *conceptos clase*, aquellos conceptos principales que pueden tener atributos y relaciones y (ii) *conceptos hoja*, aquellos conceptos que se derivan de los conceptos clase como atributos. Por ejemplo, en la Figura 2.4, los conceptos "profesor", "gato" y "celular" ilustran los conceptos clase al estar en color azul. El "detalle" aparece en color verde, señalando que varias clases tienen este mismo concepto y, en consecuencia, se presenta como un tipo especial de concepto clase que es intermedio para los conceptos clase de donde proviene. Por otro lado, "nombre" se muestra en color gris, indicando que es un concepto hoja.

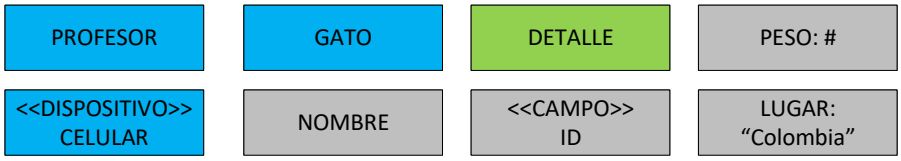


Figura 2.4. Concepto. Basado en Zapata (2007)

Los conceptos también pueden incluir *estereotipos* para proporcionar información

adicional. Por ejemplo, la clase «dispositivo» “celular” y el concepto hoja «campo» “id” en la Figura 2.4. Este tipo de anotaciones se emplea especialmente en representaciones que incluyen interfaces de usuario o en representaciones formales como ontologías y metamodelos (reglas y estructuras para modelos; Villamizar 2019).

Los *tipos de datos* para los conceptos hoja son: texto (sin símbolo), fecha(//), número (#), booleano (?) y correo (@), los cuatro últimos se definen después de usar el símbolo de dos puntos (Chaverra, 2011). Por ejemplo, el concepto hoja tipo *texto* “nombre” y el concepto hoja tipo *número* “peso” en la Figura 2.4.

Los conceptos hoja también permiten incorporar información como validaciones iniciales, especialmente en EP ejecutables (Este tipo de EP permiten probar y verificar la información del sistema). Por ejemplo, el concepto hoja “lugar” podría tener como valor “Colombia” en la Figura 2.4. Es importante anotar que el símbolo de asignación (=), se usa para indicar el valor del concepto hoja.

Concepto-clase: nodo para representar el acceso a un concepto hoja desde su concepto clase (Zapata, 2007). Por ejemplo, los conceptos clase en la Figura 2.5 para acceder a los conceptos hojas “especie” de la clase “tortuga” e “historia médica” de la clase “paciente”.

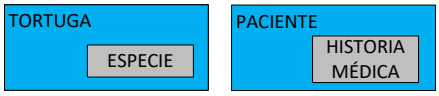


Figura 2.5. Concepto-clase. Basado en Manjarrés (2010)

Los arreglos *vector* y *matriz* son nodos que se clasifican como conceptos hoja cuando dependen de un concepto clase y se explican a continuación:

Vector: Arreglo dependiente con una dimensión, que se acompaña de un término para indicar el tamaño o posición del vector, también se conoce como arreglo unidimensional (Noreña-Cardona, 2020). Por ejemplo, la “estación temporal” y la “señal” de una clase “celular” que pueden tener diferentes valores en la Figura 2.6.

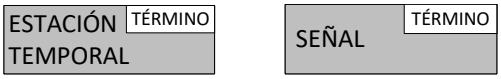


Figura 2.6. Vector. Basado en Noreña-Cardona (2020)

Matriz: Arreglo dependiente con dos dimensiones que se acompaña de dos términos para indicar la posición o el índice de la matriz, también se conoce como arreglo bidimensional (Noreña-Cardona, 2020). Por ejemplo, las matrices "coordenada" y "plano geográfico" para determinar una ubicación en la Figura 2.7.

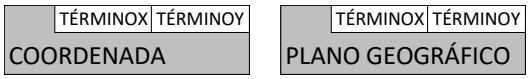


Figura 2.7. Matriz. Basado en Noreña-Cardona (2020)

Los nodos *variable independiente*, *parámetro* y *vector independiente* se consideran *conceptos independientes*, ya que no dependen de un concepto clase en el dominio. Estos se describen a continuación:

Variable independiente: Valor que no depende de otras variables y sirve para controlar las variables dependientes (conceptos hoja) de un dominio (Noreña-Cardona, 2020). Por ejemplo, la variable independiente "tiempo" es igual a "100 minutos", con la cual se puede controlar el resultado de una ecuación diferencial que depende del tiempo. La variable independiente "estado del sensor" es igual a "activo" también es un ejemplo para controlar un proceso automático que depende del "estado del sensor" (véase la Figura 2.8).

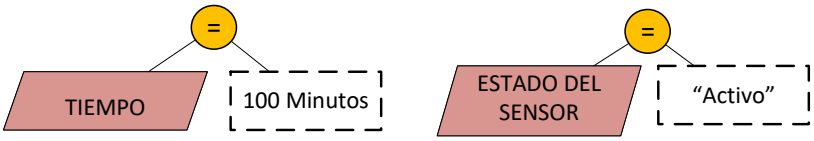


Figura 2.8. Variable independiente. Basado en Noreña-Cardona (2020)

Parámetro: Valor constante de un dominio, ecuación o función (Noreña-Cardona, 2020). Por ejemplo, el parámetro "tiempo de simulación" es "7 días"

para indicar que la simulación tarda “7 días” y el parámetro “PI”, que tiene un valor establecido “3,1416” en la Figura 2.9.

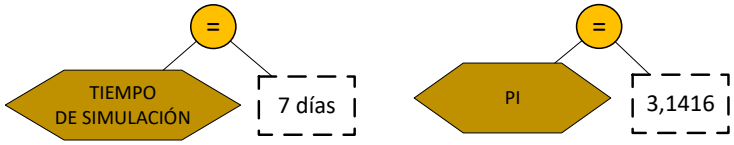


Figura 2.9. parámetro. Basado en Noreña-Cardona (2020)

Vector independiente: Cumple la misma función de las variables independientes, pero permite almacenar más de un valor (Noreña-Cardona, 2020). Por ejemplo, el vector “espacio” y el vector “recurso” en la Figura 2.10.

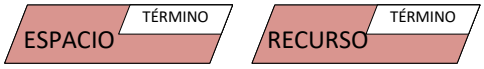


Figura 2.10. Vector independiente. Basado en Noreña-Cardona (2020)

Referencia/Asignación: La *referencia* sirve para relacionar un nodo distante; por ejemplo, la referencia “1” para el concepto “profesor” en la Figura 2.11 (a) indicando que un elemento distante en el dominio se relaciona con el “profesor”, en este caso, el otro elemento debe también tener la misma referencia. El código de colores no es obligatorio en la referencia, pero ayuda a localizar rápidamente los dos sitios que la conectan.

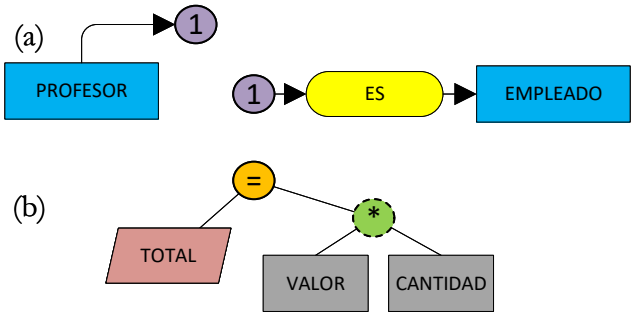


Figura 2.11. Referencia (a), Asignación (b). Basado en Zapata (2007) y Chaverra (2011)

Es importante señalar que la referencia se utiliza una sola vez por cada flecha, con el fin de evitar la proliferación de nodos espurios en la representación.

La *asignación* sirve para asignar el valor que resulta de una expresión matemática o lógica (Chaverra, 2011; Zapata, 2012); se debe representar con el signo de asignación "=", por ejemplo, el "total" es "=" al "valor" por la "cantidad" en la Figura 2.11 (b).

Operador: Símbolo lógico o matemático que sirve para formar expresiones a evaluar (Zapata, 2012). Los operadores pueden ser lógicos, aritméticos, relacionales, de funciones predefinidas (matemáticas y trigonométricas) y de arreglos (los operadores de arreglos *push* y *pop* sirven para agregar y disminuir una posición respectivamente) como se expresa en la Tabla 2.1 (Calle, 2016; Noreña-Cardona, 2020). No obstante, no se limita a los que están en la tabla, todos los operadores se pueden definir con este símbolo.

Tabla 2.1. Tipos de operadores. Basado en Calle (2016) y Noreña-Cardona (2020)

Operadores lógicos	AND	OR				
Operadores aritméticos	*	+	-	/	%	^
Operadores relacionales	=	!=	>	<	<=	>=
Operadores de funciones predefinidas (matemáticas Y trigonométricas)	Sqrt Raíz cuadrada	Exp Función exponencial	Log Función logarítmica	Abs Valor absoluto	Pow Potencia	Mod Módulo
	Sin Seno	Cos Coseno	Tan Tangente	Csc Cosecante	Ctg Cotangente	Sec Secante
Operadores de arreglos	Push	Pop				

Por ejemplo, para hallar un "resultado" en la Figura 2.12 (a) se debe operar la raíz cuadrada de "25", multiplicada por un logaritmo natural de la "magnitud" dividida por "2". Para el uso de la *función logarítmica* en diferentes bases, basta con adicionar un sufijo a lado del operador *Log*, correspondiente al número de la base y para el *logaritmo natural* la base equivale a 10. Para adicionar una posición al vector "valor" se utiliza el operador *push* en la Figura 2.12 (b).

El analista también puede definir *funciones*, con el propósito de un operador personalizado cuya especificación defina el comportamiento de una función, que se pueda reutilizar varias veces en todo el EP (Calle, 2016). Por ejemplo, la función "calcular potencial" en la Figura 2.13, primero se define y luego se utiliza para "calcular potencial" de "profundidad".

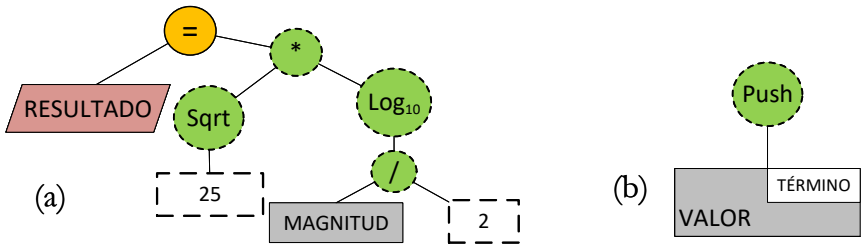


Figura 2.12. Ejemplo de operadores y funciones predefinidas (a), ejemplo de operador de arreglos (b). Basado en Calle (2016)

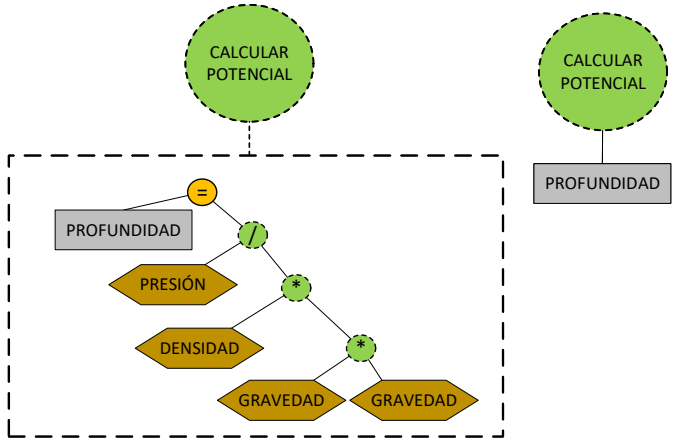


Figura 2.13. Función definida por el analista. Basado en Velásquez (2019)

Condicional: Condición para ejecutar una relación dinámica o una especificación a partir de una operación matemática formada por conceptos o variables, operadores y parámetros (Zapata, 2012; Noreña-Cardona, 2020). Por ejemplo, si se cumple el *condicional simple* (con una sola implicación) en la Figura 2.14, se puede ejecutar la relación dinámica "contador genera pago del empleado".

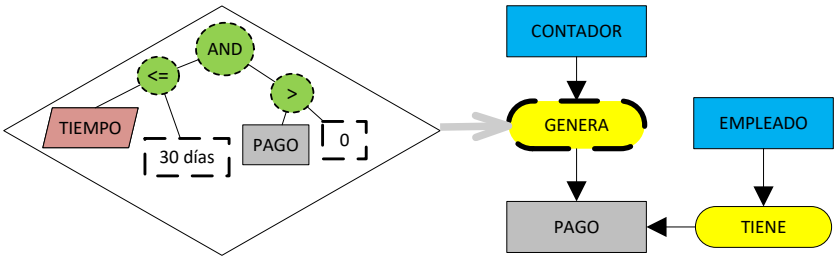


Figura 2.14. Condicional. Basado en Manjarrés (2010)

También, se puede representar el *condicional doble* con dos implicaciones (sí y no). Por ejemplo, si el "estado" es igual a "activo" entonces la puerta es igual a "abierta", sino entonces la puerta es igual a "cerrada" en la Figura 2.15.

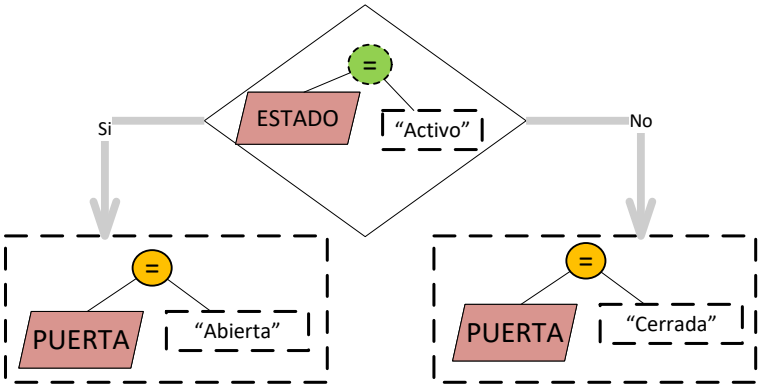


Figura 2.15. Condicional doble. Basado en Manjarrés (2010)

2.1.2. Relaciones

Relación Estructural (RE): Permite relacionar dos conceptos con los verbos *ser* o *tener* para indicar una generalización (herencia) y una agregación de clases,

respectivamente (Zapata, 2007, 2012). Por ejemplo, la clase “tortuga es animal”, es decir, que pertenece a otra clase y “tortuga tiene” un concepto hoja “nombre” en la Figura 2.16. Se utiliza la *conexión obligatoria* (doble) para indicar que el concepto hoja es obligatorio. Por ejemplo, “tortuga tiene especie” y “especie” es un concepto obligatorio en la Figura 2.16. Para un *concepto único*, se utiliza la relación de agregación con el verbo “tener” acompañado de “único” y del enlace de conexión obligatoria, que se conjuga en la triada de dos conceptos, por ejemplo, “tortuga tiene único identificador” en la Figura 2.16 (Chaverra, 2011).

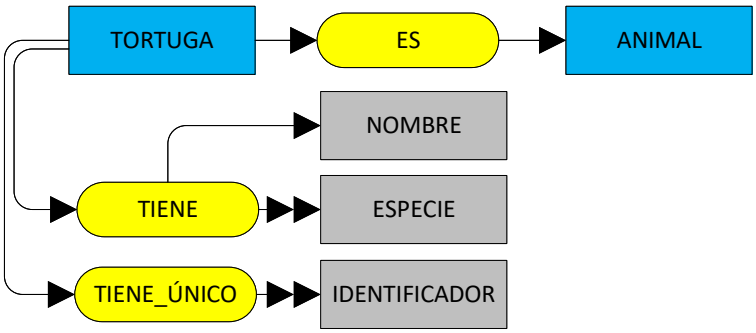


Figura 2.16. Ejemplo de relaciones estructurales. Basado en Zapata (2007)

Relación Dinámica (RD): Permite relacionar las acciones (procesos, servicios, actividades y operaciones) del sistema. Se relaciona entre el concepto que ejecuta la acción y el concepto objeto que recibe la acción (Zapata, 2007, 2012). Por ejemplo, “médico ingresa historia médica del paciente” en la Figura 2.17, siendo “ingresar” el verbo de la RD.

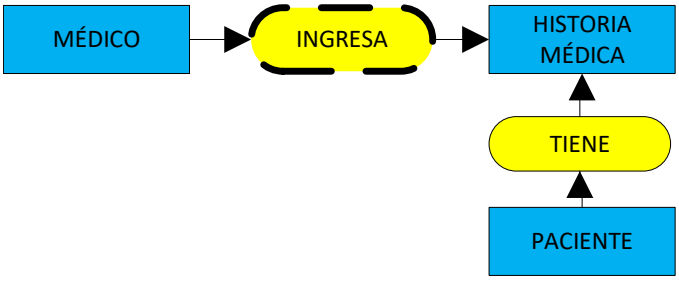


Figura 2.17. Ejemplo de relación dinámica. Basado en Zapata (2007)

Las RE y RD conforman una triada que se conjuga en tercera persona del singular.

Relación de Logro (RL): Permite relacionar los objetivos del dominio (Vargas, 2016). Los verbos en las relaciones de logro se deben escribir en infinitivo. Por ejemplo, el "estudiante presenta examen" con el objetivo de "ganar el examen" en la Figura 2.18. En este sentido, la relación de logro "ganar" se relaciona con el concepto "examen".

Figura 2.18. Ejemplo de relación de logro. Basado en Vargas (2016)

Relación Eventual (REv): Permite relacionar un evento/fenómeno o un proceso automático que pasa en el sistema. Esta relación se acompaña con cero a dos conceptos, siempre y cuando no sean actores, sino aquellos conceptos que reciben el evento o su efecto (Noreña-Cardona, 2020). Por ejemplo, los eventos "llueve", "alarma de sensor suena" y "epidemia llega a ciudad" en la Figura 2.19.

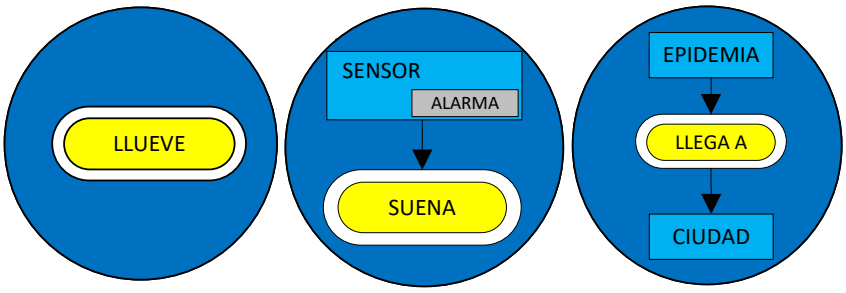


Figura 2.19. Ejemplo de relaciones eventuales. Traducido de Noreña-Cardona (2020)

2.1.3. Enlaces

Conexión: Flecha unidireccional que sirve para conectar conceptos con las RD, RE y REv (Zapata, 2007, 2012). Por ejemplo, el enlace *conexión* entre el concepto "estudiante" y la relación dinámica "presenta" en la Figura 2.18.

También se presenta la *conexión obligatoria* para generar una RE entre una clase y un concepto obligatorio (enlace doble). Por ejemplo, la conexión obligatoria en la triada "tortuga tiene especie" en la Figura 2.16.

Implicación: La *implicación (color gris)* es una línea continua y dirigida que indica una relación causa-efecto y el flujo entre RD, condicionales y eventos (Zapata, 2007, 2012). Por ejemplo, el flujo de implicaciones desde un condicional y el evento "Contador llega" que disparan la RD "contador aprueba pago" y en cuanto suceda esta acción, se finaliza el flujo con la RD "empleado retira pago" en la Figura 2.16.

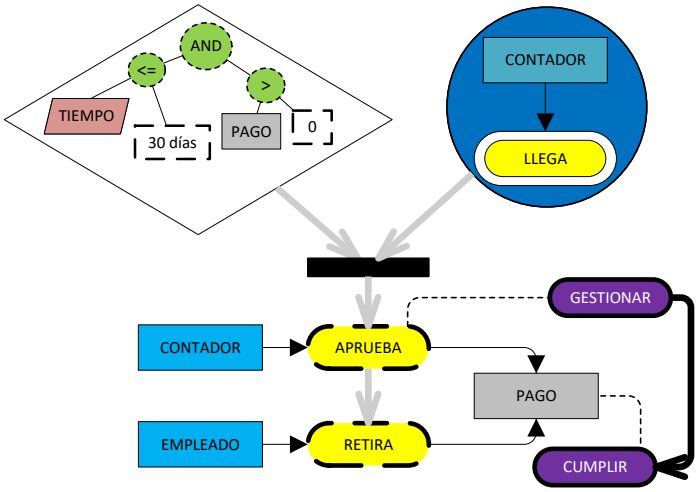


Figura 2.20. Ejemplo del enlace implicación. Basado en Zapata (2012)

La *implicación (color negro)* equivale a una relación causa-efecto entre relaciones de logro, por ejemplo, la implicación entre las relaciones de logro "gestionar la aprobación del pago" y "cumplir con el pago del empleado" en la Figura 2.20.

Concepto-Nota: Sirve para conectar un concepto a una relación de logro, un

valor-nota, una especificación o una restricción (Zapata, 2007, 2012; Vargas, 2016). Por ejemplo, el concepto hoja “especie” se conecta mediante un enlace a sus dos valores “terrestre” y “marina”, según la clasificación de la “especie” en la Figura 2.21.

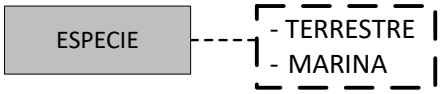


Figura 2.21. Ejemplo del enlace concepto-nota. Basado en Zapata (2007)

Operador: Enlace para conectar un valor, concepto, variable, parámetro, vector o matriz a una asignación, operador u otros operadores (Zapata, 2012; Noreña-Cardona, 2020). Por ejemplo, los elementos (operadores, conceptos y parámetros) dentro de la ecuación de la Figura 2.13, se conectan mediante el enlace *operador* para representar la función “calcular potencial” .

Conjunción/Disyunción: Sirve para agrupar o bifurcar implicaciones, estableciendo una causalidad conjunta o múltiples efectos (Zapata, 2007, 2012). Por ejemplo, la conjunción entre el condicional y el evento “contador llega” en la Figura 2.20.

Negación: Permite representar problemas del dominio mediante la negación de sus elementos: RL, RE y RD y la nota-valor (Vargas, 2016). Por ejemplo, los siguientes problemas: el “estudiante no gana el examen”, el cual se expresa negando la relación de logro “ganar” en la Figura 2.22; el “profesor no tiene disponibilidad”, el cual se representa negando la RE “tiene” en la Figura 2.23; el “profesor no envía examen”, el cual se ejemplifica negando la RD “envía” en la Figura 2.24; el “estado del examen no está disponible”, el cual se representa negando el valor-nota “disponible” del “estado” del “examen” en la Figura 2.25.

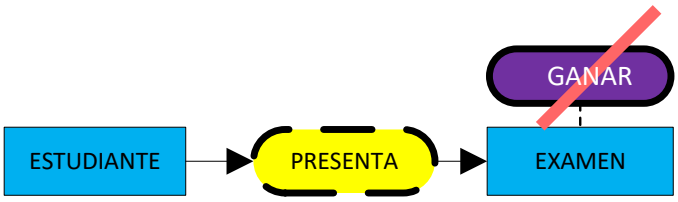


Figura 2.22. Ejemplo de problema en relación de logro con negación. Basado en Vargas (2016)

Figura 2.23. Ejemplo de problema en relación estructural con negación. Basado en Vargas (2016)

Figura 2.24. Ejemplo de problema en relación dinámica con negación. Basado en Vargas (2016)

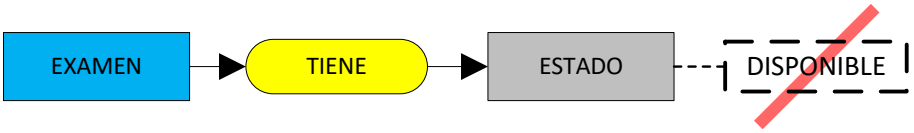


Figura 2.25. Ejemplo de problema en nota-valor con negación. Basado en Vargas (2016)

2.1.4. Aglutinadores

Marco: Sirve para asociar múltiples relaciones dinámicas a una responsabilidad o para agrupar conceptos. Su principal uso es la representación de reportes en el sistema (Zapata, 2012). Por ejemplo, el reporte de medición de contaminación de una zona en la ciudad en la Figura 2.26.

Valor-nota: Sirve para limitar los valores para un concepto a un conjunto predefinido (Zapata, 2007, 2012). Por ejemplo, el valor numérico “2” por el que se divide la “magnitud” en la Figura 2.27 o el valor-nota en el que se agrupa el valor en lista de la Figura 2.21.

Tabla: Se utilizan para simular valores que se almacenarán en una tabla/archivo de base de datos. En un EP ejecutable se puede ejemplificar la información del dominio (Zapata, 2012), permitiendo una visualización de la funcionalidad de las relaciones dinámicas o eventos con los valores de los atributos hoja mediante

tablas, es decir, que pueden aparecer tanto en los conceptos como en las tablas. Por ejemplo, la tabla de los permisos del usuario en Tabla 2.2.

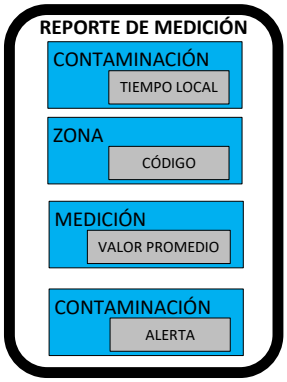


Figura 2.26. Ejemplo de marco. Basado en Noreña-Cardona (2020)

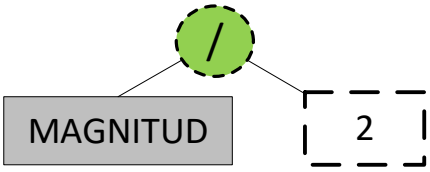


Figura 2.27. Ejemplo de valor-nota. Basado en Zapata (2007)

Tabla 2.2. Tabla de EP ejecutable. Basado en Zapata (2012)

PERMISO		
ID USUARIO	**ID**	**DESCRIPCION_DERECHO**
29168	1139	Admin
29168	1140	Gestiona albumes

Especificación: Sirve para agrupar un conjunto de operaciones que describen una relación dinámica, una relación eventual y una función (Zapata, 2012). Por ejemplo, la especificación que se utiliza para agrupar la operación de la función "calcular potencial" en la Figura 2.13. Cabe destacar que las especificaciones se pueden añadir a REs, RDs, REvs, RL y también a funciones personalizadas, enriqueciendo su definición. Asimismo, se debe destacar que únicamente se

permiten relaciones dinámicas atómicas en su interior, tales como: insertar, actualizar, borrar, buscar, seleccionar y listar.

Restricción: Sirve para establecer una condición sobre una especificación de operaciones (Zapata, 2012). Adicionalmente, se usa para establecer ciclos (Chaverra, 2011). Se puede relacionar con un concepto y con la relación dinámica, eventual (evento) y de logro. Por ejemplo, la restricción "presupuesto mayor o igual que subtotal" en la Figura 2.28.

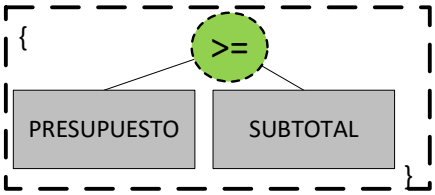


Figura 2.28. Ejemplo de restricción. Basado en Manjarrés (2010)

Condición inicial: Aglutinador que agrupa la asignación de valores iniciales de los conceptos independientes, es decir, parámetros, variables y vectores independientes (Noreña-Cardona, 2020). Por ejemplo, la variable "estado del sensor", el parámetro "tiempo de simulación" y el vector independiente "días" en la Figura 2.29.

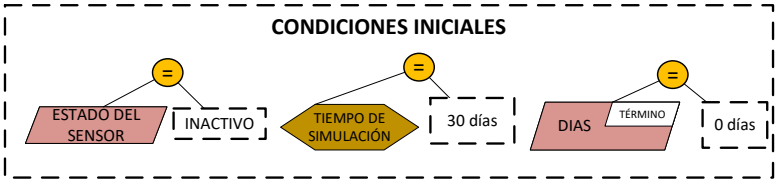


Figura 2.29. Condición inicial. Basado en Noreña-Cardona (2020)

Evento: Es un suceso que pasa en el sistema, se clasifica en evento disparador y evento de resultado.

El *evento disparador* genera el inicio de una RD o un flujo de RDs y de otros eventos. El evento disparador se clasifica en: (i) *tiempo*, cuando el evento es un tiempo o un ciclo de tiempos que activa el disparo, por ejemplo, el "tiempo pasa"; (ii) *declaración*, cuando es una instrucción que activa el disparo, por ejemplo, los eventos "llueve", "alarma de sensor suena" y "epidemia llega a

ciudad" en la Figura 2.19) y (iii) *condicional*, se considera un tipo de evento disparador cuando por una restricción se activa el disparo (Noreña-Cardona, 2020), por ejemplo, los condicionales de las Figuras 2.14 y 2.15.

El *evento de resultado* finaliza una cadena de implicaciones (Noreña-Cardona, 2014, 2020). Por ejemplo, el evento de resultado "registro creado" en la Figura 2.30, notése que el verbo se encuentra en participio pasado, que es usualmente el verbo de la relación dinámica que antecede al evento de resultado, por ejemplo, el verbo "crea" en la relación dinámica "usuario crea registro", cuyo participio pasado es "creado". Por defecto, el evento de resultado se puede omitir, ya que se sobreentiende que al final de toda cadena de implicaciones existe un evento de resultado implícito.

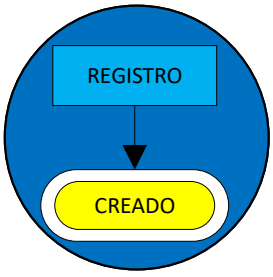


Figura 2.30. Ejemplo de evento de resultado. Basado en Noreña-Cardona (2014)

2.2. Características

Los EP tienen una naturaleza intuitiva (Zapata-Tamayo and Zapata-Jaramillo, 2018) que se basa en características estructurales, dinámicas, télicas y eventuales. Esas características se fundamentan en reglas lingüísticas y heurísticas utilizando los elementos de la notación para facilitar la equivalencia del EP a código fuente.

2.2.1. Características Estructurales

Las características estructurales se presentan como triadas entre conceptos y relaciones estructurales (véase la Figura 2.31). Cuyos conceptos son sustantivos, las relaciones son verbos *ser* (es) para generalización o herencia de clases y *tener*

(tiene) para definir una agregación entre clases y sus conceptos hoja, atributos. Esta relación puede incluir unicidad, es decir un *concepto único* con la relación estructural "tiene_único" (véase la Figura 2.16).

Figura 2.31. Relación estructural. (Zapata, 2007)

Los conceptos hoja en las RE pueden tener *valores-nota* para especificar sus valores en una lista, por ejemplo, los dos valores "terrestre" y "marina" del concepto hoja "especie" en la Figura 2.32. Estos conceptos también surgen como atributos derivados (cuyo valor se calcula a partir de otros valores y son conjuntos cerrados de valores; Chaverra, 2011). Un ejemplo es el área de un círculo en la Figura 2.33, al obtener el "radio del círculo" se deriva el valor del "área" a partir de su ecuación matemática, la cual se define usando una *restricción* en el concepto hoja "área".

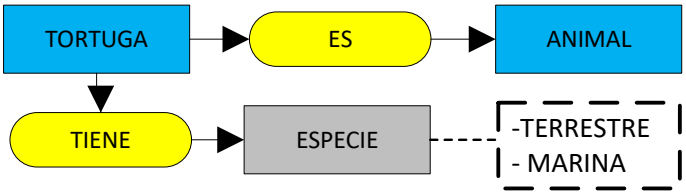


Figura 2.32. Ejemplo de relaciones estructurales y concepto con valores en lista. Basado en Zapata (2007)

Los conceptos pueden tener *restricciones* que se deben cumplir en el sistema. Por ejemplo, "edad de la persona" menor a "100" años en la Figura 2.34.

Las características estructurales también involucran la especificación de *condiciones iniciales* como se observa en la Figura 2.29 y la *especificación* de valores asignados a conceptos hoja o conceptos independientes. Por ejemplo, la asignación de los valores "abierta" o "cerrada" a la variable "puerta" a partir de una condición, se representa mediante una especificación en la Figura 2.15.

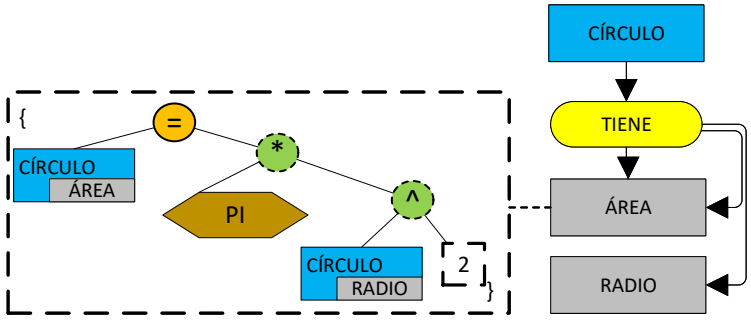


Figura 2.33. Ejemplo de atributo derivado. Basado en Calle (2016)

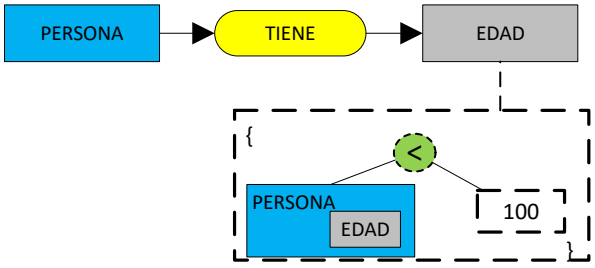


Figura 2.34. Ejemplo de concepto con restricción. Basado en Chaverra (2011)

2.2.2. Características Dinámicas

Las características dinámicas se presentan como triadas entre conceptos y relaciones dinámicas. Los conceptos son sustantivos o sintagmas nominales, siendo el primero un rol semántico (relación de un concepto en una frase o dominio) tipo actante (argumento que acompaña al verbo) *agente*, es decir, un actor que realiza una acción y el segundo concepto en el que recae la acción (véase la Figura 2.35; Zapata 2007).

Figura 2.35. Relación dinámica. (Zapata, 2007)

Las relaciones dinámicas también se pueden representar con una *especificación* para detallar la lógica interna de la funcionalidad, por ejemplo, la especificación

de la relación dinámica “experto químico registra mezcla” en la Figura 2.36. Las relaciones dinámicas y eventuales pueden incluir ciclos como el “subtotal de un producto” que se representa en la relación dinámica “vendedor registra venta”. Además, pueden incluir restricciones, por ejemplo, la restricción “presupuesto mayor o igual que subtotal” en la Figura 2.37. El aglutinador *restricción* también se puede utilizar en casos donde se presentan condiciones para ejecutar la relación dinámica.

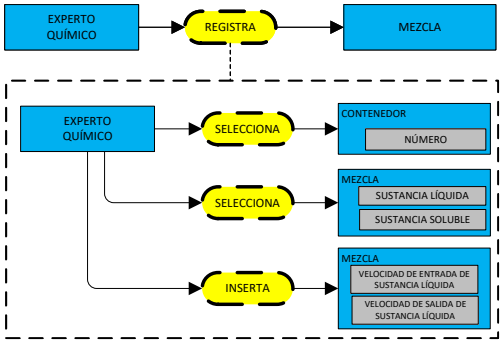


Figura 2.36. Ejemplo de relación dinámica con especificación. Basado en Noreña-Cardona (2020)

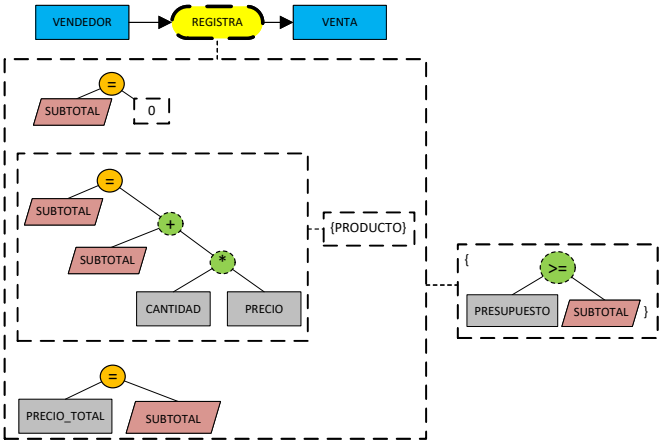


Figura 2.37. Ejemplo de relación dinámica con restricción. Basado en Chaverra (2011)

Los ciclos también se pueden representar con una *restricción con concepto* dentro

de la lógica de una relación dinámica o una relación eventual, cuando se requiere que se realice las operaciones de la especificación para todos los conceptos que contengan el mismo nombre del concepto. Por ejemplo, en la relación dinámica "profesor califica examen" en la Figura 2.38, el *ciclo* del concepto "examen" incluye, a su vez, el *ciclo* del concepto "pregunta". Este nivel de especificación permite desglosar la calificación como la suma de los puntajes que se asignan a cada respuesta correcta, considerando los porcentajes para cada pregunta (Chaverra, 2011).

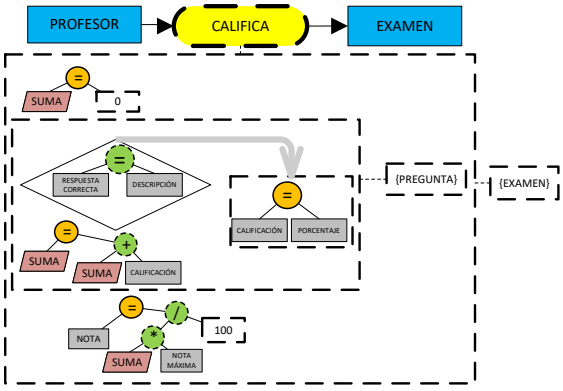


Figura 2.38. Restricción con concepto. (Chaverra, 2011)

Los *EP ejecutables* son aquellos que permiten representar datos de los conceptos hoja con el fin de generar validaciones iniciales a los datos de una ejecución en relaciones dinámicas y eventuales (Zapata, 2012).

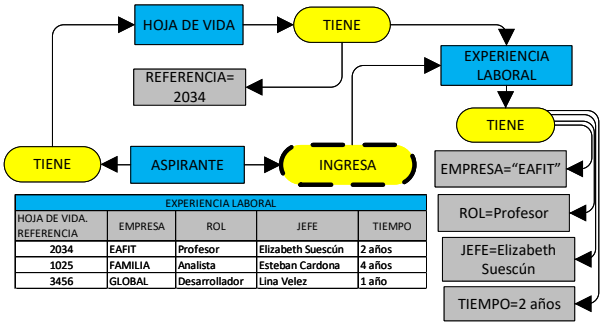


Figura 2.39. Ejemplo de EP ejecutable. Basado en Zapata (2012)

2.2.3. Características Télicas

Las características télicas se utilizan para representar *objetivos* y *problemas* del sistema.

Objetivos: Usualmente esta representación se realiza entre un concepto y una relación de logro como se observa en la Figura 2.40. Un conjunto de verbos para relaciones de logro se sugiere en la Tabla 2.3, que pueden opcionalmente llevar un adverbio para completar el significado.

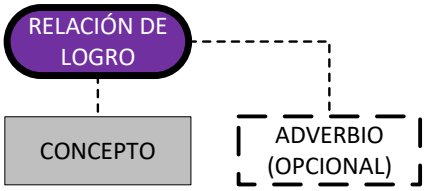


Figura 2.40. Relación de logro en concepto. Basado en Vargas (2016)

Tabla 2.3. Ejemplo de Verbos para relaciones de logro. Los autores

Mejoramiento	Mantenimiento	Logro
1. Mejorar	13. Evitar	25. Conocer
2. Aumentar	14. Asegurar	26. Hacer
3. Reducir	15. Conservar	27. Desarrollar
4. Fomentar	16. Mantener	28. Lograr
5. Promover	17. Gestionar	29. Adquirir
6. Disminuir	18. Garantizar	30. Traer
7. Avanzar	19. Obtener	31. Resolver
8. Aumentar	20. Controlar	
9. Ganar	21. Proteger	
10. Ayudar	22. Cuidar	
11. Progresar	23. Proteger	
12. Refinar	24. Dar	

Un conjunto de adverbios se incluye en la Tabla 2.4, los cuales se definen a partir de Vargas, 2016).

No obstante, los verbos y adverbios que se sugieren no se limitan a los que están en estas Tablas.

Las relaciones de logro también se pueden vincular con relaciones dinámicas y estructurales para expresar objetivos como se presentan en las Figura 2.41, Figura 2.42, y Figura 2.43.

Tabla 2.4. Adverbios para objetivos y problemas. Basada en Vargas (2016)

Adverbios		
Eficientemente	Limitadamente	Rápidamente
Adecuadamente	Ineficientemente	Eficazmente
Inadecuadamente	Tarde	Ineficazmente
A tiempo	Lentamente	Correctamente
A destiempo	Fuera de tiempo	Parcialmente

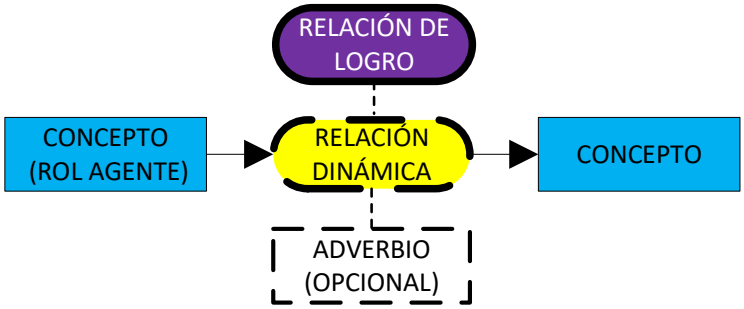


Figura 2.41. Relación de logro en relación dinámica. Basado en Vargas (2016)

Figura 2.42. Relación de logro en relación estructural. Basado en Vargas (2016)

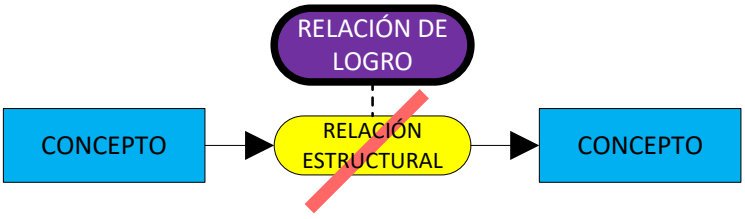


Figura 2.43. Relación de logro en relación estructural con negación. Basado en Vargas (2016)

Como ejemplos:

El objetivo “ganar el examen” se expresa en la Figura 2.44 mediante la relación de logro “ganar” en el concepto “examen”. Este mismo objetivo puede incluir un adverbio en el concepto para completar el significado; por ejemplo, “ganar el examen parcialmente” en la Figura 2.45.

Figura 2.44. Ejemplo de relación de logro en concepto. Basado en Vargas (2016)

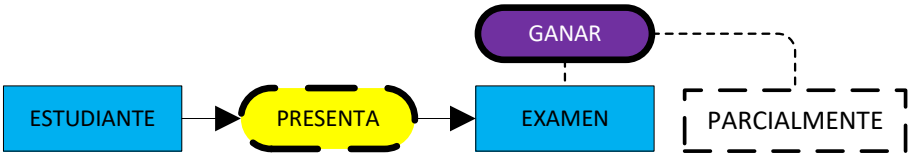


Figura 2.45. Ejemplo de relación de logro en concepto con adverbio. Basado en Vargas (2016)

El requisito “garantizar que el profesor envíe el examen” se representa en la Figura 2.46 mediante la relación de logro “garantizar” en la relación dinámica “envía”. También se puede incluir un adverbio en la relación dinámica para completar el significado como “garantizar que el profesor envíe el examen a tiempo” en la Figura 2.47.

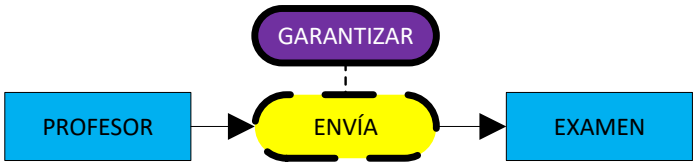


Figura 2.46. Ejemplo de relación de logro en relación dinámica. Basado en Vargas (2016)

El objetivo “lograr que el examen no tenga errores” se expresa en la Figura 2.48

mediante la relación de logro “lograr” en la relación estructural “tiene” que se representa con el símbolo de negación.

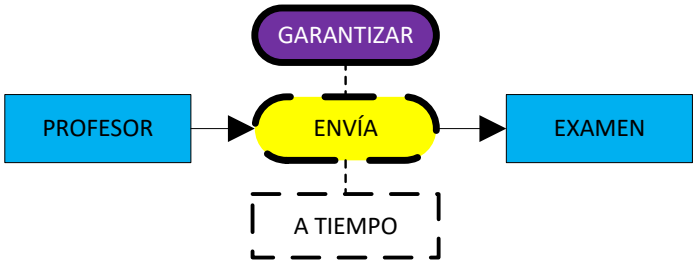


Figura 2.47. Ejemplo de relación de logro en relación dinámica con adverbio. Basado en Vargas (2016)

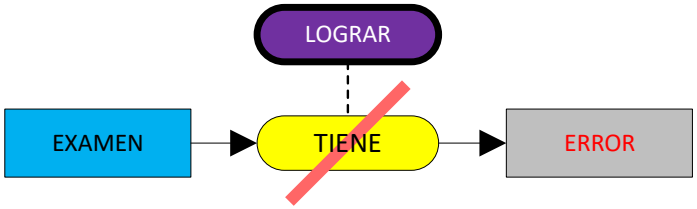


Figura 2.48. Ejemplo de relación de logro en relación estructural con negación. Basado en Vargas (2016)

Las relaciones télicas para representar objetivos se pueden detallar utilizando una *restricción* o *especificación* para medir indicadores de rendimiento clave (KPI), que permitan observar el cumplimiento del objetivo (Zapata y Castro 2017). Por ejemplo, el objetivo “mejorar paciente” que se detalla midiendo el indicador “promedio del ritmo cardíaco” en la Figura 2.49.

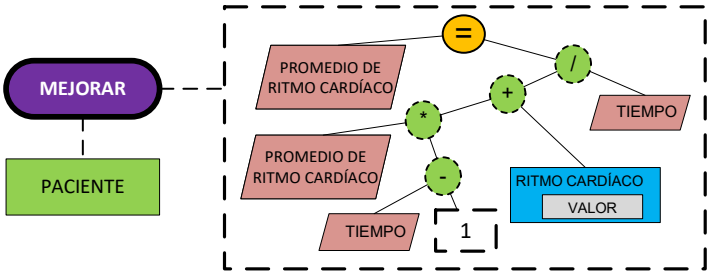


Figura 2.49. Ejemplo de objetivo con restricción. Los autores

Problemas: Las características télicas se expresan también para representar problemas del dominio, utilizando el elemento *negación*. Elementos como: relaciones de logro, estructural y dinámica y la nota-valor se pueden relacionar con problemas mediante el símbolo negación (véanse las Figuras 2.50, 2.51, 2.52 y 2.53 respectivamente; Vargas, 2016).

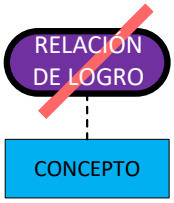


Figura 2.50. Problema en relación de logro con negación. Basado en Vargas (2016)

Figura 2.51. Problema en relación estructural con negación. Basado en Vargas (2016)

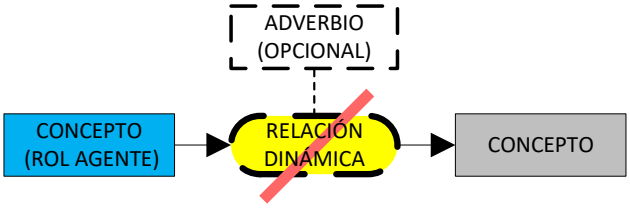


Figura 2.52. Problema en relación dinámica con negación. Basado en Vargas (2016)

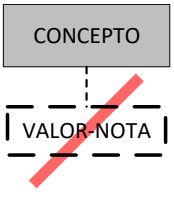


Figura 2.53. Problema en valor nota con negación. Basado en Vargas (2016)

Los problemas también se pueden representar sin el símbolo “negación” cuando se utilizan adverbios (véase la Tabla 2.4) o conceptos en una connotación negativa (véanse las Figuras 2.54, 2.55 y 2.56). Las palabras con connotación negativa se representan en color rojo, ya sean conceptos, relaciones, adjetivos o adverbios.

Figura 2.54. Problema usando valor negativo. Basado en Vargas (2016)

Figura 2.55. Problema usando un concepto negativo en una relación estructural. Basado en Vargas (2016)

Figura 2.56. Problema usando adverbio negativo en una relación dinámica. Basado en Vargas (2016)

Como ejemplos:

El problema “estudiante no gana el examen” en la Figura 2.22 se representa negando la relación de logro “ganar” relacionada al concepto “examen”; el problema “profesor no tiene disponibilidad” en la Figura 2.23 se representa negando la relación estructural “tiene”; el problema “profesor no envía examen” en la Figura 2.24 se representa negando la relación dinámica “envía”; el problema “estado del examen no está disponible” en la Figura 2.25 se representa negando el valor nota “disponible” correspondiente a un adjetivo.

El problema el “profesor no genera eficientemente el examen” en la Figura 2.57 se representa negando la relación dinámica “genera” con el adverbio “eficientemente”.

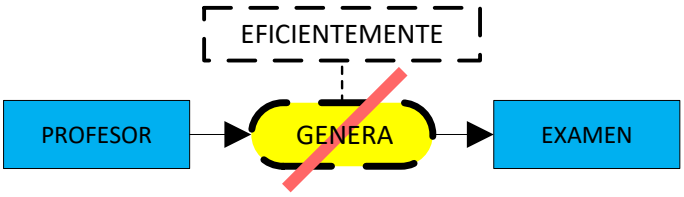


Figura 2.57. Ejemplo de problema en relación dinámica con negación y adverbio. Basado en Vargas (2016)

El problema “examen tiene un resultado deficiente” en la Figura 2.58 se representa con un valor negativo “deficiente” como posible valor del concepto “resultado”.

Figura 2.58. Ejemplo de problema usando un valor negativo con un concepto. Basado en Vargas (2016)

El problema “examen tiene demora” en la Figura 2.59 se representa usando el concepto negativo “demora” con la relación estructural “tiene”.

Figura 2.59. Ejemplo de problema usando un concepto negativo con una relación estructural. Basado en Vargas (2016)

El problema “profesor entrega inadecuadamente el examen” en la Figura 2.60 se representa usando el adverbio negativo “inadecuadamente” con la relación dinámica “entrega”.

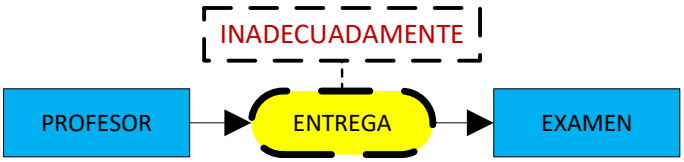


Figura 2.60. Ejemplo de problema usando adverbio negativo con una relación dinámica. Basado en Vargas (2016)

2.2.4. Características Eventuales

Las características eventuales se expresan entre cero y dos conceptos y una relación eventual, para representar fenómenos y procesos automáticos, que se presentan en forma de *eventos disparadores* y *eventos de resultado*.

Eventos disparadores: En la Tabla 2.5 se realiza un compendio de verbos para relaciones eventuales en eventos disparadores (Noreña-Cardona, 2020).

Los eventos disparadores se pueden representar de tres maneras:

(i) *Evento sin concepto*, son aquellos que el mismo verbo determina un significado completo para expresar lo que pasa (véase la Figura 2.61), usualmente corresponden a fenómenos naturales. Las relaciones eventuales que pertenecen a esta categoría son los verbos del 1 al 4 en la Tabla 2.5 y sus sinónimos.

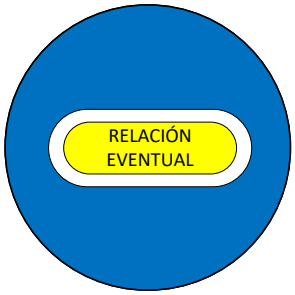


Figura 2.61. Evento sin concepto. Basado en Noreña-Cardona (2020)

(ii) *Evento con un concepto*, aquellos que requieren un actante (*experimentador*, aquel que experimenta el evento o *paciente*, aquel que recibe los efectos del

evento) para expresar el significado del evento, el cual se puede representar mediante un concepto, un concepto-clase o una variable independiente con una relación eventual (véase la Figura 2.62). Las relaciones eventuales que pertenecen a esta categoría son los verbos 5 al 31 de la Tabla 2.5 y sus sinónimos.

Tabla 2.5. Verbos para relaciones eventuales. (Noreña-Cardona, 2020)

Relación eventual	Ejemplo	Relación eventual	Ejemplo
1. Llover	Llueve	20. Llegar	Onda llega
2. Tronar	Truena	21. Emerger	Bacteria emerge
3. Granizar	Graniza	22. Venir/acercar	Señal viene
4. Nevar	Nieva	23. Aparecer	Onda eléctrica aparece
5. Subir	Precio sube	24. Surgir	Nuevo pago surge
6. Incrementar	Presión sanguínea incrementa	25. Erupcionar	Volcán erupciona
7. Crecer/nacer /germinar/ brotar	Población crece	26. Hervir	Agua hierve
8. Disminuir	Precio del dólar disminuye	27. Expirar/ caducar	Producto expira
9. Bajar	Venta baja	28. Iniciar/ empezar	Servicio inicia
10. Estornudar	Paciente estornuda	29. Pasar	Tiempo pasa
11. Sangrar	Paciente sangra	30. Suceder	Huracán sucede
12. Convulsionar	Paciente convulsiona	31. Ocurrir	Terremoto ocurre
13. Morir	Animal muere	32. Cambiar	Temperatura cambia
14. Dormir	Paciente duerme	33. Sufrir	Paciente sufre hemorragia
15. Tintinear	Campana tintinea	34. Presentar	Paciente presenta dolor
16. Sonar	Truena suena	35. Bloquear	Lípido bloquea vena
17. Timbrar	Alarma timbra	36. Perder	Paciente pierde peso
18. Volar	Abeja africana vuela	37. Ganar	Paciente gana peso
19. Caer	Rayo cae	38. Derretir	Lava derrite roca

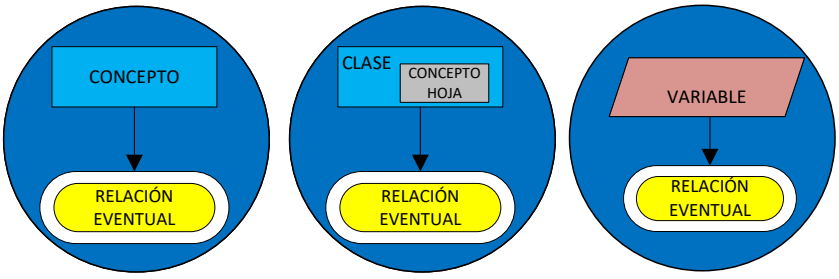


Figura 2.62. Evento con un concepto, concepto-clase y variable. Basado en Noreña-Cardona (2020)

(ii) *Evento con dos conceptos*, aquellos que requieren dos actantes para completar el significado del evento, por lo que se representa mediante dos conceptos y una relación eventual (véase la Figura 2.63). Las relaciones eventuales que pertenecen a esta categoría son los verbos 32 al 38 de la Tabla 2.5 y sus sinónimos. Además, verbos como llegar, aparecer, ocurrir, pasar, entre otros; que se encuentran en la categoría "eventos con un concepto" se puede también representar con dos conceptos al adicionar una preposición ("a", "desde", "en", "por", etc.). Por ejemplo, "llega a", "aparece en" y "viene desde".

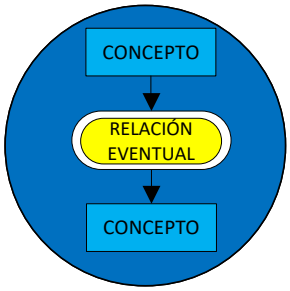


Figura 2.63. Evento con dos conceptos. Basado en Noreña-Cardona (2020)

Como ejemplos:

El evento "llueve" en la Figura 2.19 se representa con una relación eventual sin un concepto. El evento "alarma de sensor suena" se representa mediante un concepto-clase y una relación eventual en la Figura 2.19, este evento también es un ejemplo de un proceso automático. El evento "tiempo llega" se representa mediante una variable y una relación eventual en la Figura 2.64.

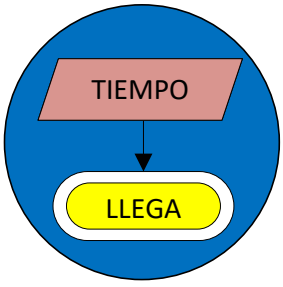


Figura 2.64. Ejemplo de evento con una variable. Basado en Noreña-Cardona (2020)

El evento “epidemia llega a ciudad” se representa con dos conceptos y una relación eventual, utilizando el verbo “llega” acompañado de la preposición “a” en la Figura 2.19.

Eventos de resultado: Los eventos de resultado se pueden representar como *eventos con un concepto*, como se presenta en la Figura 2.62, pero la relación eventual debe incluir el mismo verbo de la relación dinámica conjugado en participio pasado (Zapata, 2012). Por ejemplo, el evento de resultado “registro creado” de la Figura 2.30, el cual tiene un concepto “registro” y el verbo de la relación eventual “creado”, que se encuentra en participio pasado. Por defecto, el evento de resultado se puede omitir, ya que al final de una cadena de implicaciones existe uno implícito. Sin embargo, si el verbo en participio pasado es diferente al de la relación dinámica se debe incluir de forma explícita.

Los eventos disparadores y de resultado pueden detallar la lógica interna de la funcionalidad mediante una *restricción* o una *especificación*. Un ejemplo de evento con restricción se presenta para el evento disparador “mezcla inicia” en la Figura 2.65.

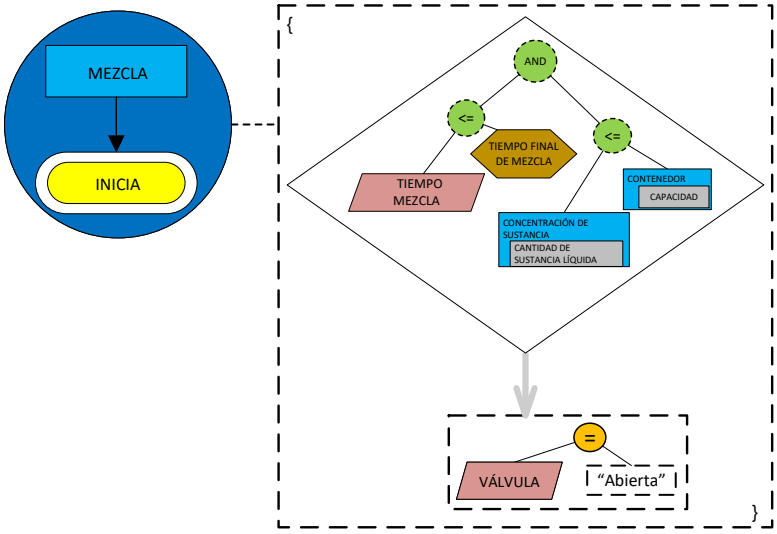


Figura 2.65. Ejemplo de evento con restricción. (Noreña-Cardona et al., 2019)

Capítulo 3

Antecedentes

3.1. Revisión de literatura

3.1.1. Especificación de las características de los EP

Características Estructurales: Zapata (2007) realiza la primera contribución en EP proponiendo relaciones estructurales (es, tiene), conceptos clase y conceptos hoja de acuerdo con el paradigma orientado a objetos (POO, programación usando clases y objetos) en el que se basan las equivalencias a diagramas UML (Lenguaje Unificado de Modelado por sus siglas en inglés) y sus vistas *estructural* (estática, conceptos y relaciones) y *dinámica* (comportamental, interacción entre objetos). En este sentido, Chaverra (2011) extiende la representación de conceptos en EP, según los tipos de datos para ambientes de desarrollo (véase la Tabla 3.1) y propone la representación de atributos derivados. Mientras que, Manjarrés (2010) resuelve la necesidad de incluir restricciones en conceptos hojas para su especificación.

Tabla 3.1. Tipos de datos. Basado en Chaverra (2011)

Tipo de dato	Representación	Ejemplo
Texto	CONCEPTO	NOMBRE
Fecha	CONCEPTO://	FECHA_VENTA://
Número	CONCEPTO:#	CANTIDAD: #
Booleano	CONCEPTO:?	ACTIVO: ?
Email	CONCEPTO:@	EMAIL: @

Chaverra (2011) también propone la representación de conceptos obligatorios

que se utilizan en desarrollo de software. Por ejemplo, en la Figura 3.1, “B” es un *concepto obligatorio* de “A”.

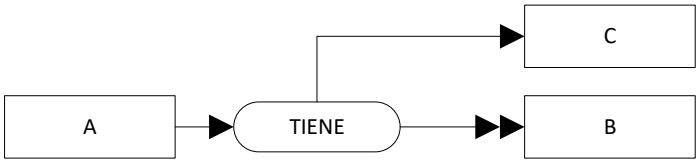


Figura 3.1. Concepto obligatorio. (Chaverra, 2011)

Características Dinámicas: Zapata (2007) crea las relaciones dinámicas en el EP con el propósito de generar la vista comportamental del dominio mediante procesos del sistema. Chaverra (2011) adiciona a esta vista la especificación de las relaciones dinámicas, para representar operaciones matemáticas que intervienen en el proceso (véase la Figura 3.2), operaciones que requieren una *restricción* (véase la Figura 3.3) y operaciones con ciclos en su funcionalidad (véase la Figura 3.4) .

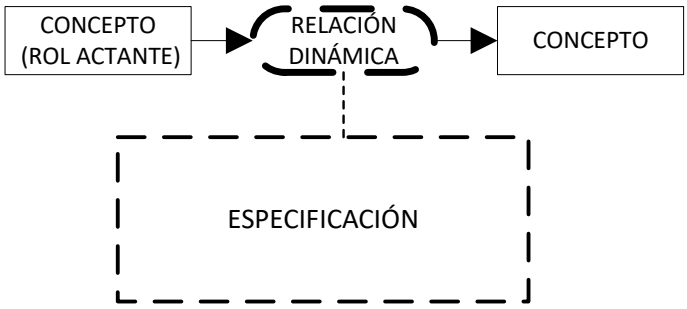


Figura 3.2. Especificación de las relaciones dinámicas. (Chaverra, 2011)

Zapata *et al.* (2011a) proponen los EP como esquemas ejecutables, es decir aquellos que permiten observar los valores que se registran en la base de datos después de la ejecución de una relación dinámica; por ejemplo, la relación dinámica “usuario crea álbum” en la Figura 3.5.

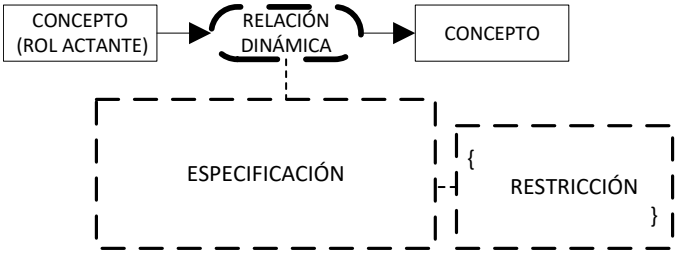


Figura 3.3. Especificación de restricción en relaciones dinámicas. (Chaverra, 2011)

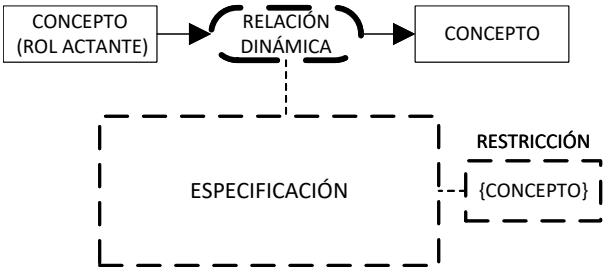
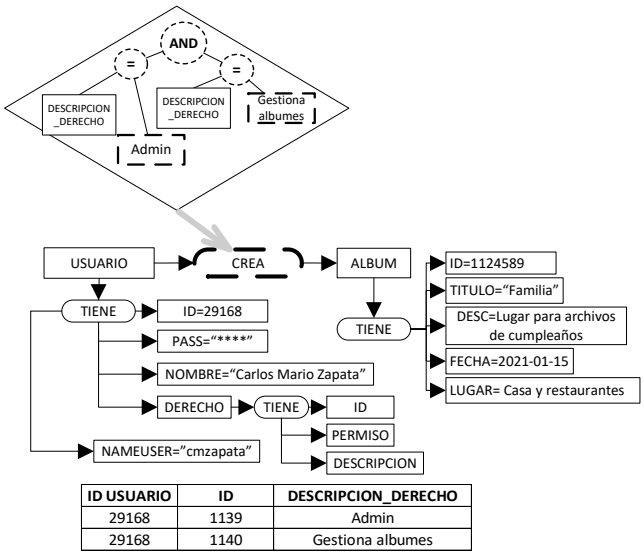


Figura 3.4. Especificación de un ciclo usando concepto. (Chaverra, 2011)

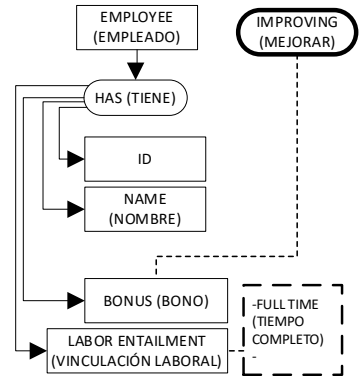
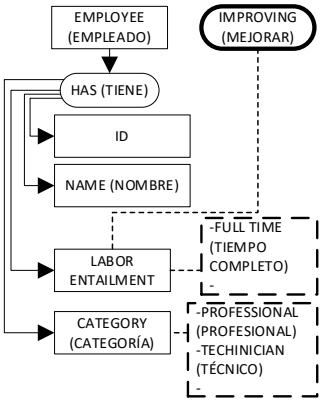


ID USUARIO	ID	DESCRIPCION_DERECHO
29168	1139	Admin
29168	1140	Gestiona albumes

Figura 3.5. EP ejecutable. Traducido de Zapata (2012)

Características Télicas: Zapata *et al.* (2007a) introducen las relaciones de logro para representar los objetivos del proyecto. Además, Vargas (2016) propone algunos casos para modelar estas relaciones en el contexto de los objetivos y problemas del proyecto. Asimismo, Zapata y Castro (2017) presentan patrones de objetivos basados en EP, incorporando indicadores KPI (indicadores de rendimiento clave como parte de la especificación de las relaciones de logro, por ejemplo, *"improving bonus of employee"* (mejorar el bono del empleado) e *"improving labor entailment of employee"* (mejorar la vinculación laboral empleado) en la Tabla 3.2.

Tabla 3.2. Patrones de objetivos en EP mediante KPI. (Zapata y Castro, 2017)

Objetivos EP	KPI
	sum of BONUSES of EMPLOYEES with LABOR ENTAILMENT « FULL TIME » maximum BONUS of EMPLOYEE with LABOR ENTAILMENT « FULL TIME » minimum BONUS of EMPLOYEE with LABOR ENTAILMENT « FULL TIME » average BONUS per EMPLOYEE with LABOR ENTAILMENT « FULL TIME »
	number of EMPLOYEES with LABOR ENTAILMENT «FULL TIME» which have CATEGORY « PROFESSIONAL» number of EMPLOYEES with LABOR ENTAILMENT «FULL TIME» which have CATEGORY «TECHNICIAN»

Características Eventuales: Zapata (2012) incluye una representación gráfica de los eventos del dominio del sistema para la representación de procesos automáticos. Noreña *et al.* (2019) y Noreña-Cardona (2020) extienden la notación matemática de los EP para representar condiciones y operaciones complejas en dominios de software científico y proponen la especificación/restricción de eventos, para incluir estás operaciones desde procesos automáticos y fenómenos naturales (véase la Figura 3.6).

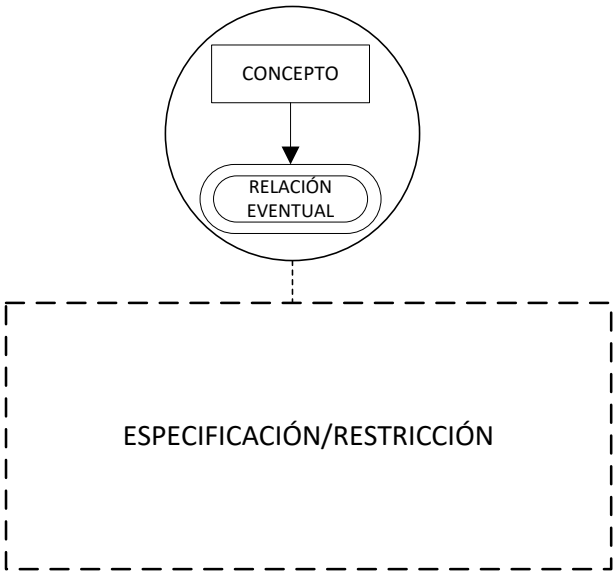
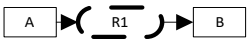
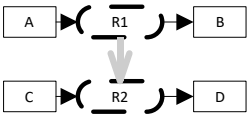
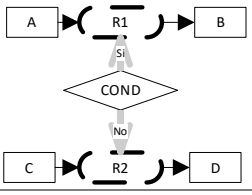


Figura 3.6. Especificación de relaciones eventuales. (Noreña-Cardona, 2020)

3.1.2. Equivalencias de los EP

Equivalencias de EP a esquemas conceptuales: Zapata (2007, 2012) define la equivalencia de relaciones estructurales y dinámicas desde el Lenguaje natural controlado (CNL, por sus siglas en inglés; un lenguaje natural con reglas estrictas de grámatica), permitiendo dar consistencia entre el discurso del interesado y la representación del dominio (véase la Tabla 3.3).

Tabla 3.3. Equivalencia de EP A CNL. (Zapata, 2007)

Notación formal	Expresión en CNL	EP
A<ES>B	A *es unA especie de* B A *es un tipo de* B A *es una clase de* B	
A<TIENE>B	A *incluye* B A *contiene* B A *posee* B A *esta compuesto por* B A *está formado por* B A *se divide en* B	
A<R1>B	<R1> puede ser cualquier verbo dinámico, por ejemplo: A *registra* B, A *paga* B	
C<R2>D, <SI>A<R1>B	*Si* A <R1> B *entonces* C<R2>D *Dado que* A <R1>B, C<R2>D *Luego de que* A <R1>B, C<R2>D	
<SI>{COND} <ENTONCES> A<R1>B, <SINO> C<R2>D	{COND} es una condición expresada en términos de conceptos. <R1> y <R2> son verbos dinámicos. <SINO>es opcional, por ejemplo: *si* M *es mayor que 100 entonces* A *registra* B	

Zapata (2007, 2012) también propone reglas heurísticas de equivalencia desde la notación de EP a esquemas conceptuales UML (OMG, 2011), como:

(i) El *diagrama de clases*, una vista estructural que representa las clases del sistema, sus atributos, operaciones y relaciones entre ellas: asociación, generalización y agregación (véase la Tabla 3.4).

(ii) El *diagrama de comunicaciones*, una vista dinámica que describe cómo interactúan los objetos y actores del sistema mediante mensajes (véase la Tabla 3.5).

(iii) El *diagrama de máquina de estados*, una representación de los posibles estados de un objeto a lo largo de su ciclo de vida, y las transiciones entre ellos (véase la Tabla 3.6).

(iv) El *diagrama de secuencia*, que proporciona una vista cronológica de la interacción entre objetos, mostrando el intercambio de mensajes, véase la Tabla 3.7, de acuerdo con las reglas que proponen Zapata y Garcés (2008).

(v) El *diagrama de casos de uso*, que permite describir los requisitos funcionales del sistema desde el punto de vista del usuario, sus relaciones de asociación y de dependencias inclusión (*include*) y extensión (*extend*), véase la Tabla 3.8, según Zapata *et al.* (2007b).

Tabla 3.4. Equivalencia de EP al diagrama de clases. (Zapata, 2007)

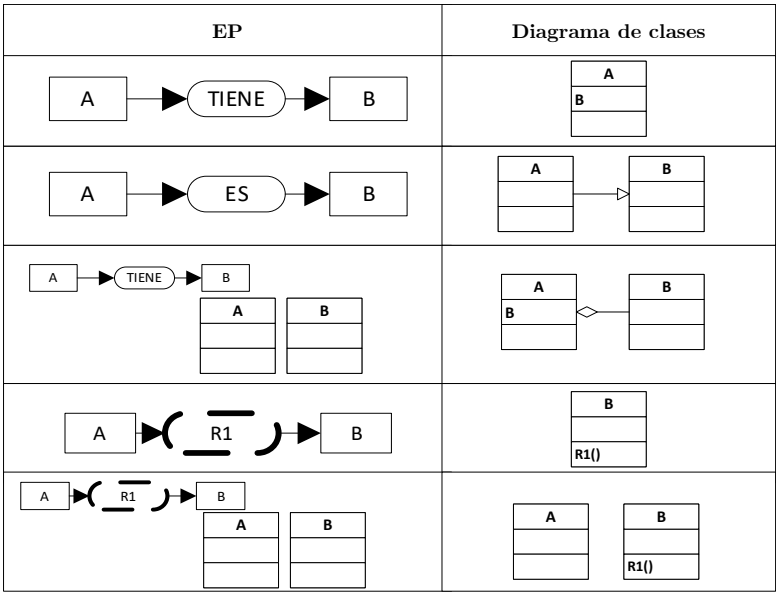


Tabla 3.5. Equivalencia de EP al diagrama de comunicaciones. (Zapata, 2007)

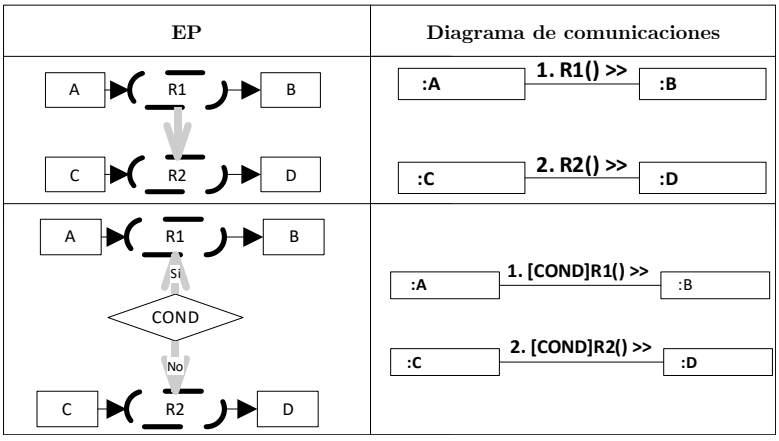


Tabla 3.6. Equivalencia de EP al diagrama de máquina de estados. (Zapata, 2007)

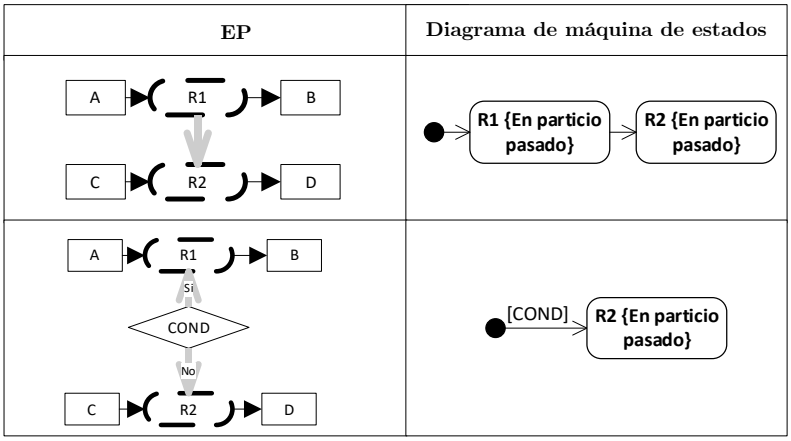


Tabla 3.7. Equivalencia de EP al diagrama de secuencia. (Zapata y Garcés, 2008)

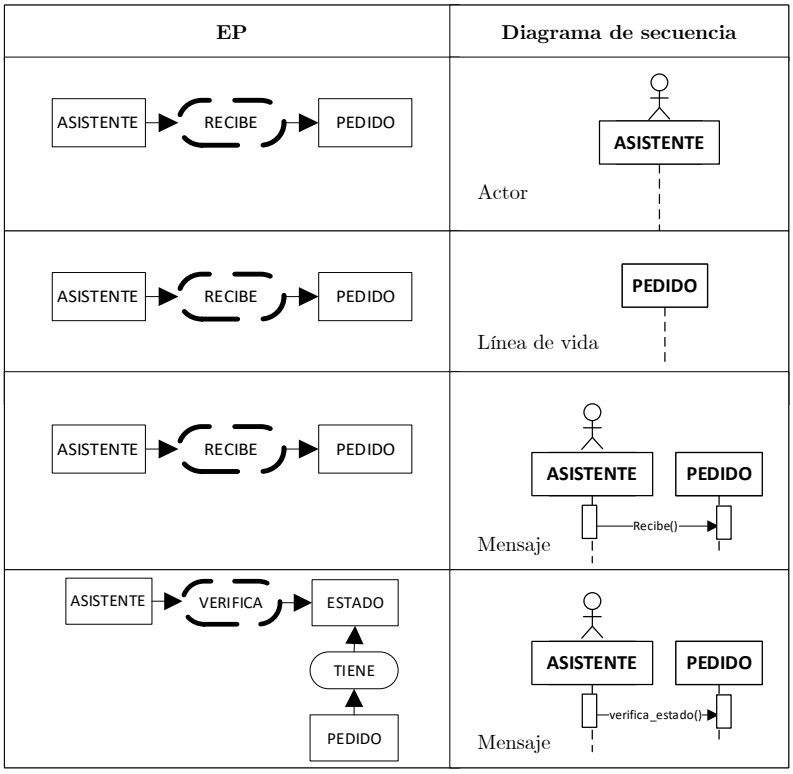
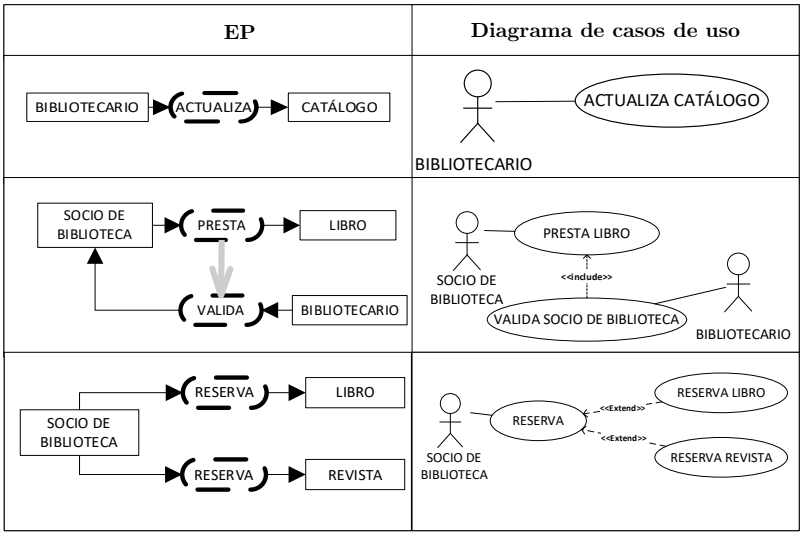
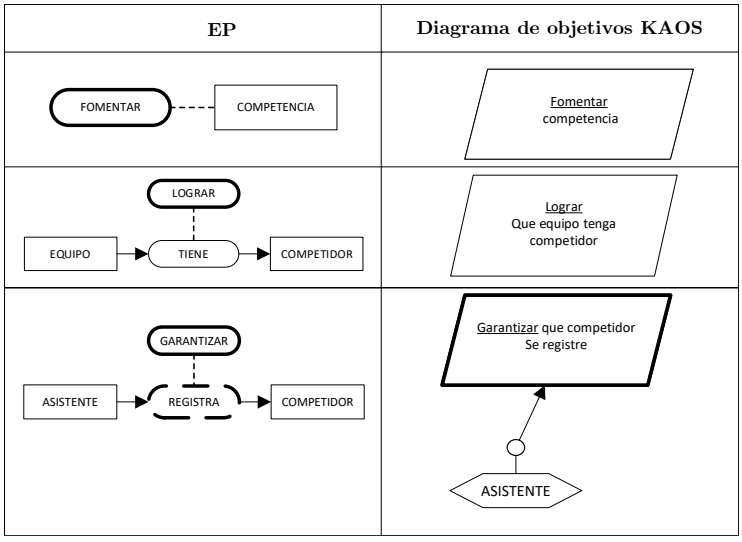


Tabla 3.8. Equivalencia de EP al diagrama de casos de uso. (Zapata, 2007)



En las siguientes propuestas se plantean equivalencias de los EP a otros esquemas conceptuales: Zapata *et al.* (2011d) y Zapata (2012) presentan una equivalencia entre los EP y el *diagrama de objetivos* KAOS (véase la Tabla 3.9).

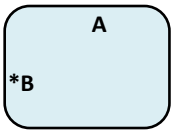
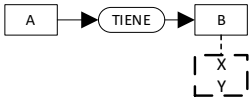
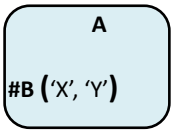
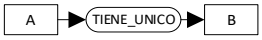
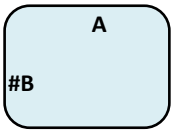
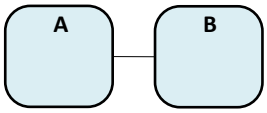
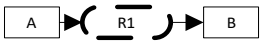
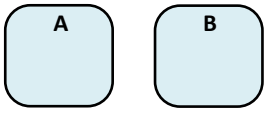
Tabla 3.9. Equivalencia de EP al diagrama de objetivos KAOS. (Zapata et al., 2011d)



El diagrama de objetivos KAOS (por sus siglas en inglés, *Knowledge Acquisition Automated Specification*), se utiliza en ingeniería de requisitos para modelar objetivos, restricciones y responsabilidades del sistema (Zapata et al., 2011d).

Por otro lado, Chaverra (2011) y Zapata *et al.* (2011c) proponen una equivalencia del EP al *diagrama entidad-relación* , un modelo utilizado para diseñar bases de datos mediante la representación de entidades, atributos y relaciones, como se detalla en la Tabla 3.10.

Tabla 3.10. Equivalencia de EP al diagrama entidad-relación. (Chaverra, 2011)

EP	Entidad-relación	SQL
		CREATE TABLE **A** (**B** varchar (30) default NULL,)
		CREATE TABLE **A** (**B** ENUM ('X', 'Y'))
		CREATE TABLE **A** (**B** varchar (30) default NULL, UNIQUE (B))
		CREATE TABLE **B** () CREATE TABLE **A** UNDER **B** ()
		CREATE TABLE **A** ( ) CREATE TABLE **B** ( )

También, Noreña (2014) propone el *grafo de interacción de eventos* (EIG, por sus siglas en inglés, *event interaction graph*), a partir de la notación de EP para representar el flujo de interacción entre eventos y estados del sistema. Las equivalencias desde el EP se basan en relaciones dinámicas (en las líneas de conjunción), eventos disparadores (círculos rojos) y eventos de resultado

(círculos azules) presentes en el EP (véase la Figura 3.7).

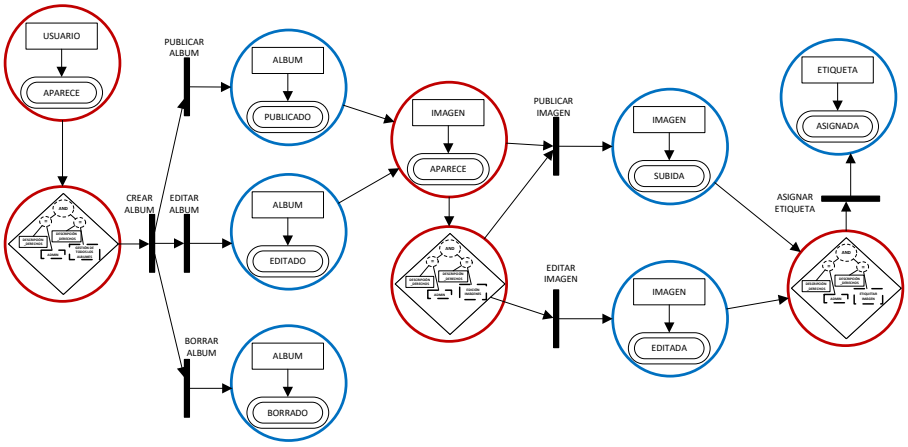
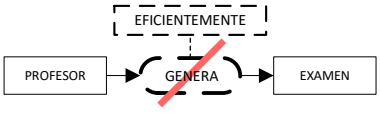
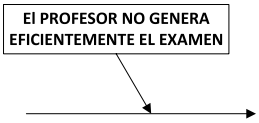


Figura 3.7. Grafo de interacción de eventos. Noreña-Cardona (2014)

Asimismo, Vargas (2016) establece una equivalencia entre la representación de problemas en EP y el diagrama *causa-efecto* (también conocido como diagrama de espina de pescado o diagrama de Ishikawa , en honor a su creador Kaoru Ishikawa). Este diagrama permite visualizar un problema central en la cabeza del pescado, mientras que sus causas se despliegan en las espinas principales, y los efectos se representan como ramificaciones secundarias de estas causas. La cabeza del diagrama, las espinas principales y sus efectos se representan conforme a la estructura sintáctica de un problema en EP (véase la Figura 3.11).

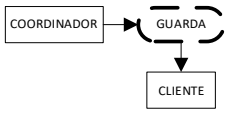
Tabla 3.11. Equivalencia de EP al diagrama causa-efecto. (Vargas, 2016)

EP	Diagrama causa-efecto
	

Equivalencias entre EP y artefactos ágiles: Villamizar (2019) propone una equivalencia entre historias de usuario (HU, artefactos que describen requisitos)

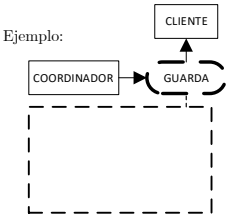
y EP utilizando UN-Lencep, un lenguaje controlado diseñado para generar artefactos . Un ejemplo de esta equivalencia se presenta para la HU “guardar cliente” en la Tabla 3.12.

Tabla 3.12. Plantilla Equivalencia de EP a HU. (Villamizar, 2019)

<table>
<tr><th colspan="3">HU</th><th>UN-Lencep/EP</th></tr>
<tr><td>Código</td><td colspan="2">001</td><td></td></tr>
<tr><td>Nombre</td><td colspan="2">Guardar cliente</td><td></td></tr>
<tr><td>Actor</td><td colspan="2">Coordinador</td><td></td></tr>
<tr><td>Descripción</td><td colspan="2">Como (actor)
quiero (funcionalidad)
para (beneficio)
Ejemplo: Como coordinador
quiero guardar información
de clientes</td><td>Relación dinámica:
actor + funcionalidad (verbo + concepto)
A<R1>B
Ejemplo: coordinador guarda cliente
 
</td></tr>
<tr><td>HU relacionada</td><td>Código</td><td>Nombre</td><td></td></tr>
<tr><td>Módulo</td><td colspan="2">Clientes</td><td></td></tr>
<tr><td rowspan="2">Criterios de aceptación</td><td>Condición</td><td>Resultado</td><td>Condicional</td></tr>
<tr><td>Cuando se + acción + concepto

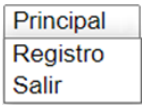
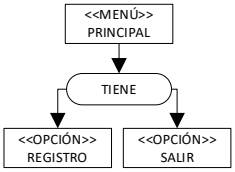
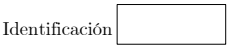
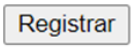
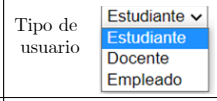
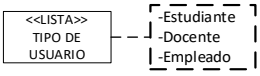
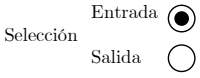
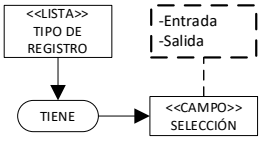
Ejemplo: Cuando se ingrese la información de cliente</td><td>Se debe cumplir que + especificación (restricción o concepto + se + acción + concepto)

Ejemplo: Se debe cumplir que si coordinador ingresa información del cliente, se presenta mensaje "Cliente se guardó satisfactoriamente".</td><td><SI>{COND
<ENTONCES>
A<R1>B,
<SINO> C<R2> D

Especificación (restricción o concepto) <R1> C
(Las condiciones y el resultado se deben incluir en una especificación).
Ejemplo:
 
</td></tr>
</table>

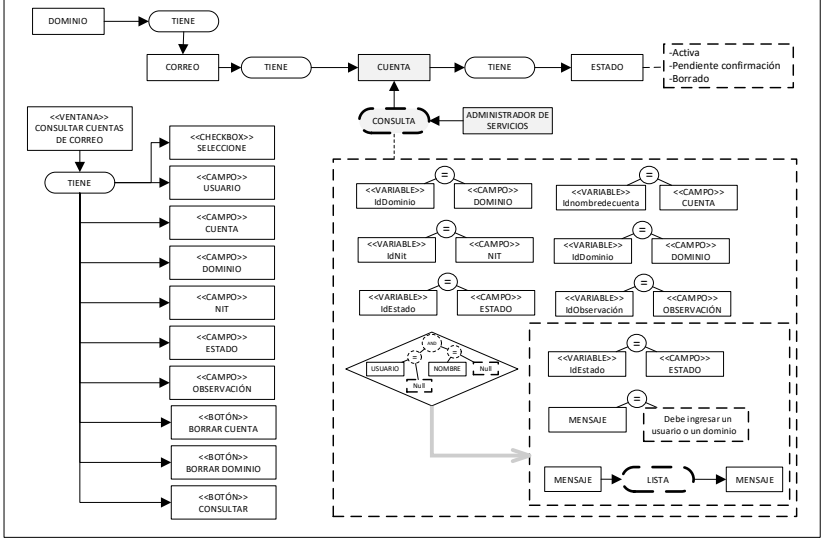
Los criterios de aceptación, que definen las condiciones y resultados esperados de una HU, se representan mediante la especificación/restricción de relaciones dinámicas en los EP. Además, cuando estos criterios incluyen elementos visuales de un prototipo de interfaz de usuario , se pueden describir mediante EP utilizando *estereotipos*, como señala Villamizar (2019). En estos casos, las relaciones estructurales permiten establecer dependencias entre elementos visuales, como menús, campos, botones, listas, botones radio, entre otros como se representan en la Tabla 3.13. Por ejemplo, «menú» "principal" "tiene" «opción» "registro" y «opción» "salir".

Tabla 3.13. Equivalencia de EP a HU. (Villamizar, 2019)

Elemento interfaz	Prototipo	EP
Menú		
Campo		
Botón		
Lista		
Botón radio		

También, Villamizar (2019) muestra ejemplos donde se aplican las reglas descritas en la Tabla 3.13 para derivar especificaciones desde las HU. Un caso específico de esta aplicación se detalla en la Tabla 3.14, donde se representa el EP a partir de los criterios de aceptación de la HU.

Tabla 3.14. Ejemplo de Equivalencia entre EP e HU. (Villamizar, 2019)

<table>
<tr><th colspan="5">HU</th></tr>
<tr><td>Código</td><td colspan="4">002</td></tr>
<tr><td>Nombre</td><td colspan="4">Consultar cuentas de correo</td></tr>
<tr><td>Actor</td><td colspan="4">Administrador de servicios</td></tr>
<tr><td>Descripción</td><td colspan="4">Como Administrador de servicios quiero consultar cuenta de correo que se encuentre en el sistema de la compañía para gestionarla.</td></tr>
<tr><td>HU relacionada</td><td>Código</td><td>001</td><td>Nombre</td><td>Administrador de cuentas</td></tr>
<tr><td>Módulo</td><td colspan="4">Gestión cuentas de correo</td></tr>
<tr><td rowspan="4">Criterios de aceptación</td><td colspan="2">Condición</td><td colspan="2">Resultado</td></tr>
<tr><td colspan="2">Cuando se consulta la cuenta de correo</td><td colspan="2">Se debe cumplir que si campo Usuario es vacío y campo Dominio es vacío, se presenta mensaje “Debe ingresar un usuario o un dominio”</td></tr>
<tr><td colspan="2">Cuando se consulta la cuenta de correo</td><td colspan="2">Se debe cumplir que si Idusuario es igual a Usuario o Iddominio es igual a Dominio, se visualiza Tabla “Listado de cuentas de correo registradas”.</td></tr>
<tr><td colspan="2">Cuando se visualiza tabla “Listado de cuentas de correo registradas”</td><td colspan="2">Se debe cumplir que el checkbox seleccione es vacío.</td></tr>
<tr><td colspan="5"> 
</td></tr>
</table>

Equivalencias de EP a Código fuente: Zapata y Cardona (2008) proponen el código 3.1 en C# basados en las reglas heurísticas del diagrama de clases (véase la Tabla 3.4).

Código 3.1. De EP a C#

```
foreach (ElementoPreconceptual elem in
     conexionesDiagrama)
{
    obj = elem as Conexion;
    if (obj != null)
    {
        rel = obj.Inicio as
             RelacionEstructura;
        con = obj.Fin as Concepto;
        if (!(rel != null && con !=
             null))
            continue;
        if (!(rel.Nombre.ToLower().
             Equals("es")))
            continue;
        foreach (ElementoPreconceptual
             elem2 in
             conexionesDiagrama)
        {
            Conexion con2 = elem2 as
                 Conexion;
            Implicacion impl = elem2 as
                  Implicacion;
            Condicional cond = elem2 as
                  Condicional;
            if (impl != null —— cond !=
                  null)
                continue;
herencia = new Herencia(rel
     .Id);
if (con2.Fin.Id == rel.Id
     && con2.Inicio is
     Concepto)
{
    claseMadreB = new Clase
         (con.Id);
    claseMadreB.Nombre =
         con.Nombre;
    claseHijaA = new Clase(
         con2.Inicio.Id);
    claseHijaA.Nombre = (
         con2.Inicio as
         Concepto).Nombre;
    herencia.Inicio =
         claseHijaA;
    herencia.Fin =
         claseMadreB;
    this.Herencias.Add(
         herencia);
    this.agregarClase(
         claseHijaA);
    this.agregarClase(
         claseMadreB);
}
            }
        }
    }
```

Manjarrés (2010) a partir de las restricciones de conceptos hoja (véase la Figura 3.8) propone su equivalencia en el código 3.2 en Visual Basic.

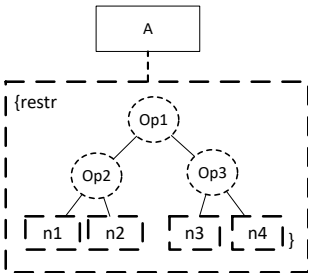


Figura 3.8. Restricción en concepto hoja. (Manjarrés, 2010)

Código 3.2. Restricción EP en Visual Basic

```
If nombreClase = "" Then
 nombreClase = clase.Text
 End If

 For Each c In clase.FromConnects
 nombreRestr = ""
 If esLinea(c.FromSheet) Then
    For Each c2 In c.FromSheet.Connects
    'Buscar las "notas" asociadas que
         contienen restricciones
     If esCompuesta(c2.ToSheet) And
          tiene Nota(c2.ToSheet) And c2
          .ToSheet.Name <> clase.Name
          Then
 Set restr = c2.ToSheet

 For Each fig In restr.Shapes
 If (aux = 1) And (Not esNota(fig))
      Then
 yMin = Val(fig.Cells("PinY").ResultStr
      ("mm"))
 Set figMin = fig
 aux = aux + 1
 End If
  'Extraer el nombre de la restricción
       del texto de la "nota"

If esNota(fig) Then
 Set nota = fig

For nNombre = 1 To Len(nota.Text)
letra = Mid(nota.Text, nNombre, 1)
If (letra >= "a" And letra <= "z") Or
(letra >= "A" And letra <= "Z") Then
nombreRestr = nombreRestr & Mid(nota.
     Text, nNombre, 1)
End If
Next nNombre
'Determinar el nodo raíz del arbol de
     restricciones
Else
   If Val(fig.Cells("PinY").ResultStr(
        "mm")) > yMin Then
    yMin = Val(fig.Cells("PinY").
         ResultStr("mm"))
    Set figMin = fig
   End If
End If
Next
operadoresProcesados = ""
'Función que procesa el árbol para
     generar la expresion declarativa
     de la restricción
restrStr = GenerarRestriccion(figMin)
agregarRestriccion nombreClase,
     nombreRestr & " = " & restrStr
End If
Next
End If
Next
```

Chaverra (2011) presenta reglas de traducción automática para las relaciones estructurales y dinámicas en EP hacia código JSP y PHP (véase la Tabla 3.15) y presenta un ejemplo de la relación dinámica "profesor lista alumno" en la Tabla 3.16 en estos lenguajes.

También, Chaverra (2011) presenta la equivalencia de código HTML desde EP mostrando el resultado en la interfaz en la Tabla 3.17.

Chaverra (2011) y Zapata *et al.* (2011b) generan reglas para controladores en código JSP y PHP (véanse las Tablas 3.18 y 3.19).

Desde bases de datos, Chaverra (2011) y Zapata *et al.* (2011c) proponen la generación automática de diagramas entidad-relación en código SQL a partir de EP (véase la Tabla 3.10).

Tabla 3.15. Equivalencia de EP a código JSP y PHP. (Chaverra, 2011)

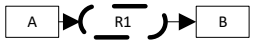
EP	JSP	PHP
	`public class A extends B { } public class A { private TIPO b; }`	`<?php class A extends B { } ?> <?php class A { var $b; } ?>`
	`public class A { private TIPO b; }`	`<?php class A { var $b; } ?>`
	`public class A { } Public class B { Public TIPO R1(){ } }`	`<?php class A { } class B { function R1(){ } }`

Tabla 3.16. Equivalencia de relación dinámica "lista" a JSP y PHP. (Chaverra, 2011)

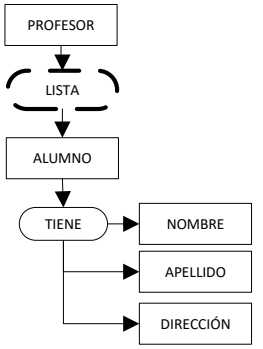
EP	JSP	PHP
 Nombre | Apellido | Dirección Carolina | Gutierrez | Cr 70 # 25-30 Pedro | Cardona | Cr 81 # 10-24	`<table> <tr> <th>Nombre</th> <th>Apellido</th> <th>Direccion</th> </tr> <% List alumnos=(List)request. getAttribute(" alumnos"); Iterator alumno = alumnos.iterator(); while(alumno.hasNext()) { out.print("<tr>"); out.print("<td>"+alumno. nombre+"</td>"); out.print("<td>"+alumno. apellido+"</td>"); out.print("<td>"+alumno. direccion+"</td>"); out.print("</tr>"); } %> </table>`	`<table> <tr> <th>Nombre</th> <th>Apellido</th> <th>Direccion</th> </tr> <?php $i=0; foreach ($alumnos as $alumno): ?> <tr> <td><?php echo $alumno['Alumno'] ['nombre']; ?> </td> <td><?php echo $alumno['Alumno'] ['apellido']; ?> </td> <td><?php echo $alumno['Alumno'] ['direccion']; ?> </td> </tr> <?php endforeach; ?> </table>`

Tabla 3.17. Equivalencia de EP a HTML. (Chaverra, 2011)

EP	HTML	Interfaz
	`<label for="B">B:</label>` `<input name="B" type="text"/>`	B:

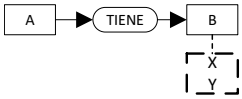
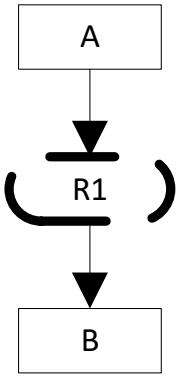
	Opción 1:
	`<select id="B">`
	`<option>X</option>`
	`<option>Y</option>`
	`</select>`
	
	Opción 2:
	`<label for="B">B:</label>`
	`<label for="x">X</label>`
	`< input type=radio name="b" value="X">`
	`<label for="y">Y</label>`
	`<INPUT type=radio name="b" value="Y">`
		B:
		X
		Y
		
		B: X
		Y

Tabla 3.18. Equivalencia de EP a controladores JSP. (Chaverra, 2011)

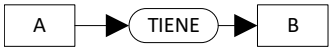
EP	Descriptor	Controlador

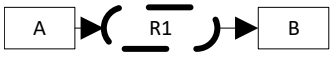
	`<web-app>`
	`<servlet>`
	`<servlet-name>R1B`
	`</servlet-name>`
	`<servlet-class>action.R1B`
	`</servlet-class>`
	`</servlet>`
	
	`<servlet-mapping>`
	`<servlet-name>R1B`
	`</servlet-name>`
	`<url-pattern>/R1B.do`
	`</url-pattern>`
	`</servlet-mapping>`
	`</web-app>`
		`public class R1B extends HttpServlet {`
		
		`public void doPost(`
		`HttpServletRequest request,`
		`HttpServletResponse response) throws ServletException {`
		
		`}`
		`}`

Zapata (2012) implementa la especificación de la relación dinámica “usuario edita album” de la Figura 3.9 al código fuente 3.3 en PHP, donde se pueden observar los diferentes elementos que pertenecen al album en un formulario.

Tabla 3.19. Equivalencia de EP a controladores PHP. (Chaverra, 2011)

EP	PHP

	<?php
	class AController {
	}
	?>
	
	<?php
	class BController {
	}
	?>

	<?php
	class BController {
	function R1 () {}
	}
	?>

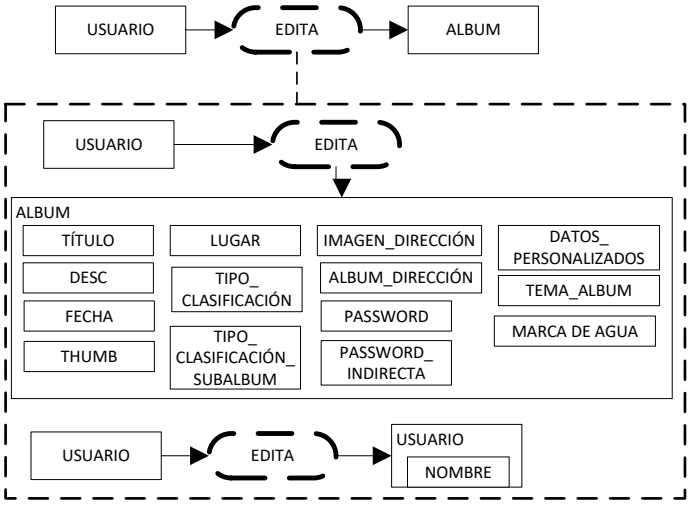


Figura 3.9. Especificación de relación dinámica. (Zapata, 2012)

Código 3.3. De EP a PHP

```
<FORM action="edita" method="post">
<label for="titulo">Título:</label>
<input name="titulo" type="text" id="titulo"/>
<label for="desc">Desc:</label>
<input name="desc" type="text" id="desc"/>
<label for="fecha">Fecha:</label>
<input name="fecha" type="text" id="date"/>
<label for="thumb">Thumb:</label>
<input name="thumb" type="text" id="thumb"/>
<label for="lugar">Lugar:</label>
<input name="lugar" type="text" id="lugar"/>
```

```
<label for=”tipo_clasificacion”>Tipo_Clasificación:</label>
<input name=” tipo_clasificacion” type=”text” id=” tipo_clasificacion”/>
<label for=” tipo_clasificacion_subalbum”> Tipo_clasificación_Subalbum:</label>
<input name=” tipo_clasificacion_subalbum ” type=”text” id=” tipo_clasificacion_
    subalbum”/>
<label for=” imagen_direccion”> Imagen_Dirección</label>
<input name=” imagen_direccion” type=”text” id=”imagen_direccion”/>
<label for=” album_sortdirection”>Album_Dirección:</label>
<input name=”album_direccion” type=”text” id=” album_direccion”/>
<label for=” password”>Password:</label>
<input name=”password” type=”text” id=” password”/>
<label for=” password_indirecta”>Password_Indirecta:</label>
<input name=”password_indirecta ” type=”text” id=” password_indirecta”/>
<label for=”datos_personalizados”>Datos_Personalizados:</label>
<input name=”datos_personalizados” type=”text” id=”datos_personalizados”/>
<label for=”tema_album”> Tema_Album:</label>
<input name=”tema_album” type=”text” id=”tema_album”/>
<label for=”marcadeagua”>MarcadeAgua:</label>
<input name=”marcadeagua” type=”text” id=”marcadeagua”/>
<input name=create type='submit' value='Edita'/>
</FORM>
```

Calle (2016) define patrones de diseño en EP y traduce algunos a Java como el patrón creacional *Abstract Factory* como se observa en la Figura 3.10 y el código 3.4. Calle (2016) también propone reglas de traducción de operadores (*Log, Mod, Sqrt, Exp, Pow, Ab*) y funciones matemáticas predefinidas (*Sin, Cos, Tan, Ctg, Csc, Sec, push y pop*) en la Tabla 3.1.

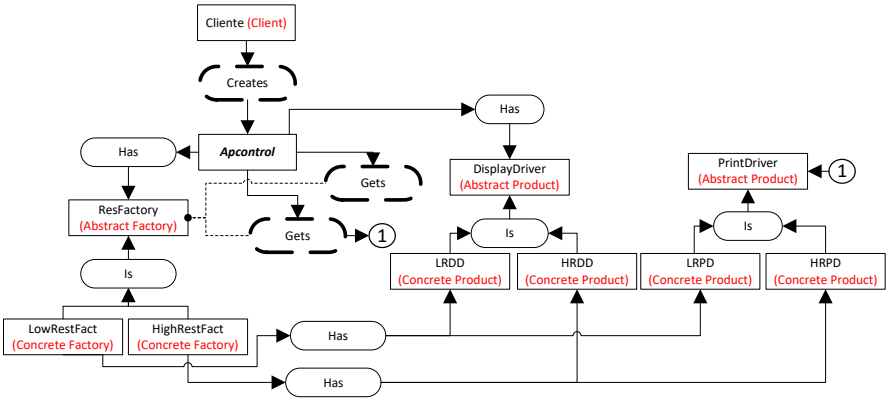


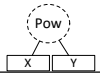
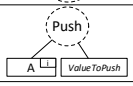
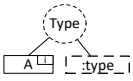
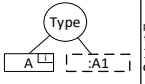
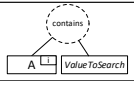
Figura 3.10. Patrón *Abstract Factory* en EP. (Calle, 2016)

Código 3.4. De Patrones en EP a Java

```
public class Client {
    public static void main (String[]
        args) {
    AppControl Apcontrol = new
        ApControl();
```

```
      AppControl.Resfactory = new
           LowResFact();
      AppControl.creates();
    }
}
public abstract class ResFactory {
     public abstract DisplayDriver Gets
          _DisplayDriver();
     public abstract PrintDriver Gets_
          PrintDriver();
}

public class LowResFac extends
     ResFactory{
   HRDD LRDD = new HRDD();
   HRPD LRDD = new HRPD();
   @Override
   public DisplayDriver Gets_
        DisplayDriver() {
      return LRDD;
   }
   @Override
   public DisplayDriver Gets_
        DisplayDriver() {
      return LRPD;
   }
}
public class HightResFac extends
     ResFactory{
   HRDD LRDD = new HRDD();
   HRPD LRDD = new HRPD();
      @Override
   public DisplayDriver Gets_
        DisplayDriver() {
      return HRDD;
   }
   @Override
   public DisplayDriver Gets_
        DisplayDriver() {
      return HRPD;
   }
}
```

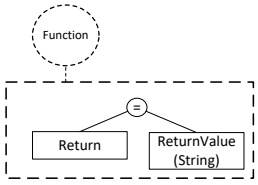
Tabla 3.20. Equivalencia de Funciones en EP a Java. (Calle, 2016)

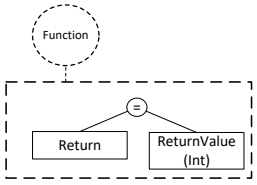
Operador EP	Java	Operador EP	Java
	Math.log(x);		x%y;
	Math.sqrt(x);		Math.exp(x);
	Math.pow(x,y);		Math.abs(x);
	Math.sin(x);		Math.cos(x);
	Math.tan(x);		Math.csc(x);
	1/Math.tan(x);		1/Math.cos(x);
	A.add(ValueToPush);		A.remove(A.size()-1);
	if(A.getClass().getName().equals("type")) { return true; }else { return false; }		**Precondición:** public class A { } public class A1 extends A { } **Código del operador:** A A = new A1();
	A.contains(ValueToSearch);		

Algunos operadores que también propone Calle (2016) son *type* (de asignación para definir un tipo de dato a un concepto y de información para describir el tipo de dato de un concepto: entero, booleano, string, doble o de tipo clase) y *contains* (se usa para verificar si un concepto, parámetro o valor se encuentra almacenado en un vector, véase la Tabla 3.20), y funciones que define el analista en EP a código fuente en Java (véase la Tabla 3.21).

Tabla 3.21. Equivalencia de Funciones definidas en EP a Java. (Calle, 2016)

Funciones EP	Java

	`public String Function()`
	`{`
	`String returnValue = /*Code...*/;`
	`return returnValue;`
	`}`

	`public int Function()`
	`{`
	`int returnValue = /*Code...*/;`
	`return returnValue;`
	`}`

Velásquez (2019) desarrolla una implementación en código C++ basada en EP, aplicando un modelo de simulación avanzada para yacimientos de petróleo. Este trabajo destaca por integrar ecuaciones matemáticas complejas, funciones específicas y eventos dinámicos que permiten modelar y analizar los comportamientos de los yacimientos bajo diferentes condiciones operativas. En la Tabla 3.11, se presenta un ejemplo detallado de una de las ecuaciones y su correspondiente representación en EP y su traducción al lenguaje C++ en el código 3.5.

Por su parte, Zapata-Tamayo (2019) introduce un conjunto de reglas heurísticas orientadas a la generación semiautomática de código en PL/SQL, las cuales se presentan para la especificación de eventos dentro del marco de los EP. Esta propuesta permite traducir la lógica de los eventos en consultas y operaciones de bases de datos. (Véase la Tabla 3.22).

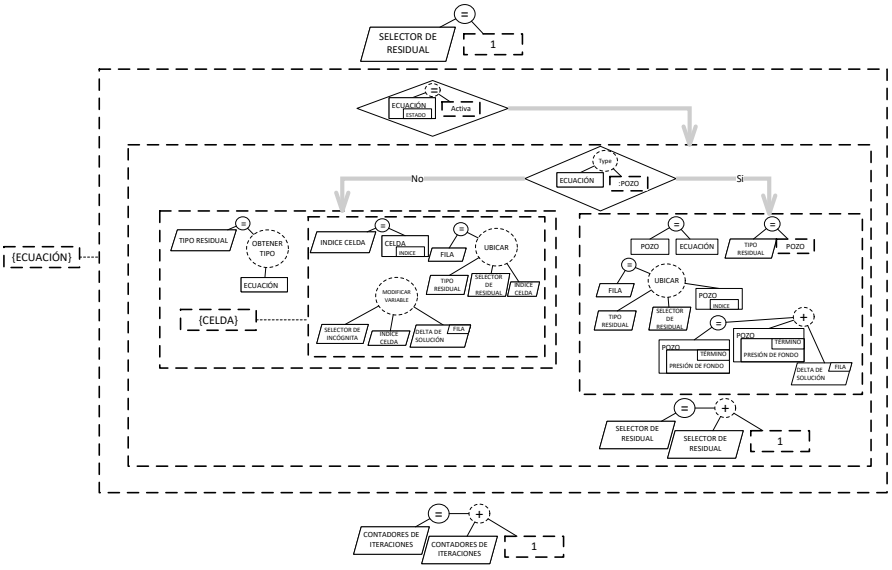


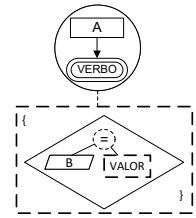
Figura 3.11. Especificación de una ecuación matemática. (Velásquez, 2019)

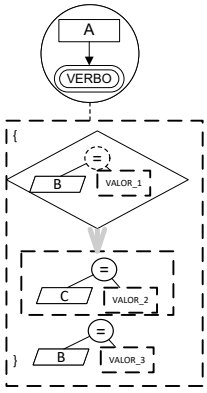
Código 3.5. De Ecuaciones en EP a C++

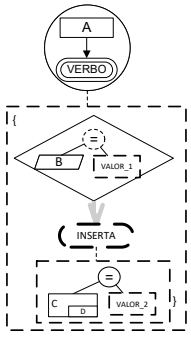
```
residual_selector = 0;

for (auto equation : equations) {
    if (equation->status()) {
        if (equation->type() == typeid(Well).name()) {
            constexpr auto residual_type = "well";
            auto residual_well = std::dynamic_pointer_cast<Well, Equation_Base>(
                 equation);
            row = locate(residual_type, residual_selector, residual_well->index()
                 );
            _residual(row) = calculateWellResidual(term, residual_well);
        } else {
            constexpr auto residual_type = "fluid";
            auto residual_fluid = std::dynamic_pointer_cast<Fluid, Equation_Base>
                 (equation);

            for (auto cell = mesh.begin(); cell != mesh.end(); ++cell) {
                cell_index = (*cell)->index();
                row = locate(residual_type, residual_selector, cell_index);
                _residual(row) = calculateResidual(term, *residual_fluid, mesh, *
                     cell, rock, wells);
            }
        }
    }
    ++residual_selector;
}
```

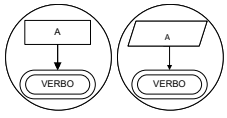
Tabla 3.22. Equivalencia de eventos en EP a PL/SQL. (Zapata-Tamayo, 2019)

Eventos EP	PL/SQL

	**CREATE** [**OR** REPLACE] **TRIGGER** -"A"_VERBO"
	AFTER
	[**UPDATE/INSERT**] OF "B"
	ON ["VARIABLE"]
	DECLARE
	"v"_" + B [VARIABLE].B%TYPE;
	BEGIN
	SELECT B **INTO** "v"_" + B **FROM** [
	VARIABLE];
	IF "v"_" + B [CONDICION] VALOR **THEN**
	END IF;
	END;

	**CREATE** [**OR** REPLACE] **TRIGGER** -"A"_VERBO"
	AFTER
	[**UPDATE/INSERT**] OF "B"
	ON ["VARIABLE"]
	DECLARE
	"v"_" + B [VARIABLE].B%TYPE;
	BEGIN
	SELECT B **INTO** "v"_" + B **FROM** [
	VARIABLE];
	IF "v"_" + B [CONDICION] VALOR"_1
	THEN
	UPDATE [VARIABLE] **SET** C = 'VALOR_2';
	END IF;
	END;
	CREATE [**OR** REPLACE] **TRIGGER** "A"_VERBO +
	"_1"
	AFTER
	[**UPDATE/INSERT**] OF "C"
	ON ["VARIABLE"]
	BEGIN
	UPDATE [VARIABLE] **SET** B = 'VALOR"_3'
	END;

	**CREATE** [**OR** REPLACE] **TRIGGER** -"A""_VERBO"
	AFTER
	[**UPDATE/INSERT**] OF "B"
	ON ["VARIABLE"]
	DECLARE
	"v_" + B [VARIABLE].B%TYPE;
	BEGIN
	SELECT B **INTO** "v"_" + B **FROM** [
	VARIABLE];
	IF "v"_" + B [CONDICION] VALOR"_1
	THEN
	[**UPDATE/INSERT**] "C" **SET** D= 'VALOR
	"_2'
	END IF;
	END;

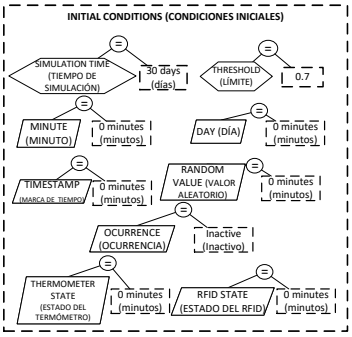
	**CREATE** [**OR** REPLACE] **TRIGGER** -"A_VERBO"

Noreña-Cardona (2020) incluye la implementación de eventos y condiciones iniciales en código Python . Esta propuesta se enfoca en automatizar procesos dinámicos y complejos que dependen de un conjunto inicial de parámetros y variables independientes (véase la Tabla 3.23).

Finalmente, Mosquera (2021) plantea reglas heurísticas desde los EP para la generación automática de aplicaciones móviles multiplataforma. En esta aproximación se combinan las capacidades de Java-Android y Swift-iOS para producir soluciones que son funcionales en diferentes dispositivos y sistemas operativos (véase la Tabla 3.24).

Tabla 3.23. Equivalencia de condiciones iniciales y eventos EP a Python. (Noreña-Cardona, 2020)

EP	Python

	`#PARAMETERS-INITIAL CONDITIONS`
	
	`SIMULATION_TIME = 30 measured in days`
	`THRESHOLD = 0.7 probability in (0-1)`
	`  range`
	
	`#INDEPENDENT VARIABLES-INITIAL`
	`  CONDITIONS`
	
	`MINUTE = 0 measured in days`
	`DAY = 0 probability in (0-1) range`
	`TIMESTAMP = "Next"`
	`RANDOM VALUE = 0.0`
	`OCURRENCE = "Inactive"`
	`THERMOMETER_STATE = "Off"`
	`RFID_STATE = "Off"`

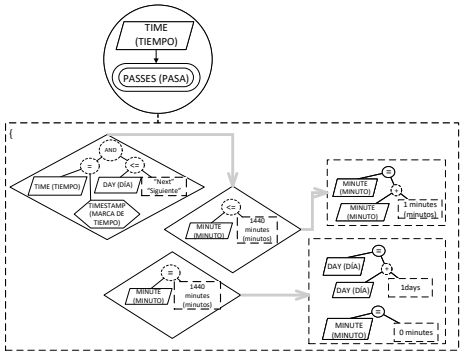
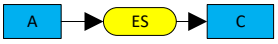
	`Def TIME_PASSES():`
	`  global MINUTE`
	`  global DAY`
	`  global TIMESTAMP`
	`if (DAY <= SIMULATION_TIME`
	` and TIMESTAMP == "Next"):`
	`if (MINUTE < 1440):`
	`  MINUTE = MINUTE + 1`
	`if (MINUTE == 1440):`
	`  DAY = DAY + 1`
	`  MINUTE = 0`

Tabla 3.24. Equivalencia de EP a código de apps móviles. (Mosquera, 2021)

EP	Java-Android	Swift-iOS
	`public class A {}`	`public class A {}`

	`public class A {`
	`  private TYPE b ;`
	`  public TYPE getB ( ) {`
	`  return this.b ;`
	`  }`
	`  public void setB (TYPE newB) {`
	`  this. b = newB;`
	`  }`
	`}`
		`public class A {`
		`  private var b : TYPE ;`
		`  public func getB ( ) ->TYPE{`
		`  return self.b`
		`  }`
		`  public func setB (newB : TYPE!) {`
		`  self. b = newB`
		`  }`
		`}`

	`public class A extends C { }`	`public class A : C { }`

	`public class A {`
	`  public void checkConstraints ( )`
	`  throws AC_NAMEException{`
	`  if (!LOGICAL_EXPRESSION) {`
	`  throw new AC_NAMEException ( ) ;`
	`  }`
	`  }`
	`}`
	`public class AC_NAMEException extends Exception {`
	`  public AC_NAMEException ( ) {`
	`  super (”A must meet : LOGICAL_EXPRESSION” ) ;`
	`  }`
	`}`
		`public class A {`
		`  public void checkConstraints ( )`
		`  throws {`
		`  if (!LOGICAL_EXPRESSION) {`
		`  throw AException.C_NAME(message ”A must meet : LOGICAL_EXPRESSION ” ) ;`
		`  }`
		`  }`
		`}`
		`enum AException : Error {`
		`  case C_NAME(message : String)`
		`  }`

3.2. Planteamiento del problema

En la Tabla 3.25 se realiza una síntesis de los antecedentes. Esta síntesis se presenta para relacionar las contribuciones que cada propuesta realiza a las características (estructural, dinámica, télica y eventual) y equivalencias de los EP.

En estas propuestas se presentan contribuciones en la evolución de la especificación de las características estructurales, dinámicas, télicas y eventuales de los esquemas preconceptuales. Asimismo, se presentan equivalencias de los EP con esquemas conceptuales, como diagramas UML (casos de uso, comunicaciones, máquina de estados, secuencia y clases) diagramas de objetivos KAOS, entidad-relación, EIG y causa-efecto. Además, las propuestas incluyen equivalencias entre los EP, artefactos ágiles y código fuente utilizando reglas heurísticas e implementación desde los EP en lenguajes de programación como C#, Visual Basic, JSP, HTML, SQL, PHP, Java, Python, C++, PL/SQL, Java-Android, Swift-iOS). Sin embargo, ninguna de las propuestas presenta las cuatro características (estructural, dinámica, télica y eventual) de los EP integradas en un único enfoque. En cambio, en estas propuestas las características se representan aisladamente en diferentes referencias bibliográficas. Como consecuencia, el código fuente equivalente a cada característica se implementa en referencias separadas, como se resume en la Figura 3.12. Esta división causa dificultad en la comprensión de las características, generando ambigüedad en la creación de EP durante las etapas de análisis y diseño de los sistemas de software.

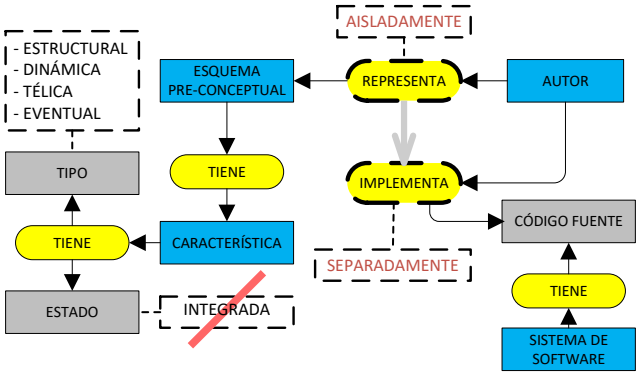


Figura 3.12. Planteamiento del problema. Los autores

Tabla 3.25. Propuestas previas en características y equivalencias de EP. Los autores

Propuesta Previa	Estructural	Dinámica	Télica	Eventual	Esquema conceptual, artefacto	Código Fuente	Lenguaje
Zapata (2007	Conceptos clase, hoja, R. estructural	R. dinámica	-	-	UML (clase, comunicaciones, máquina de estados)	-	-
Zapata *et al.* (2007a)	-	-	Relación de logro	-	-	-	-
Zapata *et al.* (2007b)	-	-	-	-	Casos de uso	-	-
Zapata y Cardona (2008)	-	-	-	-	-	R. estructural	C#
Manjarrés (2010)	Restricción	-	-	-	-	Restricción	Visual Basic
Chaverra (2011)	Tipos de datos	Especificación R. dinámica	-	-	Entidad-relación	Reglas R. estructural y dinámica	JSP, PHP, HTML, SQL
Zapata *et al.* (2011b)	-	-	-	-	-	R. estructural y dinámica	PHP
Zapata *et al.* (2011a)	-	-	-	-	-	EP ejecutables	-
Zapata *et al.* (2011c)	-	-	-	-	Entidad-relación	Código SQL	SQL
Zapata *et al.* (2011d)	-	-	-	-	KAOS	-	-
Zapata (2012)	-	-	-	R. eventual	UML	Código fuente	PHP
Zapata y Garcés (2008)	-	-	-	-	D. secuencia	-	-
Villamizar (2019)	-	-	-	-	HU, mock-up	-	-
Calle (2016)	-	-	-	-	-	Patrones de diseño, operadores y funciones matemáticas	Java
Vargas (2016)	-	-	R. objetivos y problemas	-	Causa-efecto	-	-
Zapata y Castro (2017)	-	-	KPI R. de logro	-	-	Código KPI	KPI
Noreña *et al.* (2019); Noreña (2014; 2020)	-	-	-	Especificar eventos, notación matemática	EIG	Eventos, condición inicial	Python
Velásquez (2019)	-	-	-	-	-	Código eventos, ecuaciones matemáticas	C++
Zapata-Tamayo (2018)	-	-	-	-	-	Generación semiautomática eventos	PL/SQL
Mosquera (2021)	Equivalencia	Equivalencia	-	-	-	Reglas app multiplataforma	Java-Android, Swift-iOS

Capítulo 4

Especificación de las características de los EP

Se presenta un compendio de las reglas para la especificación revisada de las características estructurales, dinámicas, télicas y eventuales de EP. Este compendio se organiza como un manual guía para facilitar la creación de EP, destacando además sus equivalencias con esquemas conceptuales, artefactos ágiles y código fuente. Estas reglas se pueden utilizar en la representación de cualquier dominio, proporcionando un enfoque integral para el diseño y la especificación de un sistema dentro del proceso de desarrollo de software.

4.1. Características

4.1.1. Características Estructurales

La construcción formal que realiza Zapata (2007) de las características estructurales de los EP, permite relacionar los conceptos de un dominio, aquellos que se pueden clasificar como clases, atributos o conceptos hoja con sus relaciones estructurales de agregación y generalización.

Concepto: La primera regla general para la representación en EP establece que un concepto (sustantivo o sintagma nominal) se debe nombrar en su forma singular. Esto facilita la construcción de triadas—estructura básica compuesta por dos conceptos unidos por una relación—que permiten una lectura coherente y comprensible. Este enfoque lingüístico, influenciado por principios de la lingüística computacional, contribuye a reducir la ambigüedad y promueve un diseño estandarizado que mejora la trazabilidad desde el lenguaje natural hasta la estructura formal del EP. Los conceptos pueden adoptar distintas formas según el nivel de abstracción, tal como se ilustra en la Tabla 4.1.

Tabla 4.1. Conceptos. Basado en Zapata (2007), Chaverra(2011), Villamizar(2019), Calle (2016) y Noreña(2020)

Descripción	Regla EP	Ejemplo
Concepto clase	CLASE	AUTÓMOVIL AUTÓMOVIL
Concepto hoja	CONCEPTO HOJA	PLACA
	TÉRMINO VECTOR	i MEDIDA
	TÉRMINO1 TÉRMINO2 MATRIZ	i j CAMINO
Tipo de dato	CONCEPTO	NOMBRE
	CONCEPTO://	FECHA DE EXPIRACIÓN://
	CONCEPTO:#	PESO:#
	CONCEPTO:?	ESTADO: ?
	CONCEPTO:@	EMAIL: @
Concepto estereotipado	<<ESTEREOTIPO>> CONCEPTO	<<MENÚ>> PRINCIPAL <<BOTÓN>> CONSULTAR
Concepto independiente	PARÁMETRO	= UMBRAL 0,7
	VARIABLE INDEPENDIENTE	= ESTADO DEL TERMÓMETRO ON
	TÉRMINO VECTOR INDEPENDIENTE	= TÉRMINO TEMPERATURA DE ESTACIÓN 40°C

- **Concepto clase:** El concepto clase se representan mediante el nodo *concepto* en la notación de los EP. Por ejemplo, un concepto clase "automóvil", tal como se ilustra en la Tabla 4.1. De manera general, los conceptos clase se representan en color azul; sin embargo, si varias clases comparten el mismo

concepto, este se representa en color verde. Por ejemplo, el concepto clase "automóvil" en color verde en la Tabla 4.1.

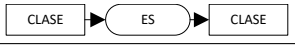
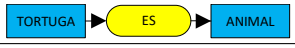
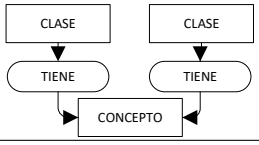
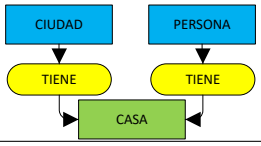
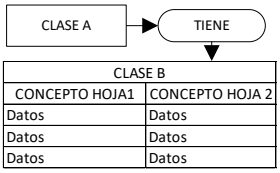
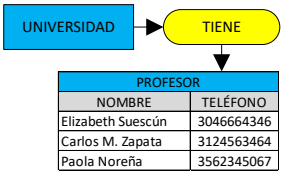
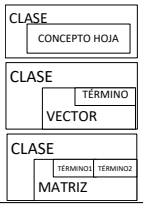
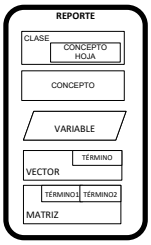
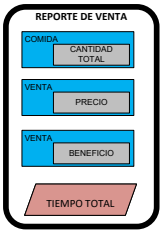
- **Concepto hoja:** El concepto hoja se representan con el nodo *concepto* en la notación de los EP en color gris, por ejemplo el concepto hoja "placa" en la Tabla 4.1. Dentro de esta categoría, los arreglos, se clasifican como un tipo de concepto hoja, los cuales se representan con los nodos *vector* y *matriz* respectivamente en la notación de EP (Noreña-Cardona, 2020), por ejemplo el vector "medida[i]" y la matriz "camino[i][j]" en la Tabla 4.1).
- **Tipo de dato:** Los conceptos hoja en un nivel de abstracción para programación se pueden clasificar según el *tipo de dato* (texto, fecha, número, booleano, email) desde EP. Primero se nombra el concepto y luego el tipo de dato, por ejemplo, el concepto "peso" es de tipo número y se representan en color gris (véase la Tabla 4.1; Chaverra, 2011). Esta regla se aplica también, para vectores y matrices (véase la Tabla 4.1).
- **Concepto estereotipado:** El concepto (clase u hoja) puede incluir un *estereotipo* para facilitar la legibilidad de elementos, por ejemplo, en la interfaz gráfica o en bases de datos se incluye el tipo de elemento dentro del símbolo de doble etiqueta "« »" y luego se nombra el concepto (válido para todos los tipos de conceptos). Por ejemplo, el «menú» "principal" y el «botón» "continuar" en la Tabla 4.1 (Villamizar, 2019).
- **Concepto independiente:** Los conceptos independientes involucran variables independientes, parámetros y vectores independientes y no requieren algún tipo de relación con los demás conceptos, por lo que, se pueden encontrar dentro de las especificaciones, restricciones y condiciones iniciales, cuando se requiere asignar valores temporales. En la notación de EP, estos valores se representan utilizando los nodos *variable independiente*, *parámetro* y *vector independiente* respectivamente. Por ejemplo, la variable independiente "estado del termómetro", el parámetro "umbral" y el vector independiente "temperatura de estación" en la Tabla 4.1.

La relación estructural (RE) permite representar la generalización y agregación entre clases.

RE de generalización: La relación estructural de generalización o herencia entre clases, se compone de una triada de dos conceptos clases y una relación estructural (mediante el nodo *relación estructural* y el verbo "es" en su notación formal). Algunas de las expresiones que se relacionan son: "es una especie de",

"es un tipo de" y "es una clase de". La relación estructural de generalización se representa como "clase A es clase B" (Zapata, 2007), por ejemplo, "tortuga es animal", cuya lectura también puede ser: "tortuga" es una clase de "animal" (véase la Tabla 4.2).

Tabla 4.2. Relación estructural. Basado en Zapata (2007) y Chaverra (2011)

Descripción	Regla en EP	Ejemplo
RE de generalización		
RE de agregación		
RE de agregación: Agrupación de conceptos hoja con datos		
RE de agregación: Concepto hoja obligatorio		
RE de agregación: Concepto único		
RE de agregación: vector		
RE de agregación: matriz		
RE de agregación usando nodo "concepto-clase"		
Reporte		

RE de agregación: La relación estructural de *agregación* establece una dependencia entre una clase y su concepto hoja. Esta relación se representa mediante una triada que incluye: un concepto clase, una relación estructural (utilizando el nodo *relación estructural* y el verbo "tiene" en su notación formal) y un concepto hoja. También se presenta la agregación entre clases, es decir que la triada se compone de dos conceptos clase y la relación estructural "tiene".

En el lenguaje natural, el verbo "tiene" se puede expresar con términos como: "incluye", "contiene", "posee", "está compuesto por", "está formado por", "se conforma de", "se compone de" o "se divide en". Estas variaciones facilitan la representación de la agregación en diferentes contextos. La regla que rige esta relación se define como: "clase tiene clase" y "clase tiene concepto hoja" (Zapata, 2007). Un ejemplo es: "cuenta tiene perfil" y "automóvil tiene placa" respectivamente en la Tabla 4.2.

Cuando varias clases comparten un mismo concepto, este se representa en *color verde*, conectándose a las clases correspondientes mediante relaciones estructurales de agregación. Por ejemplo, las relaciones estructurales "ciudad tiene casa" y "persona tiene casa" comparten el concepto hoja "casa", indicando que este concepto pertenece a ambas clases (véase la Tabla 4.2).

- **Agrupación de conceptos hoja con datos:** Los EP ejecutables permiten la *agrupación de conceptos hoja con datos*, permitiendo observar el detalle de los datos con el elemento de la notación *tabla*, por ejemplo, la clase "universidad tiene profesor" y "profesor" tiene "nombre" y "teléfono", como se presenta en la Tabla 4.2. De este modo, hay una visualización de los diferentes profesores con sus respectivos datos.
- **Concepto hoja obligatorio:** Cuando un concepto hoja es obligatorio, se representa mediante un enlace de *conexión obligatoria* (doble) desde la clase hacia el concepto hoja. Por ejemplo, en la Tabla 4.2, "extranjero tiene pasaporte", donde "pasaporte" es un concepto obligatorio.
- **Concepto único:** La RE de agregación para un *concepto único* se compone de una triada con un concepto clase, una RE (mediante el nodo "relación estructural", el verbo "tiene_único" en su notación formal), un enlace de *conexión obligatoria* (doble) y un concepto hoja. Por ejemplo, "empresa tiene_único Nit" (véase la Tabla 4.2).
- **Vector y matriz:** En los conceptos hoja tipo *vector* y *matriz*, se aplican las

mismas reglas para la relación estructural de agregación, es decir, una triada con la clase, la RE y el concepto hoja tipo vector o matriz. Por ejemplo, "evaluación tiene medida[i]" con un concepto hoja tipo vector y "juego tiene camino [i][j]" con un concepto hoja tipo matriz (véase la Tabla 4.2).

- **Nodo "concepto-clase":** Cuando se representa la RE de agregación, se puede incluir en otro lugar utilizando el nodo de *concepto-clase* después de tener la representación. Por ejemplo, "placa-automóvil" para utilizar la RE "automóvil tiene placa", "medida-evaluación" para utilizar la RE "evaluación tiene medida", donde medida es un vector y "camino-juego", para usar la RE "juego tiene camino", donde "camino" es una matriz (véase la Tabla 4.2).
- **Reporte:** Para representar un reporte se utiliza el elemento *marco* de la notación y se incluyen los conceptos hoja que se requieran. Por ejemplo, el "reporte de venta" con los conceptos "cantidad total de comida", "precio de venta", "beneficio de venta" y la variable "tiempo total" en la Tabla 4.2.

Operaciones matemáticas: las operaciones permiten representar la funcionalidad de los conceptos hoja y los conceptos independientes mediante valores y operadores aritméticos, relacionales, lógicos y de arreglos. Estas se representan en forma de *árboles binarios*, en los que los *conceptos* se conectan mediante nodos *operador* y *asignación* (Véase la Tabla 4.3).

- **Asignación:** El operador *asignación* se puede usar para asignar valor a cualquier tipo de concepto. Por ejemplo, el concepto hoja "temperatura" de la "medida" es igual a "5°C", la variable "estado del sensor" es igual a "activo", el parámetro "PI" igual a 3,1416, el vector "enlace de red" es igual a N24, la matriz "dato" es igual a 0 en la Tabla 4.3.
- **Ecuación matemática:** Los *operadores matemáticos* permiten representar ecuaciones mediante el uso de operadores matemáticos y valores-nota que se conectan mediante el enlace "operador". Por ejemplo, la ecuación del concepto "resultado" permite la relación de varios operadores como multiplicación, raíz cuadrada de 25, logaritmo base 10 y división de la variable "magnitud" y el parámetro "tiempo inicial" en la Tabla 4.3.
- **Condición:** Las *condiciones* permiten evaluar conceptos o valores para determinar si se cumple un criterio específico. Este criterio permite que se tomen decisiones ejecutando acciones solo cuando los elementos coinciden con la condición, por ejemplo, el "tiempo" es menor o igual a 30 días y el "pago" es mayor que 0 en la Tabla 4.3.

Tabla 4.3. Operaciones matemáticas. Basado en Chaverra (2011), Calle (2016), Velásquez (2019) y Noreña (2020)

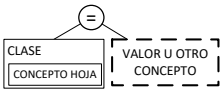
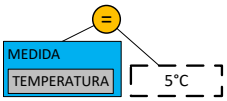
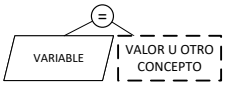
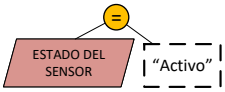
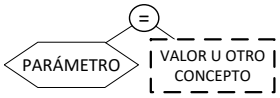
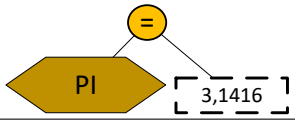
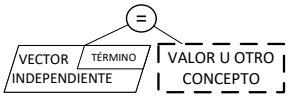
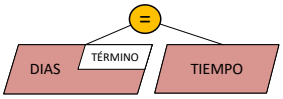
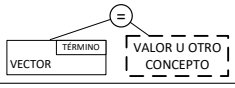
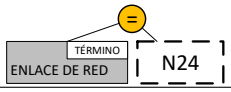
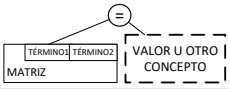
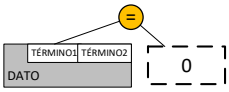
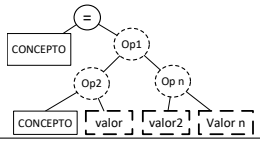
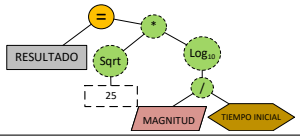
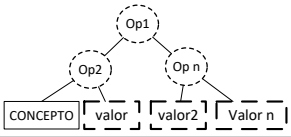
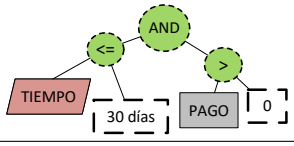
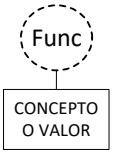
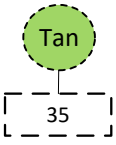
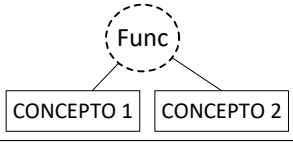
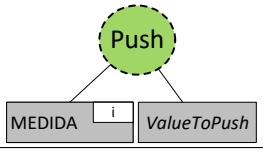
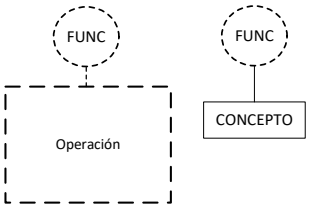
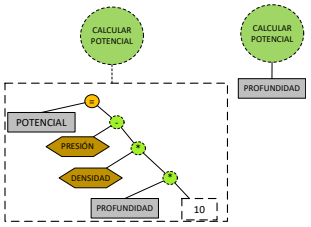
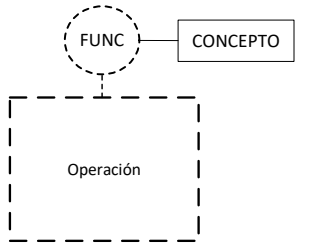
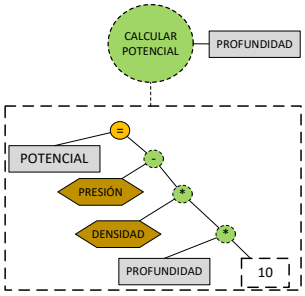
Descripción	Regla en EP	Ejemplo
Asignación de concepto hoja		
Asignación de variable		
Asignación de parámetro		
Asignación de vector independiente		
Asignación de vector		
Asignación de matriz		
Ecuación matemática		
Condición		
Función predefinida con un concepto o valor		
Función predefinida con varios conceptos		

Tabla 4.3. (Continuación) Operaciones matemáticas. Basado en Chaverra (2011), Calle (2016), Velásquez (2019) y Noreña (2020)

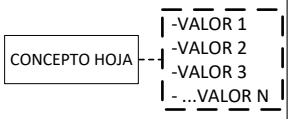
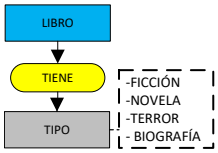
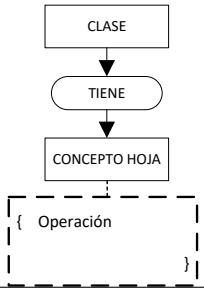
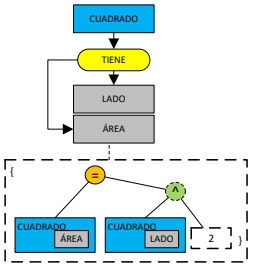
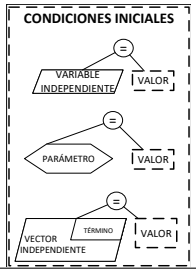
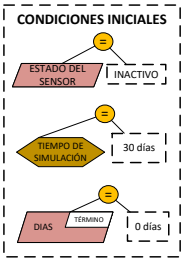
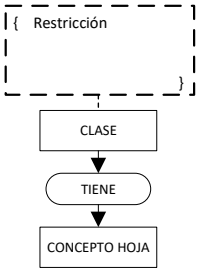
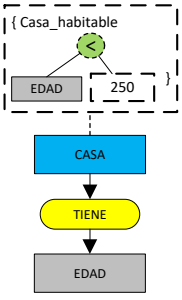
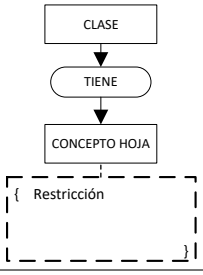
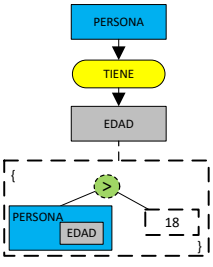
Descripción	Regla en EP	Ejemplo
Función definida		
Función definida con concepto		

- **Función:** Una *función predefinida* es aquella que se encuentra establecida desde las reglas matemáticas y se puede invocar en el desarrollo. Por ejemplo, las funciones "tangente de 35" y "push" del vector "medida" en la Tabla 4.3. Una *función definida* se crea manualmente por el analista o desarrollador para ejecutar una operación dentro del sistema. Por ejemplo, "calcular potencial": primero se establece su especificación y, luego, se puede utilizar en distintos lugares de un EP. Además, el operador de una función puede incorporar un concepto, cuando es diferente al de su especificación, un caso ilustrativo es la función "calcular profundidad" en la Tabla 4.3.

Especificación de conceptos: Los conceptos pueden incluir una especificación usando un enlace de *concepto-nota* con un *valor-nota*, *especificación* o *restricción*. (Véase la Tabla 4.4).

- **Valores:** Los conceptos hoja pueden tener un conjunto cerrado de valores, ya sean booleanos o provenientes de una lista desplegable en la interfaz. Para especificar estos valores se relaciona el concepto con un enlace *concepto-nota* hacia una *nota-valor* que contiene los valores, por ejemplo "tipo de libro" en la Tabla 4.4).

Tabla 4.4. Especificación de conceptos. Basado en Chaverra (2011) y Manjarrés (2010)

Descripción	Regla en EP	Ejemplo
Valores		
Valores derivados		
Condiciones iniciales		
Restricción de clase		
Restricción de concepto hoja		

- **Condiciones iniciales:** Los valores iniciales de los *conceptos independientes* se representan mediante el aglutinador *condición inicial*, en el que se incluye la asignación de los valores iniciales de parámetros, variables y vectores independientes, con el objetivo de completar información del sistema en una simulación. Por ejemplo, la especificación de condiciones iniciales con la variable "estado del sensor igual a inactivo", el parámetro "tiempo de simulación igual a 30 días" y el vector "días igual 0 días" en la Tabla 4.4.

Restricción de conceptos: Los conceptos clase y los conceptos hoja pueden incluir restricciones mediante el uso del aglutinador *restricción* (Manjarrés, 2010). Estas restricciones se definen internamente mediante operaciones de comparación usando operadores lógicos y relacionales, los cuales se aplican sobre conceptos hoja, variables o parámetros (véase la Tabla 4.4).

- **Restricción de clase:** La restricción de una clase se relaciona mediante la *restricción* y se vincula al concepto clase mediante el enlace *concepto-nota*, por ejemplo, la clase "casa" tiene una restricción: si "edad de la casa" menor de "250" años, para que sea una "casa_habitable" en la Tabla 4.4.
- **Restricción de concepto hoja:** La restricción de un concepto hoja se relaciona mediante la *restricción* al *concepto hoja*, por ejemplo, la *restricción*: si la "edad de persona" es mayor a "18" años, del concepto hoja "edad" en la Tabla 4.4).

4.1.2. Características Dinámicas

La construcción formal que realiza Zapata (2007) de las *características dinámicas* de los EP permite relacionar la vista comportamental del sistema. Estas características se representan mediante los procesos y flujos de procesos de un dominio, los cuales se relacionan con conceptos que se definen en las relaciones estructurales.

Relación dinámica: La relación dinámica (RD) se compone de una triada de dos conceptos y una relación dinámica (mediante el nodo *relación dinámica* y un verbo de acción en su notación formal). La definición lingüística de la relación dinámica se basa en el rol semántico *agente*, aquella persona o rol que realiza la acción en el primer concepto y el objeto sobre el que recae la acción en el segundo concepto (Zapata, 2007). Por ejemplo, la relación dinámica

"profesor crea curso" en la Tabla 4.5, siendo el "profesor" el primer concepto, cuyo rol semántico es un agente realizando la acción, "crea" el verbo de acción y "curso" el segundo concepto (una clase) sobre el que recae la acción. Las funciones, procesos, actividades u operaciones atómicas se deben representar con una relación dinámica.

Agrupación de RDs: Las RDs permiten especificar una RD general, que involucra otras acciones dentro de un *marco*. Por ejemplo la RD general "profesor gestiona curso", permite detallar las acciones "profesor crea, actualiza y elimina curso" en la Tabla 4.5, también se presenta una segunda opción para explicitar las líneas de las triadas en esta agrupación.

Tabla 4.5. Relación dinámica. Basado en Chaverra (2011) y Zapata (2012)

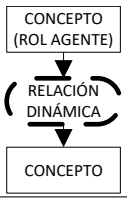
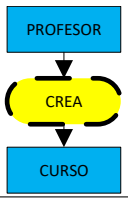
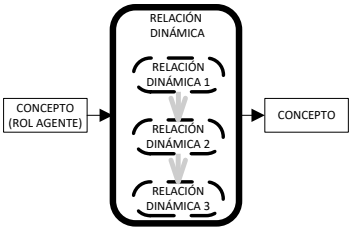
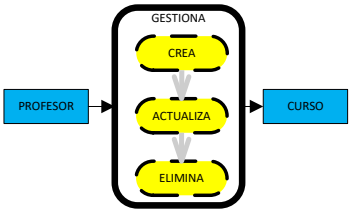
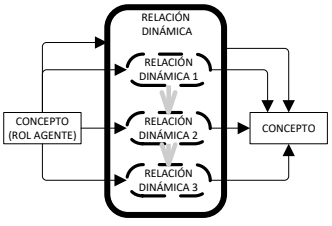
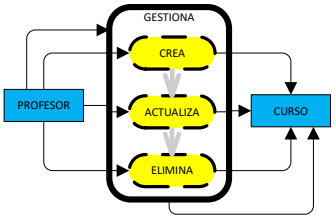
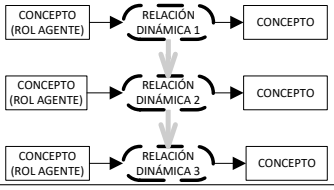
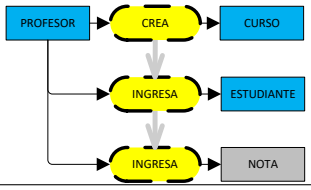
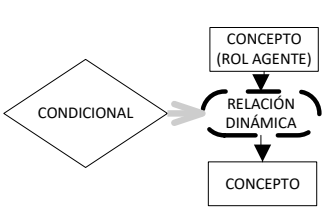
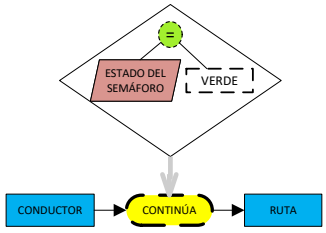
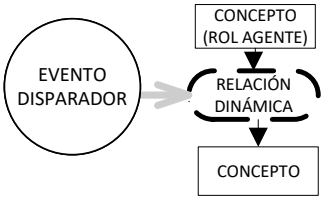
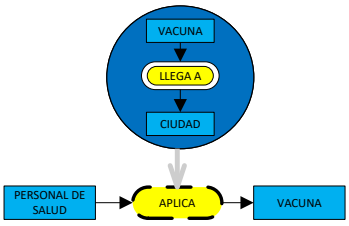
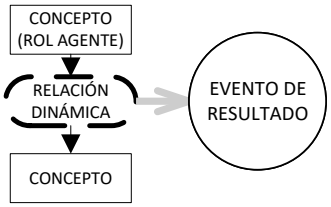
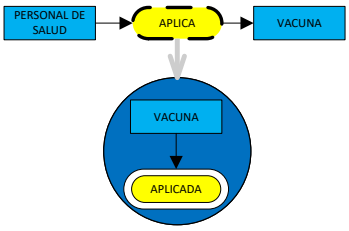
Descripción	Regla en EP	Ejemplo
Relación dinámica		
Agrupación de RDs (1)		
Agrupación de RDs (2)		
Causa-efecto: flujo de RDs		

Tabla 4.5. (Continuación) Relación dinámica. Basado en Chaverra (2011) y Zapata (2012)

Descripción	Regla en EP	Ejemplo
Causa-efecto: flujo desde un condicional		
Causa-efecto: flujo desde un evento disparador		
Causa-efecto: fin de un flujo de RDs		

Causa-efecto en RD: El flujo o secuencia de relaciones dinámicas denotan una causa y un efecto mediante el enlace de *implicación* (color gris) de la notación de EP.

- **Flujo de RDs:** Para representar el flujo de relaciones dinámicas, se utiliza el enlace *implicación* desde una relación dinámica hacia la otra, con el fin de dar obligatoriedad entre las acciones, es decir, una relación *causa-efecto*. Se presenta la regla y el ejemplo de un flujo de tres triadas de relaciones dinámicas: "profesor crea curso", "profesor ingresa estudiante" y "profesor ingresa nota" en la Tabla 4.5.
- **Flujo desde un condicional:** Para representar el inicio de una relación dinámica o de un flujo de relaciones dinámicas se utiliza el enlace *implicación* desde un *condicional* hacia la relación dinámica. Por ejemplo, el condicional si "estado del semáforo" está en "verde" entonces se puede ejecutar la relación

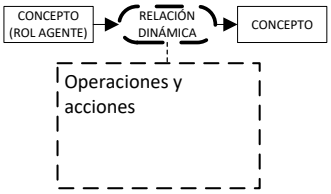
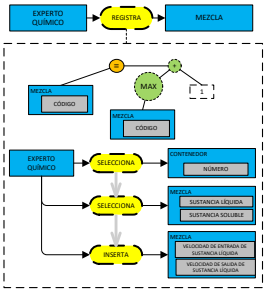
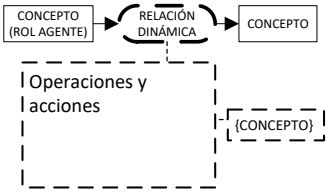
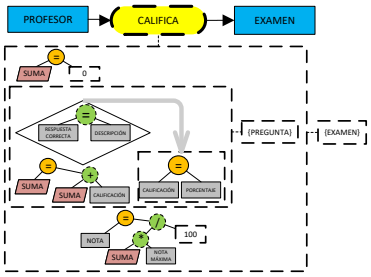
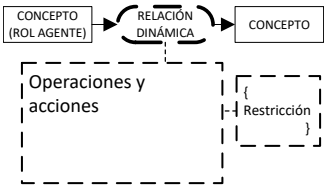
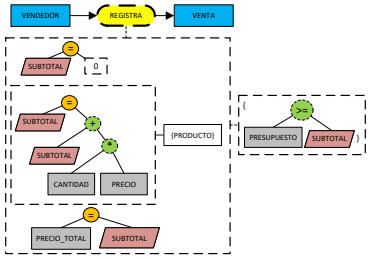
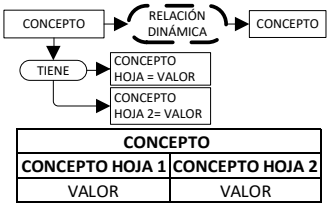
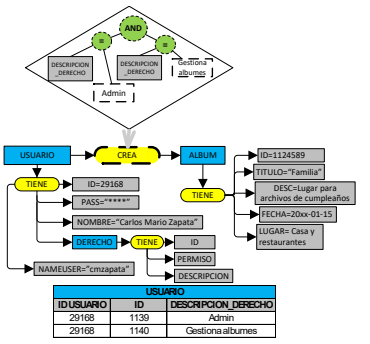
dinámica "conductor continúa ruta" en la Tabla 4.5.

- **Flujo desde un evento disparador:** Para representar el inicio de una relación dinámica o de un flujo de relaciones dinámicas se utiliza el enlace *implicación* desde un *evento disparador* hacia la relación dinámica. Por ejemplo, el evento disparador "vacuna llega a ciudad" entonces la relación dinámica "personal de la salud aplica vacuna" se puede ejecutar en la Tabla 4.5.
- **Fin de un flujo de RDs:** Para representar el fin de una relación dinámica o de un flujo de relaciones dinámicas se utiliza el enlace *implicación* desde la relación dinámica o desde la última relación dinámica (si es un flujo) hacia el *evento de resultado*. Por ejemplo, la relación dinámica "personal de la salud aplica vacuna" y con ella se culmina el proceso que se indica con el evento de resultado "vacuna aplicada" en la Tabla 4.5. Es importante señalar que, al finalizar un flujo, no es obligatorio representar el evento de resultado si el verbo es el mismo que el de la relación dinámica, pues se sobreentiende. En caso de utilizar un verbo diferente, entonces sí se debe representar explícitamente.

Especificación de RD: La especificación de una relación dinámica permite detallar su comportamiento, es decir, las acciones internas (inserta, actualiza, borra, lista, consulta y selecciona) que contiene la relación dinámica. Al interior se permite representar cualquier estructura perteneciente a un EP (Chaverra, 2011). En este caso, se puede considerar paralelamente un caso de uso o una historia de usuario que incluye los detalles de una misma función. Para representar la especificación de una relación dinámica se utiliza el aglutinador *especificación* que se une con la relación dinámica mediante un enlace "nota-valor". Por ejemplo, la especificación de la relación dinámica "experto químico registra mezcla" que incluye diferentes operaciones internas, que permiten lograr el registro de la mezcla (véase la Tabla 4.6). Estas especificaciones se conocen como abstracción 2, en el EP, ya que representan más detalles de las relaciones.

Ciclo en RD: Los ciclos pueden ser de iteración definida o indefinida. Su representación se basa en una *restricción*, la cual, en el código fuente, se traduce en la condición de finalización del ciclo. Esta restricción se puede expresar de dos formas: mediante un concepto o mediante operaciones matemáticas (Chaverra, 2011) como se observa en la Tabla 4.6.

Tabla 4.6. Especificación de relación dinámica. Basado en Chaverra (2011)

Descripción	Regla en EP	Ejemplo
Especificación de RD		
Ciclo usando concepto		
Ciclo usando restricción		
EP ejecutable		

- **Ciclo usando concepto:** Se usa una *restricción con concepto*, cuando se requiere que se realicen las operaciones de la especificación para todos los

conceptos que contengan el mismo nombre del concepto. Por ejemplo, para todas las "preguntas" del "examen" se debe comparar si es correcta y sumar la calificación y para todos los exámenes también se puede utilizar la misma operación (véase la Tabla 4.6).

- **Ciclo usando restricción:** Se usa una *restricción con operación matemática*, cuando se requiere explícitamente la condición, para ejecutar la relación dinámica. Por ejemplo, sólo se ejecuta la relación dinámica "vendedor registra venta" cuando se cumpla que, "presupuesto sea mayor o igual que subtotal" en la Tabla 4.6.

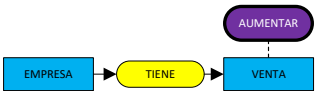
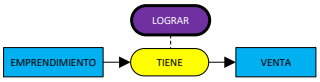
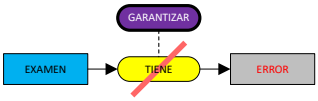
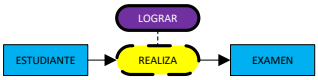
EP ejecutable de una RD: Para representar un *EP ejecutable* de una relación dinámica se utiliza la relación dinámica junto con los conceptos y sus relaciones estructurales. En el *EP ejecutable* se incluyen los valores en en el mismo concepto hoja después del signo "=" y se utiliza una *tabla* para incluir una representación del almacenamiento de estos *valores* en la base de datos (véase la Tabla 4.6). Por ejemplo, la relación dinámica "usuario crea álbum" en la Tabla 4.6, tanto "usuario" como "álbum" son clases, que incluyen conceptos hoja. La clase "usuario" tiene "nombre" y uno de sus valores es "Carlos Mario Zapata", el cual tiene un "id" igual a "29168" y se observa también en su respectiva *tabla*.

4.1.3. Características Télicas

La construcción formal que realiza Vargas (2016), Zapata y Castro (2017) de las características télicas de los EP, permite relacionar los objetivos y los problemas del dominio del sistema mediante relaciones de logro (RL), negación, conceptos en connotación negativa y adverbios.

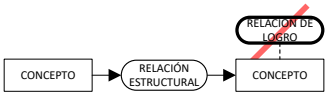
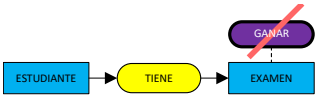
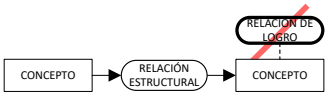
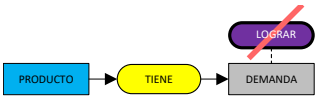
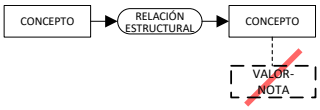
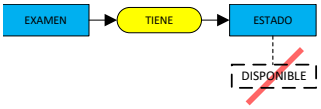
Objetivos: Se presentan diferentes casos de representación de *objetivos* organizacionales y del sistema en el contexto de la educción temprana de requisitos de software como: la RL en concepto, en RE, en RE y concepto negativo, en RD y en RD con adverbio. Las reglas gráficas y ejemplos para estos casos de representación, se encuentran respectivamente en la Tabla 4.7. Es importante aclarar que algunas RL también expresan requisitos o expectativas a nivel operativo; por ejemplo, las RL "lograr que el estudiante realice el examen" y "lograr que el estudiante realice el examen correctamente".

Tabla 4.7. Relación de logro en objetivos. Basado en Vargas (2016)

Descripción	Regla en EP	Ejemplo
RL en concepto		 Lectura del objetivo: aumentar las ventas de la empresa
RL en RE		 Lectura del objetivo: lograr que el emprendimiento tenga ventas
RL en RE y concepto negativo		 Lectura del objetivo: garantizar que el examen no tenga errores
RL en RD		 Lectura de la expectativa: lograr que el estudiante realice el examen
RL en RD con adverbio		 Lectura de la expectativa: lograr que el estudiante realice el examen correctamente

Problemas: Se presentan diferentes casos de representación de *problemas* del dominio del sistema en el contexto de la educción temprana de requisitos de software como se observa en la Tabla 4.8: problema en RL con negación ("el estudiante no gana el examen" y "la demanda del producto no se logra"), en RE con negación ("el profesor no tiene disponibilidad"), en nota-valor con negación ("el examen no está disponible"), en RD con negación ("el profesor no genera el examen eficientemente"), usando valores en connotación negativa en un concepto ("el examen tiene un resultado ineficiente"), usando un concepto negativo en una RE ("el examen tiene demora"), usando adverbio negativo con una RD ("El profesor entrega el examen inadecuadamente").

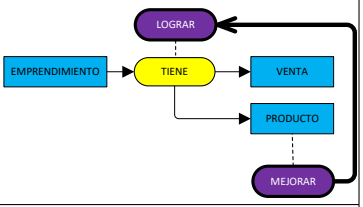
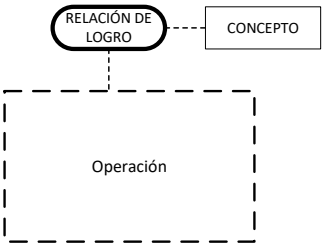
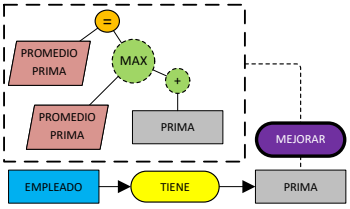
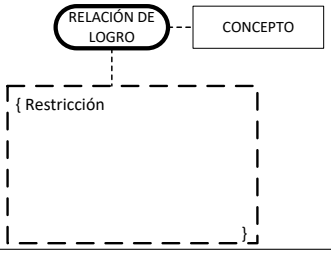
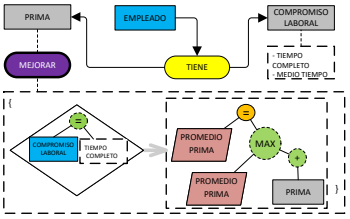
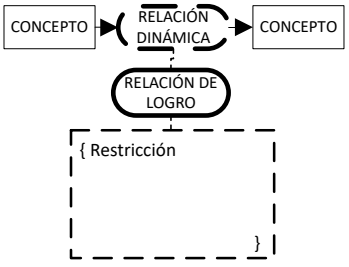
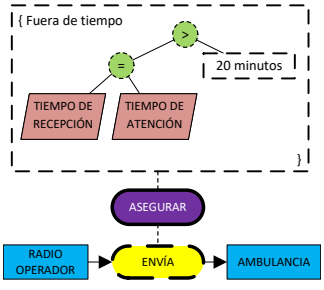
Tabla 4.8. Relación de logro en problemas. Basado en Vargas (2016)

Descripción	Regla en EP	Ejemplo
RL con negación en concepto clase		 Lectura del problema: El estudiante no gana el examen
RL con negación en concepto hoja		 Lectura del problema: La demanda del producto no se logra
RE con negación		 Lectura del problema: El profesor no tiene disponibilidad
Nota-valor con negación		 Lectura del problema: El examen no está disponible
RD con negación		 Lectura del problema: El profesor no genera el examen eficientemente
Concepto con valor negativo		 Lectura del problema: El examen tiene un resultado ineficiente
Concepto negativo		 Lectura del problema: El examen tiene demora
RD con adverbio negativo		 Lectura del problema: El profesor entrega el examen inadecuadamente

Causa-efecto en RL: Para representar el flujo de RL se utiliza el enlace de

implicación (color negro), que se une desde la primera relación de logro hasta la siguiente, indicando una relación *causa-efecto*, donde se requiere cumplir un objetivo para que se cumpla el siguiente. Por ejemplo, el primer objetivo "mejorar producto del emprendimiento" se debe cumplir para que se efectúe el segundo objetivo "lograr que el emprendimiento tenga ventas" en la Tabla 4.9.

Tabla 4.9. Especificación de relaciones de logro. Basado en Vargas (2016) y Zapata y Castro (2017)

Descripción	Regla en EP	Ejemplo
Causa-efecto: flujo de RL		
Especificación de RL		
Restricción de RL en concepto		
Restricción de RL en RD		

Especificación de relación de logro: en este compendio de características télicas de los EP, se propone la especificación de relaciones de logro basados en los indicadores de rendimiento clave (KPI, por sus siglas en inglés) para objetivos estratégicos propuesto por Zapata y Castro (2017). En la representación, se debe incluir una *relación de logro* con una *especificación* mediante el enlace de *concepto-nota*.

La *especificación* debe incluir *operaciones matemáticas* de indicadores o métricas, que permitan medir el cumplimiento de los objetivos o la incidencia de los problemas del dominio. Como ejemplo, la especificación del objetivo "mejorar la prima del empleado" en donde se utiliza la operación de un indicador para observar el promedio de primas en la Tabla 4.9.

Restricción de relación de logro: De acuerdo con Vargas (2011) las relaciones de logro también pueden tener restricciones, que permitan cumplir con el objetivo o reflejar el problema que incide en el incumplimiento del objetivo. Para representar la restricción de RLs, se debe incluir una *relación de logro* con una *restricción* mediante el enlace de *concepto-nota*.

- **Restricción de RL en concepto:** La *restricción* debe incluir operaciones matemáticas de condición, que permitan medir el cumplimiento de los objetivos o la incidencia de los problemas del dominio (véase la Tabla 4.9). Como ejemplo, la restricción del objetivo "mejorar la prima del empleado" cuya operación requiere que se cumpla la condición de que el "compromiso laboral del empleado" sea igual a "tiempo completo", para incluir su prima y a su vez hallar el promedio.
- **Restricción de RL en RD:** La relación de logro también se puede conectar a RDs, como se puede ver en las reglas de la Tabla 4.9. Se considera como ejemplo la restricción que condiciona el objetivo "asegurar que el radio operador envíe la ambulancia" por el tiempo de recepción y atención en la Tabla 4.9.

4.1.4. Características Eventuales

La construcción formal que realizan Zapata (2012) y Noreña (2020) de las características eventuales en los EP, permite relacionar los eventos disparadores y eventos de resultado mediante una relación eventual (REv) y aglutinador

"evento", especificaciones y restricciones.

Evento disparador:

- **Evento sin concepto:** Se puede representar solo con la *relación eventual* dentro del aglutinador *evento* para dar el significado completo del evento. Por ejemplo, el evento natural "nevar" que se representa con la relación eventual "nieva" en la Tabla 4.10.
- **Evento con un concepto:** Se puede representar con una REv junto con un *concepto (clase, concepto hoja, vector, matriz), concepto independiente (variable o vector)*, que se relaciona mediante el enlace de *conexión* dentro del aglutinador *evento*. El concepto debe ser un *actante* experimentador o paciente, quien recibe el efecto del evento y no realiza la acción, sino que se origina por una causa o fuerza (*circunstantes*). Ejemplos de eventos con un concepto en la Tabla 4.10: "mensaje aparece" ("mensaje" es una clase, verde cuando es una clase con múltiples dependencias), "temperatura incrementa" (temperatura" es un concepto hoja), "cantidad surge" ("cantidad" es un vector), "valor baja" ("valor" es una matriz), "cantidad de partícula surge" ("cantidad" es un vector), "valor del dólar baja" ("valor" es una matriz), "presión sube" ("presión" es una variable) y "medida cambia" ("medida" es un vector independiente).
- **Evento con dos conceptos:** Los eventos con dos conceptos se pueden representar con una REv que se relaciona con dos conceptos mediante el enlace de "conexión" dentro del aglutinador *evento*. Los conceptos también deben ser *actantes* experimentadores o pacientes, lo que expresa una ocurrencia por una causa o fuerza y no por un actor. Por ejemplo, el evento "paciente presenta dolor abdominal". También se pueden representar eventos con dos conceptos mediante una REv, la cual se compone del verbo y una preposición. Por ejemplo, el evento "meteorito aparece en Antártida" en la Tabla 4.10.

Evento de resultado: Se representa como un evento con un concepto, cuya REv debe el mismo verbo de la RD conjugado en participio pasado y el *concepto (clase, concepto hoja, vector, matriz, variable o vector independiente)*. Por ejemplo, los eventos "archivo actualizado" ("archivo" es un concepto) y "total de ventas publicada" ("total de ventas" es concepto clase) en la Tabla 4.11. El evento de resultado se puede omitir, ya que al final de un flujo existe uno implícito o hacerlo explícito si el verbo y/o el concepto son diferentes.

Tabla 4.10. Evento disparador. Basado en Zapata (2012) y Noreña (2020)

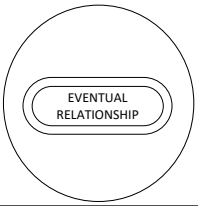
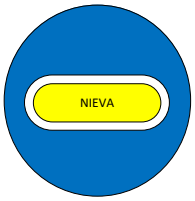
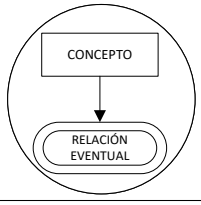
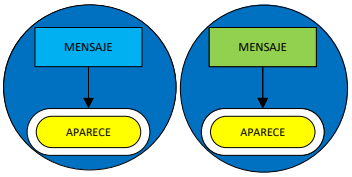
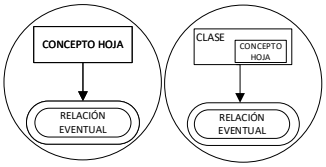
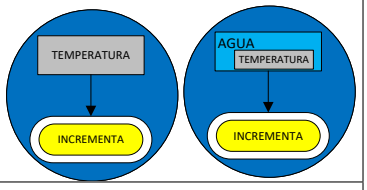
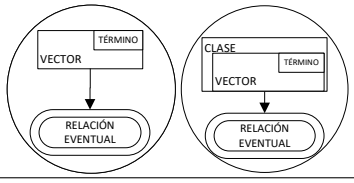
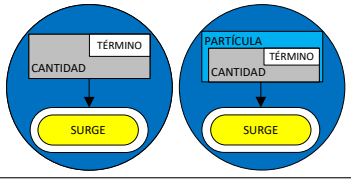
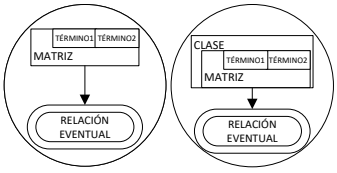
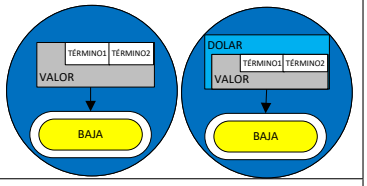
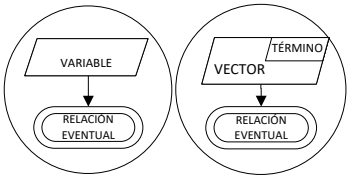
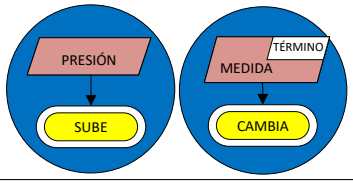
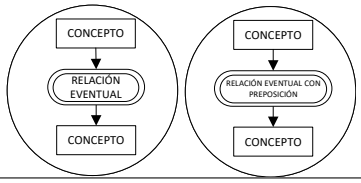
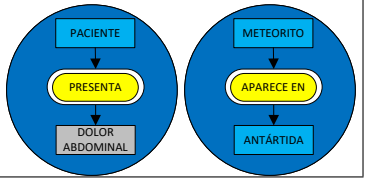
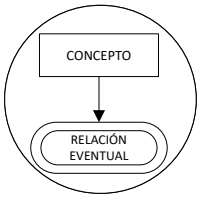
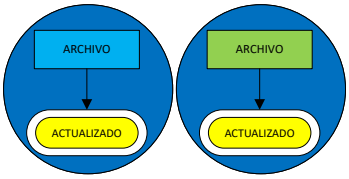
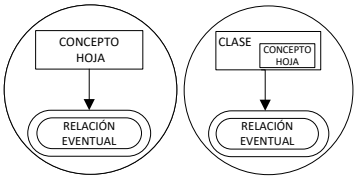
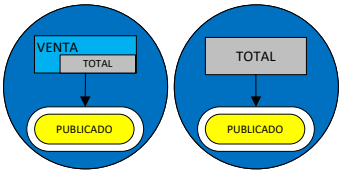
Descripción	Regla en EP	Ejemplo
Evento sin concepto		
Evento con concepto clase		
Evento con concepto hoja		
Evento con vector		
Evento con matriz		
Evento con variable y vector independiente		
Evento con dos conceptos		

Tabla 4.11. Evento de resultado. Basado en Zapata (2012) y Noreña (2020)

Descripción	Regla en EP	Ejemplo
Evento de resultado con clase		
Evento de resultado con concepto hoja		

Causa-efecto en eventos: El flujo o secuencia de eventos denotan una causa y un efecto mediante el enlace de "implicación" (color gris) de la notación de EP.

- **Flujo de eventos:** Para representar el flujo de eventos, se utiliza el enlace *implicación* desde un evento hacia otro, con el fin de dar obligatoriedad entre las acciones automáticas, es decir una relación de *causa-efecto*. Por ejemplo, el primer evento "tiempo de simulación inicia" se debe cumplir para que se efectúe el segundo verbo "huracán aparece en radar", consecuentemente la "alerta del huracán surge" en la Tabla 4.12. Es importante aclarar que los eventos también pueden ser asincrónicos, por lo que pueden ocurrir de forma paralela o sin seguir una secuencia obligatoria.
- **Flujo desde un condicional:** Para representar el inicio de un evento disparador o de un flujo de eventos se utiliza el enlace *implicación* desde un *condicional* (el condicional se considera un tipo de evento disparador) hacia el evento. Por ejemplo, el condicional si "simulación" es igual a "encendida" dispara el evento "tiempo de simulación inicia" en la Tabla 4.12.
- **Flujo desde un evento disparador:** Para representar el inicio de una RD o de un flujo de RDs se utiliza el enlace *implicación* desde un *evento disparador* hacia la RD. Por ejemplo, el evento disparador "vacuna llega a ciudad" dispara la relación dinámica "personal de la salud aplica vacuna" en la Tabla 4.12.

- **Fin de un flujo usando evento de resultado:** Para representar el fin de una RD o de un flujo de RDs se utiliza el enlace *implicación* desde la RD o desde la última RD (si es un flujo) hacia el evento de resultado. Por ejemplo, la RD "personal de la salud aplica vacuna" y con ella se culmina el proceso que se indica con el evento de resultado "vacuna aplicada" en la Tabla 4.12.

Tabla 4.12. Causa-efecto en eventos y condicionales. Basado en Zapata (2012) y Noreña (2014)

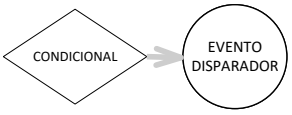
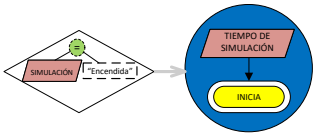
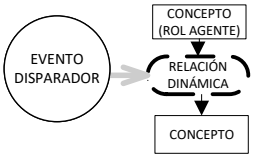
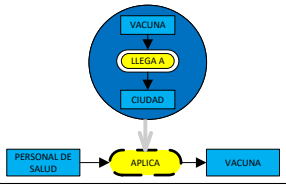
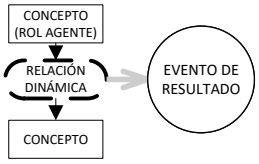
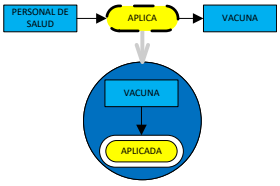
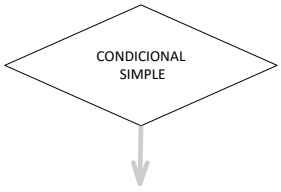
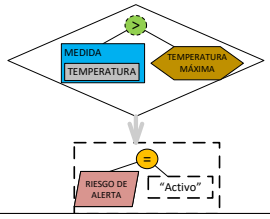
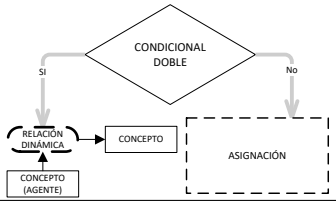
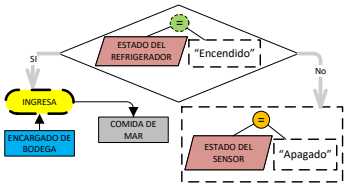
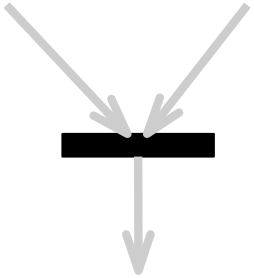
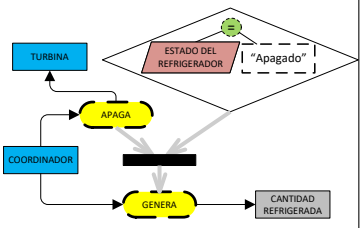
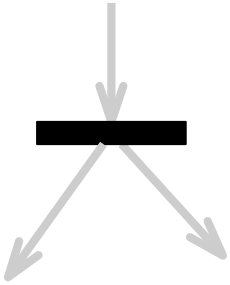
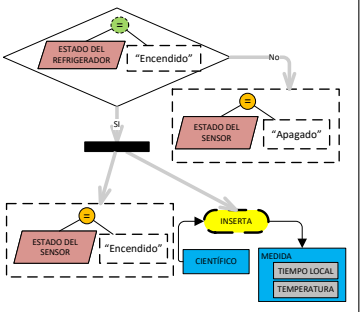
Descripción	Regla en EP	Ejemplo
Causa-efecto: flujo de eventos		
Causa-efecto: desde un condicional		
Causa-efecto: con evento disparador		
Causa-efecto: fin de un flujo		
Condicional simple		
Condicional doble		

Tabla 4.12. (Continuación) Causa-efecto en eventos y condicionales. Basado en Zapata (2012) y Noreña (2014)

Descripción	Regla en EP	Ejemplo
Conjunción		
Disyunción		

- **Condicional simple:** Para representar el flujo de un *condicional simple* se conecta el enlace *implicación* hacia el elemento (relación dinámica, evento o asignación; es importante recordar que un condicional es un tipo de evento disparador). Por ejemplo, el condicional simple si "medida de temperatura" es mayor que "temperatura máxima" entonces "alerta de riesgo" igual a "activo" (asignación) en la Tabla 4.12.
- **Condicional doble:** Para representar el flujo de un *condicional doble*, se conectan los dos enlaces *implicación* (si y no) hacia los elementos (relación dinámica o asignación). Por ejemplo, el condicional doble si "estado del refrigerador" es igual a "apagado" entonces "estado del sensor" igual a "encendido" (asignación) sino "estado del sensor" igual a "apagado" en la 4.12.
- **Conjunción:** Para representar una conjunción, se conectan los enlaces *implicación* al enlace *conjunción/disyunción*. Por ejemplo, el condicional si "estado del refrigerador" es igual a "apagado" y la RD "coordinador apaga turbina"

entonces “coordinador genera cantidad refrigerada” en la 4.12.

- **Disyunción:** Se conecta una *implicación* que se puede bifurcar mediante el enlace *conjunción/disyunción* para generar varias acciones. Por ejemplo, la conjunción que se utiliza en el condicional doble si “estado del refrigerador” es igual a “on” entonces “estado del sensor” igual a “encendido” y “científico inserta tiempo local” y la “temperatura de medida” en la 4.12.

Especificación de eventos: La *especificación* de un evento permite detallar su comportamiento, es decir, las operaciones y acciones automáticas internas que tiene el evento (insertar, actualizar, borrar, buscar, seleccionar y listar). Al interior se permite representar cualquier estructura perteneciente a un EP, los cuales usualmente se determinan por *operaciones matemáticas* las cuales pueden ser básicas o complejas (Noreña-Cardona, 2020). Para representar la especificación de un evento se utiliza el aglutinador *especificación* que se une con el *evento* mediante un enlace *nota-valor*. Por ejemplo, la especificación del evento “beneficio de comida de mar baja” que incluye la operación automática interna “inserta comida de mar” sin alguien que realice la acción e incluye una operación matemática para realizar la acción (véase la Tabla 4.13).

Restricción de eventos: La restricción de un evento permite condicionar la ejecución del evento. Para representar la restricción de una relación dinámica se utiliza el aglutinador *restricción* que se une con el evento mediante un enlace *nota-valor*. Por ejemplo, la restricción del evento “medida inicia” (véase la Tabla 4.13; Noreña-Cardona, 2020).

Evento de tiempo: Los eventos de tiempo (también se conocen como temporizadores) se representan con restricciones donde se incluyen ciclos de incrementos en el tiempo, que se pueden aplicar a cualquier formato y unidad de tiempo. Por ejemplo, el evento “tiempo pasa” tiene una *restricción* que se condiciona por una “marca de tiempo” (es una variable que puede tener dos valores: siga o pare; para controlar acciones que se basan en el tiempo) y el “tiempo de simulación” para incrementar el tiempo de 1 a 1 minuto (véase la Tabla 4.13).

Ciclo en evento: La restricción de los ciclos para representar operaciones automáticas en eventos se define de forma similar a como se representan en relaciones dinámicas, mediante un concepto y mediante operaciones matemáticas. Se une la *restricción* al *evento*, como se observa en la regla gráfica de la Tabla 4.13.

Tabla 4.13. Especificación de eventos. Basado en Noreña (2020)

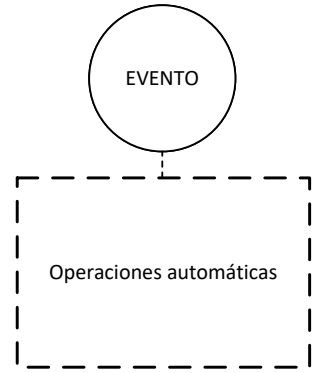
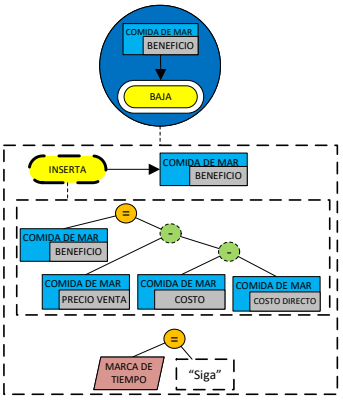
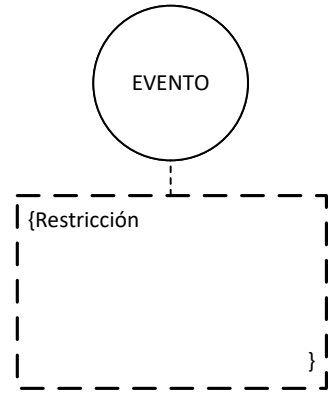
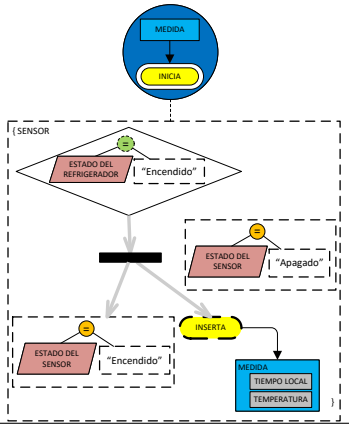
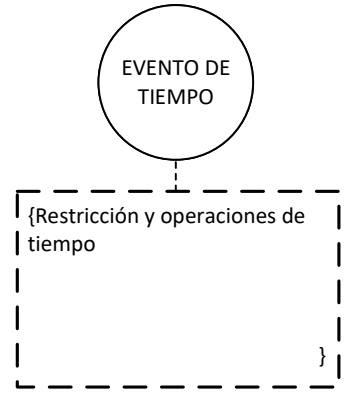
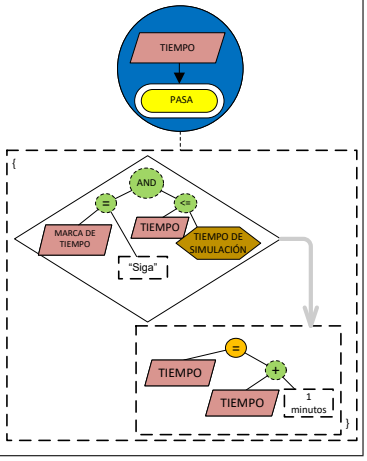
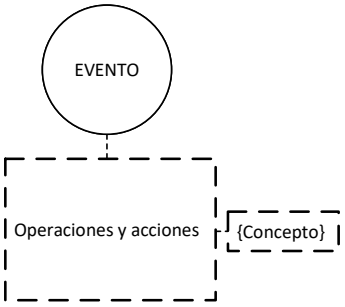
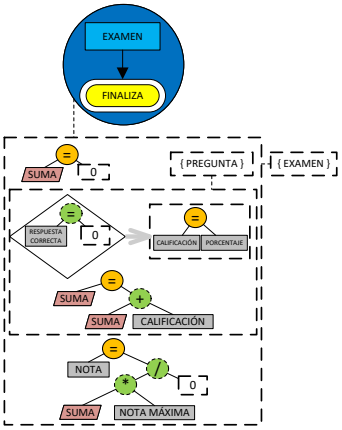
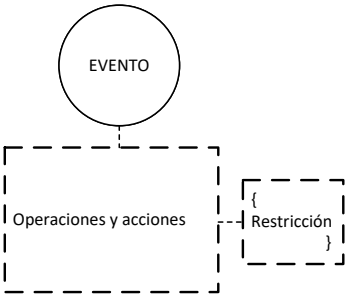
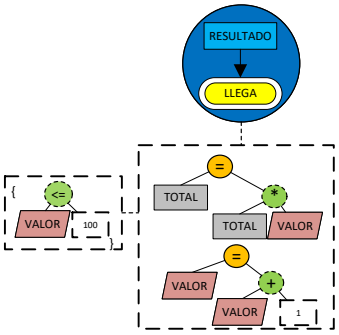
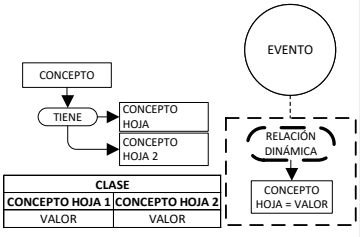
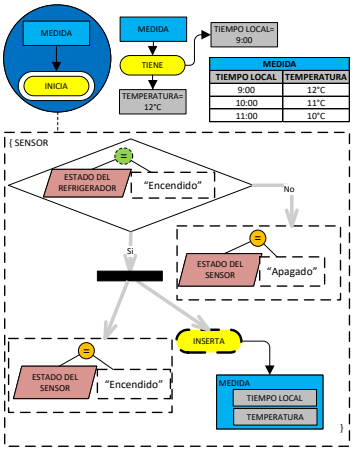
Descripción	Regla en EP	Ejemplo
Especificación de un evento		
Restricción de un evento		
Restricción de un evento de tiempo		

Tabla 4.13. (Continuación) Especificación de eventos. Basado en Noreña (2020)

Descripción	Regla en EP	Ejemplo
Ciclo usando concepto		
Ciclo usando restricción		
EP ejecutable de un evento		

- **Ciclo usando concepto:** Se usa una *restricción con concepto*, cuando se requiere que se realicen las operaciones de la especificación para todos los conceptos que contengan el mismo nombre del concepto. Por ejemplo, para un proceso automático de revisión de un examen, las "preguntas" se debe comparar si es correcta y sumar la calificación, esto sucede cuando cada "examen finaliza" (véase la Tabla 4.13).
- **Ciclo usando restricción:** Se usa una *restricción con operación matemática*, cuando se requiere explícitamente la condición, para ejecutar la relación dinámica (acción automática sin actor) dentro del evento. Por ejemplo, el evento "resultado llega" cuya especificación involucra una operación matemática, se cumpla cuando, "valor sea menor o igual que 100" en la Tabla 4.13.

EP ejecutable de un evento: Para representar un *EP ejecutable* de un evento, se utiliza el evento con su RD sólo con el segundo concepto dentro de una *especificación* o *restricción*. También, se presentan los conceptos, sus REs y conceptos hoja que se almacenan a partir de una acción automática. En el *EP ejecutable* se incluyen los *valores* en el mismo concepto después del signo "=" y se utiliza una *tabla*, para incluir una representación del almacenamiento de estos valores en la base de datos. Por ejemplo, el evento "medida inicia" en la Tabla 4.13, el cual incluye los datos "tiempo local" y "temperatura" de la "medida". Estos datos se insertan automáticamente en la base de datos mediante el evento.

4.2. Equivalencias

Los EP tienen equivalencias con esquemas conceptuales, artefactos ágiles y código fuente gracias a las características estructurales, dinámicas, télicas y eventuales que componen su notación.

4.2.1. Equivalencias de EP a esquemas conceptuales

Equivalencia de lenguaje controlado a EP: Según las reglas de la notación formal de los EP, los esquemas tienen una traducción equivalente al lenguaje natural controlado. Esto permite traducir dominios definidos textualmente y,

en este caso, posibilita la representación gráfica mediante los EP. Las reglas del lenguaje controlado también garantizan una correcta lectura de las tríadas en los EP. En la Tabla 4.14 se presentan las reglas desde la notación formal a RE y RD de los EP con sus respectivos ejemplos (Zapata, 2007). Por ejemplo, la regla de la relación estructural se ilustra con "estudiante tiene nombre", implicando que de forma textual se pudo identificar como "estudiante incluye nombre", "tiene nombre" o "posee nombre". También se pueden encontrar frases con RD, es decir, con verbos dinámicos. Por ejemplo, para la regla de la RD se representa con "profesor ingresar nota" siendo "ingresar" el verbo dinámico. Estas relaciones también incluyen implicaciones cuando tiene un cumplimiento de condiciones y acciones para proceder a la siguiente acción. Por ejemplo, si "estudiante presenta examen" entonces "profesor ingresa nota". Si es la fecha del examen entonces el "estudiante presenta examen", en caso contrario "profesor propone ejercicio" (véase la Tabla 4.14).

La regla de la notación formal para RL se presenta en la Tabla 4.14 y se ejemplifica mediante el objetivo "lograr que el estudiante realice el examen correctamente". Las RL se pueden asociar con un concepto, una RE (tiene) o una RD para expresar objetivos y requisitos/expectativas.

Tabla 4.14. Equivalencia de lenguaje controlado a EP. Basado en Zapata (2007)

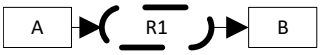
Notación formal	**Regla en EP**	**Ejemplo**
A<ES>B	 A es una especie de B A es un tipo de B A es una clase de B	 ESTUDIANTE es una clase de PERSONA
A<TIENE>B	 A incluye B A contiene B A posee B A esta compuesto por B A está formado por B A se divide en B	 ESTUDIANTE contiene NOMBRE ESTUDIANTE posee NOMBRE
A<R1>B	 <R1>puede ser cualquier verbo dinámico	 INGRESAR es un verbo dinámico

Tabla 4.14. (Continuación) Equivalencia de lenguaje controlado a EP. Basado en Zapata (2007)

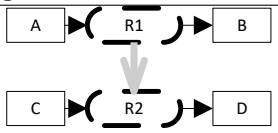
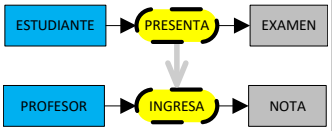
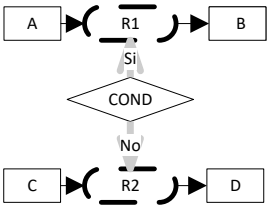
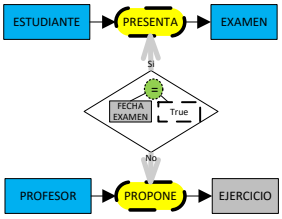
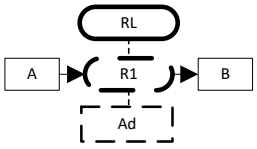
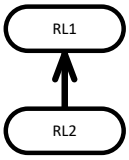
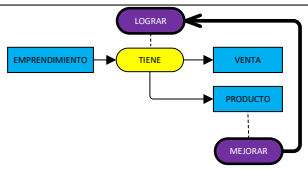
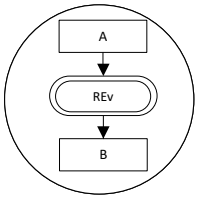
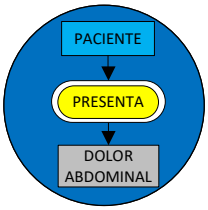
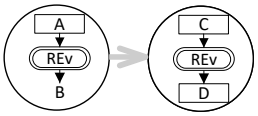
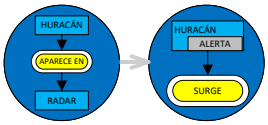
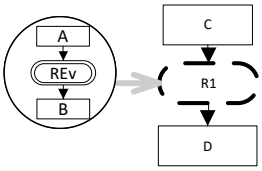
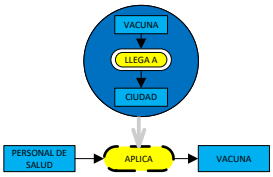
Notación formal	Regla en EP	Ejemplo
A<ES>B C<R2>D, <SI>A <R1>B	 Si A <R1>B entonces C<R2>D Dado que A <R1>B, C<R2>D Luego de que A <R1>B, C <R2>D	 Si ESTUDIANTE PRESENTA EXAMEN entonces PROFESOR INGRESA NOTA
A <ES>B <SI>{COND} <ENTONCES>A <R1>B, <SINO>C<R2>D	 {COND} condición expresada en términos de conceptos. <R1>y <R2>son verbos dinámicos. <SINO>es opcional.	 Si FECHA EXAMEN = True entonces ESTUDIANTE PRESENTA EXAMEN Sino PROFESOR PROPONE EJERCICIO
<RL><A, TIENE, R1+Ad>	 <RL>verbo de logro <A>concepto (opcional negativo) <TIENE>RE <R1>verbo dinámico +<Ad>Adverbio opcional (connotación negativa para expresar problema)	 LOGRAR que el ESTUDIANTE realice el EXAMEN correctamente
<RL2><SI><RL1>	 Para <RL2> se debe <RL1>	 Para LOGRAR tener ventas se debe MEJORAR el producto

Tabla 4.14. (Continuación) Equivalencia de lenguaje controlado a EP. Basado en Zapata (2007)

Notación formal	Regla en EP	Ejemplo
A <REv>B	 A es concepto opcional <REv>verbo eventual sin rol agente B concepto opcional	 PACIENTE PRESENTA DOLOR ABDOMINAL
A <REv>B, C <REv>D	 Si A <REv>B entonces C <REv>D	 Si HURACÁN APARECE EN RADAR Entonces ALERTA SURGE
A <REv>B, C <R1>D	 Si A <REv>B entonces C <R1>D	 Si VACUNA LLEGA A CIUDAD Entonces PERSONAL DE SALUD APLICA VACUNA

Para representar la secuencia de las RL, en el lenguaje natural se observa como una secuencia en la que se debe trabajar primero en el objetivo situado más abajo para alcanzar el que está arriba, o resolver el problema de base para poder atender el siguiente en la cadena. La notación formal para obtener los eventos desde el lenguaje natural se presenta con verbos sin rol de agente humano, es decir, que son acciones automáticas que se pueden desencadenar por agentes, por ejemplo, de inteligentes de inteligencia artificial (IA) o eventos naturales. Las reglas se presentan en la Tabla 4.14 y se ilustran mediante el evento “paciente presenta dolor abdominal”; es importante aclarar que el paciente recibe el efecto del evento, mas no lo genera. De igual manera, un evento puede disparar otro evento o RD, generando una secuencia como se ejemplifica en los eventos Si

"huracán aparece en radar" entonces "alerta surge" y si "vacuna llega a ciudad" entonces "personal de salud aplica vacuna".

Equivalencia de lenguaje natural procesado por LLM a EP: Los modelos de lenguaje de gran escala (LLM, *Large Language Models*, por sus siglas en inglés) son modelos de IA capaces de procesar y generar información a partir de lenguaje natural, y en algunos casos, de forma multimodal (incluyendo texto, imágenes, audio o video). A partir de las reglas de notación formal para lenguaje natural descritas en la Tabla 4.14, se propone en la Tabla 4.15 un conjunto de reglas que permiten representar EP a partir del lenguaje procesado por los LLM. Esto facilita la extracción automática de elementos del dominio y la generación sugerida de notación textual en formato EP. Asimismo, ofrece la posibilidad de automatizar la creación de esquemas como paso previo a su transformación en código fuente.

Tabla 4.15. Equivalencia de lenguaje natural procesado por LLM a EP

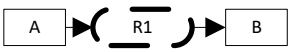
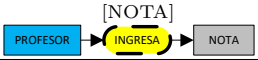
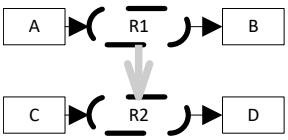
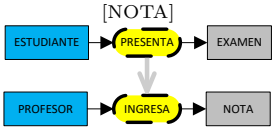
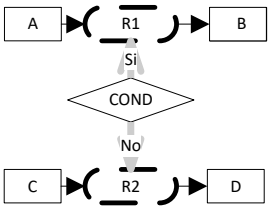
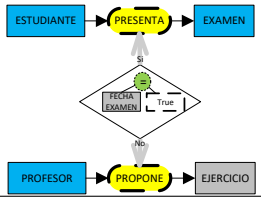
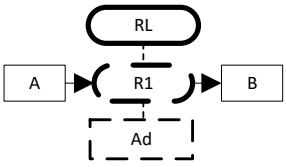
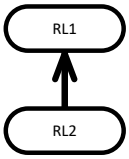
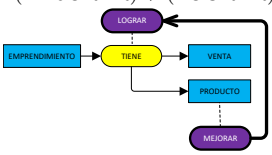
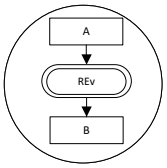
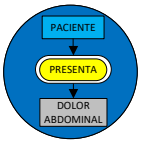
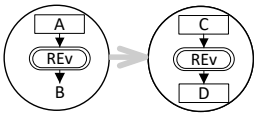
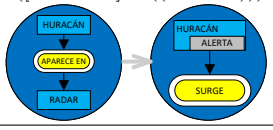
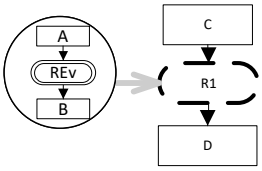
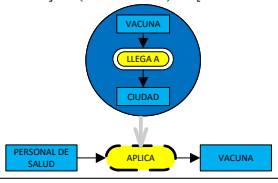
Notación Textual	**Regla en EP**	**Ejemplo**
[A]→(ES)→[B]		[ESTUDIANTE]→(ES)→ [PERSONA] 
[A]→(TIENE)→ [B]		[ESTUDIANTE]→(TIENE)→ [NOMBRE] 
[A]→(R1)→[B]		[PROFESOR]→(INGRESA)→ [NOTA] 
[A]→(R1)→[B] ⇒ [C]→(R2)→[D]		[ESTUDIANTE]→(PRESENTA)→ [EXAMEN] ⇒ [PROFESOR]→(INGRESA)→ [NOTA] 
<COND>SI ⇒ [A]→(R1)→[B] SINO ⇒ [C]→(R2)→[D]		<FECHA EXAMEN = True> Si ⇒ [ESTUDIANTE]→(PRESENTA)→ [EXAMEN] Sino ⇒ [PROFESOR]→(INGRESA)→ [NOTA]

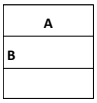
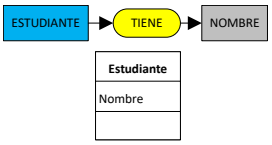
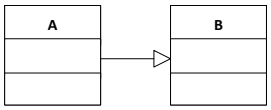
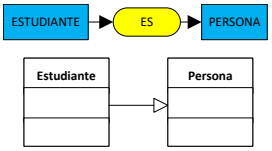
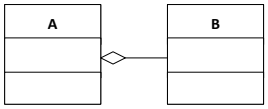
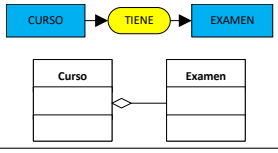
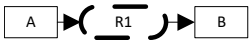
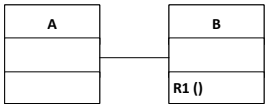
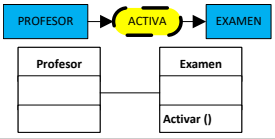
Tabla 4.15. (Continuación) Equivalencia de lenguaje natural procesado por LLM a EP

Notación formal	Regla en EP	Ejemplo
		
(RL) ⇒ [A]→(R1[Ad])→[B] (RL) ⇒ [A]→(TIENE)→[B]		(LOGRAR) ⇒ [ESTUDIANTE]→(REALIZA [CORRECTAMENTE])→ [EXAMEN] 
(RL2) ⇒ (RL1)		(MEJORAR)⇒ (LOGRAR) 
([A]→((REv))→[B])		([PACIENTE]→((PRESENTA))→ [DOLOR ABDOMINAL]) 
([A]→((REv))→[B]) ⇒ ([C]→((REv2))→[D])		([HURACÁN]→((APARECE EN))→ [RADAR]) ⇒ ([ALERTA]→ ((SURGE))) 
([A]→((REv))→[B]) ⇒ [C]→(R1)→[D]	 Si A <REv>B entonces C <R1>D	([VACUNA]→((LLEGA A))→ [CIUDAD]) ⇒ ([PERSONAL DE SALUD]→(APLICA)→[VACUNA]) 

Los siguientes esquemas conceptuales hacen parte del lenguaje de modelado unificado UML (OMG, 2011).

Equivalencia de EP al diagrama de clases: Las *clases* son representaciones de objetos, que implican nombre, atributos (propiedades), métodos (comportamientos) y tipos de relaciones de las clases: generalización (cuando una clase hija o subclase puede heredar los atributos y operaciones de la clase padre o superclase), asociación (cuando tiene una relación de uso entre dos clases) y agregación (cuando una clase contiene a otra, en este relación puede surgir una composición cuando hay una obligatoriedad para la supervivencia de otra clase). Las reglas de equivalencias entre EP y diagramas de clase se presentan en la Tabla 4.16 (Zapata, 2007).

Tabla 4.16. Equivalencia de EP al diagrama de clases. Basado en Zapata (2007)

EP	Diagrama de clases	Ejemplo
		
		
		
		

Desde los EP, las *clases* se identifican con el color azul y sus atributos con el color gris, mediante la relación estructural “tiene”. Por ejemplo, el “estudiante tiene nombre”, donde “estudiante” es la clase y “nombre” el atributo. Una relación de generalización desde el EP se expresa mediante la relación estructural

"es", lo que indica que la subclase tiene características de la superclase. Por ejemplo, "estudiante es persona" donde "estudiante" es la subclase y "persona" la superclase en la Tabla 4.16. También una clase puede ser de color verde cuando tiene varias dependencias estructurales hacia ella. Por otro lado, si la relación estructural está concatenando dos clases mediante la relación "tiene", se traduce al diagrama de clases como una agregación. Por ejemplo, si "curso tiene examen" entonces "curso" agrega al "examen" por lo que el rombo se representa en la clase que agrega a la otra (véase la Tabla 4.16). Para representar los métodos, se debe incluir el verbo de la relación dinámica en la clase que recae la acción. Por ejemplo, si la relación dinámica es "profesor activa examen" ,el verbo "activar" se incluye en la clase "examen" (véase la Tabla 4.16). En este mismo sentido, se puede observar la relación de asociación al indicar que "profesor" realiza una acción en "examen" trazando la línea de la clase "profesor" a la clase "examen".

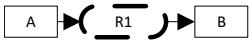
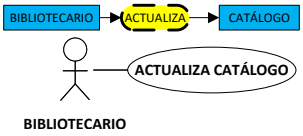
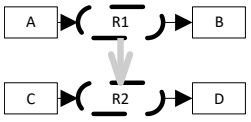
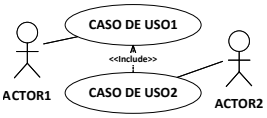
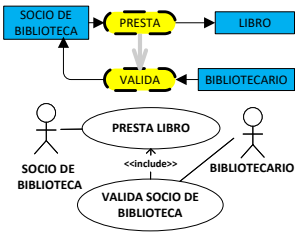
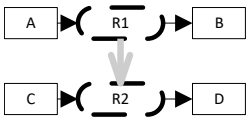
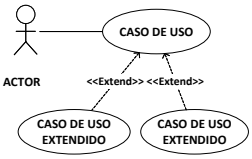
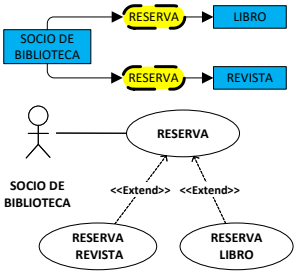
Equivalencia de EP al diagrama de casos de uso: Los *casos de uso* representan la interacción de los actores con las acciones o comportamientos que realizan en un sistema de software. Es por ello que desde los EP se consideran las relaciones dinámicas, ya que implican la acción de los actores del sistema (Zapata *et al.*, 2007b). Un ejemplo de la equivalencia a casos de uso es, si la relación dinámica "bibliotecario actualiza catálogo" entonces, se representa el actor "bibliotecario" y la acción "actualiza catálogo" se incluye en el óvalo del caso de uso (véase la Tabla 4.17).

La relación *include* en los casos de uso indica obligatoriedad en la secuencia de una acción, por lo que desde el esquema una implicación en relaciones dinámicas es correspondiente con esta secuencia. Por ejemplo, si un "socio de biblioteca presta libro", se requiere que el bibliotecario valide al socio de biblioteca. Por ello, se representan los actores con los casos de uso y en el enlace se incluye la etiqueta «include» en dirección al caso de uso "presta libro" (véase la Tabla 4.17).

La relación *extend* extiende la funcionalidad de un caso de uso. Por ejemplo, si "socio de biblioteca reserva libro" y "socio de biblioteca reserva revista" ambas acciones son una reserva por lo que se representa en un caso de uso y se extienden los dos tipos de reservas "reserva libro" y "reserva revista" (véase la Tabla 4.17). En los enlaces se incluye la etiqueta «extend» en dirección hacia el caso de uso

que se extiende.

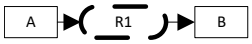
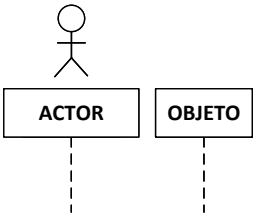
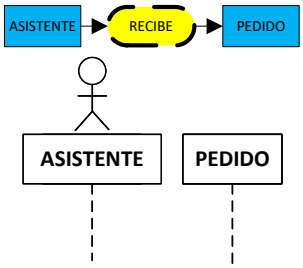
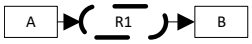
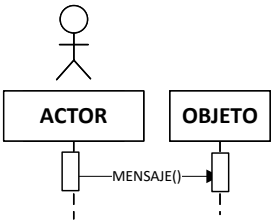
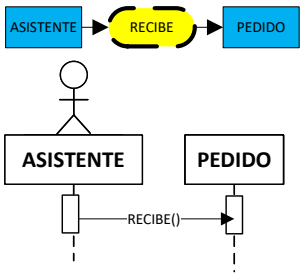
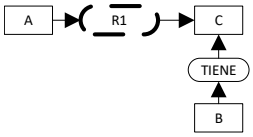
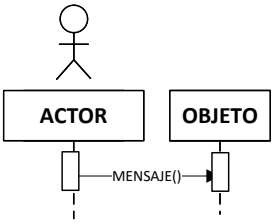
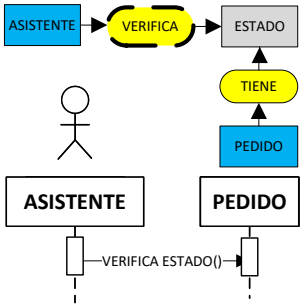
Tabla 4.17. Equivalencia de EP al diagrama de casos de uso. Basado en Zapata (2007)

EP	Diagrama de casos de uso	Ejemplo
		
		
		

Equivalencia de EP al diagrama de secuencia: El *diagrama de secuencia* permite observar la vista comportamental de un sistema de software a partir de la secuencia de acciones que se dan entre los actores y objetos. En la Tabla 4.18 se presentan las reglas de equivalencia entre los EP y el diagrama de secuencias (Zapata y Garcés, 2008).

En el ejemplo, “asistente recibe pedido” permite la representación de las líneas de vida del actor “asistente” y el objeto “pedido”, además, el verbo dinámico “recibe” se usa como mensaje desde el actor hasta el objeto en la Tabla 4.18. Esto también aplica para atributos de un objeto, por ejemplo, la relación dinámica “asistente verifica estado del pedido”, el atributo “estado” se representa en el mensaje junto al verbo dinámico “verifica” y la línea se vincula al objeto “pedido” (véase la Tabla 4.18).

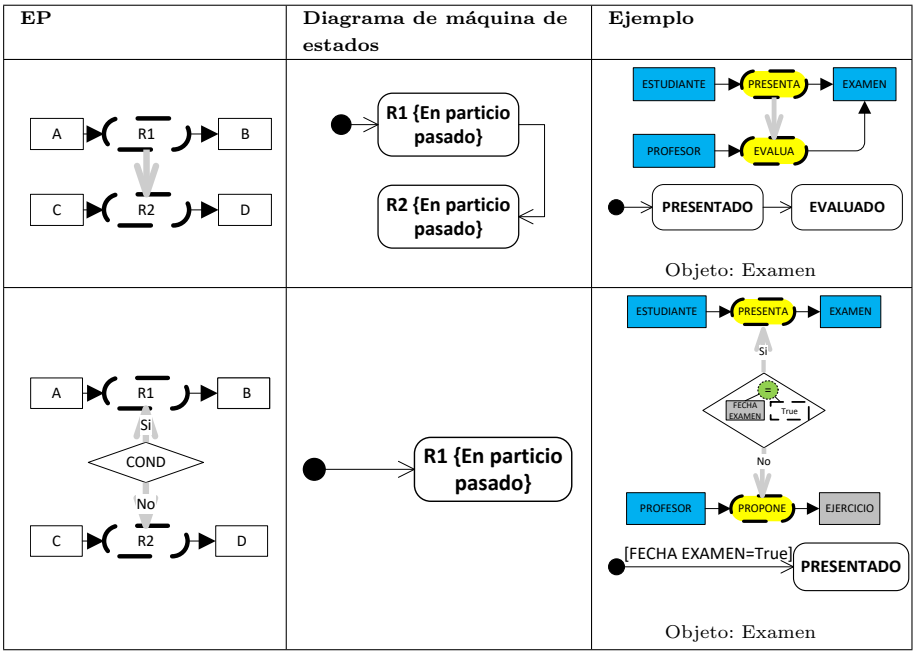
Tabla 4.18. Equivalencia de EP al diagrama de secuencias. Basado en Zapata (2007)

EP	Diagrama de secuencias	Ejemplo
		
		
		

Equivalencia de EP al diagrama de máquina de estados: El *diagrama de máquina de estados* permite la vista de los estados de cada objeto. Las reglas de equivalencia entre los EP y el diagrama de máquina de estados se presentan en la Tabla 4.19 (Zapata, 2007). Por ejemplo, desde las relaciones dinámicas "estudiante presenta examen" y "profesor evaluar examen", se definen los estados del objeto estado a partir de los verbos dinámicos "presenta" y "evalúa". De este modo los estados resultantes son "presentado" y "evaluado".

Por otro lado, cuando una relación dinámica está sujeta a una condición, esta se debe incluir en el enlace que conecta los estados. Por ejemplo, ante la condición "fecha de examen = true", se desencadena la acción "estudiante presenta examen". En el diagrama de máquina de estados, primero se representa la condición y, a continuación, el estado correspondiente con el verbo dinámico "presentado" (véase la Tabla 4.19).

Tabla 4.19. Equivalencia de EP al diagrama de máquina de estados. Basado en Zapata (2007)

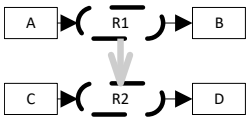
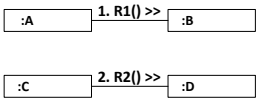
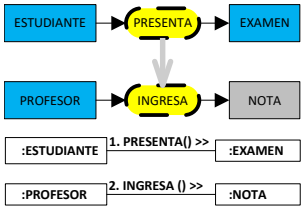
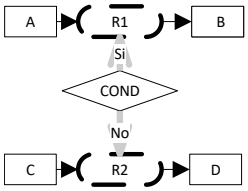
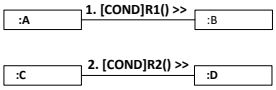
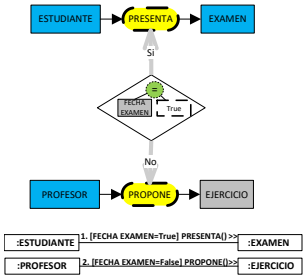


Equivalencia de EP al diagrama de comunicaciones: El *diagrama de comunicaciones* permite la interacción entre actores y objetos. Las reglas de equivalencia entre los EP y el diagrama de comunicaciones se presentan en la Tabla 4.20 (Zapata, 2007). Por ejemplo, desde las triadas de las relaciones dinámicas "estudiante presenta examen" y "profesor ingresa nota", los elementos se representan en el mismo orden y los verbos dinámicos "presenta" e "ingresa" se presentan en la línea de comunicación (véase la Tabla 4.20).

También, si la relación dinámica tiene una condición, esta se debe incluir en el enlace de comunicación dentro de un [corchete] junto con los verbos dinámicos,

de acuerdo con la regla de notación desde UML, por ejemplo, para la condición "fecha de examen = true" entonces "estudiante presenta examen" sino "profesor propone ejercicio", las relaciones dinámicas se representan en el mismo orden y en los dos enlaces de comunicación se incluye las condiciones y verbos "[fecha examen= true] presenta()" y "[fecha examen= false] propone()" (véase la Tabla 4.20).

Tabla 4.20. Equivalencia de EP al diagrama de comunicaciones. Basado en Zapata (2007)

EP	Diagrama de comunicaciones	Ejemplo
		 Objeto: Examen
		 Objeto: Examen

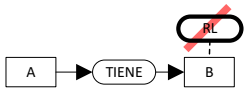
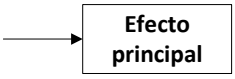
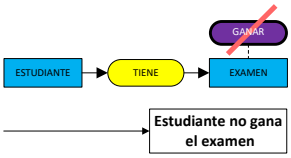
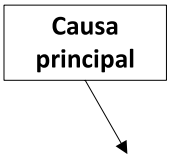
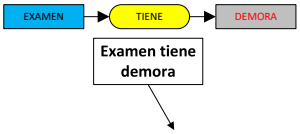
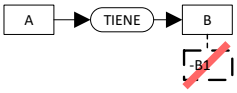
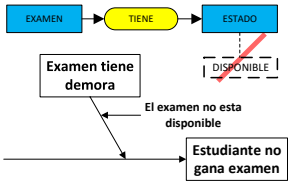
A continuación se presentan equivalencias de los EP a otros esquemas conceptuales.

Equivalencia de EP al diagrama causa-efecto: El *diagrama causa-efecto* permite visualizar un problema o efecto principal y sus causas. Todas las formas de representación de un problema en EP (véase la Tabla 4.8) son equivalentes a las causas y efectos del diagrama causa-efecto, por lo que el efecto principal y las causas se representan conforme a la estructura sintáctica de un problema en EP (véase la Tabla 4.21).

Para ilustrar los tres niveles del diagrama causa-efecto se presentan los siguientes

ejemplos en la Tabla 4.21: Si el problema en EP es "estudiante no gana el examen" suponiendo que este sea el problema principal, asímismo se representa como efecto principal en el diagrama causa-efecto. Si una causa principal para este efecto, es que el "examen tiene demora" y este a su vez tiene una causa secundaria, y es que el "examen no está disponible", también se describen de la misma manera en el diagrama.

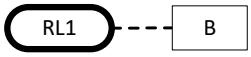
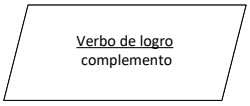
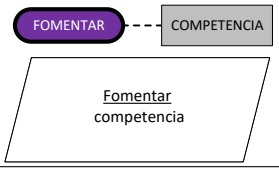
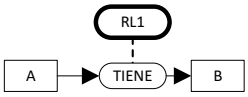
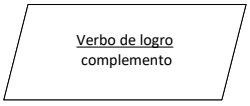
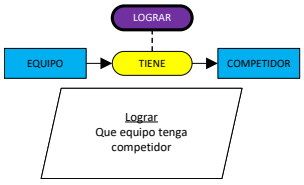
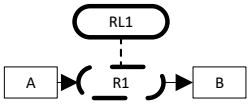
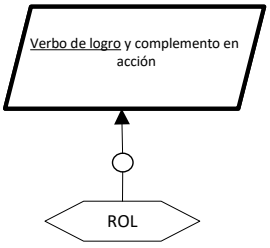
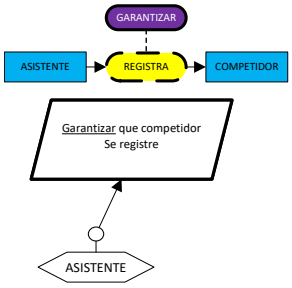
Tabla 4.21. Equivalencia de EP al diagrama de causa-efecto. Basado en Vargas (2016)

EP	Diagrama causa-efecto	Ejemplo
 Negación de relación de logro (aplica para las demás relaciones)		
 Concepto, valor o adverbio en connotación negativa		
 Valor negado	 Causa secundaria	

Equivalencia de EP al diagrama de objetivos KAOS: El *diagrama de objetivos* KAOS (especificación automática de adquisición de conocimientos; por sus siglas en inglés *Knowledge Acquisition Automated Specification*) se usa en ingeniería de requisitos para capturar los objetivos de un sistema de software. En la Tabla 4.22 se representan las reglas de equivalencia entre los EP y este diagrama. Por ejemplo, la relación de logro "fomentar" conectada al concepto "competencia" desde los EP se expresa de la misma manera en el paralelogramo del diagrama de objetivos, subrayando el objetivo "fomentar competencia" (véase la Tabla 4.22). También, si la relación de logro se vincula con una relación

estructural se extiende la frase en el mismo sentido de la lectura del EP. Por ejemplo, en la triada “equipo tiene competidor” la relación estructural “tiene” vincula la relación de logro “lograr”, es así como en el diagrama de objetivos se representa como “lograr que equipo tenga competidor” (véase la Tabla 4.22). Cuando la relación de logro se integra con una relación dinámica en los EP, también se puede representar en el diagrama de objetivos como un requisito/expectativa. Por ejemplo, la tríada “asistente registra competidor” tiene la relación dinámica “registra” y la relación de logro “garantizar”. Consecuentemente, el requisito/expectativa se representa como “garantizar que competidor se registre”. Es importante mencionar que todas las formas de representación de objetivos en EP (véase la Tabla 4.7) son equivalentes a los objetivos del diagrama de objetivos KAOS.

Tabla 4.22. Equivalencia de EP al diagrama de objetivos KAOS. Basado en Zapata et al. (2011d)

EP	Diagrama de objetivos KAOS	Ejemplo
		
		
		

Equivalencia de EP al diagrama de procesos: Los diagramas de procesos son representaciones del flujo de procesos, actores, eventos y objetos presentes

en los procesos. En la Tabla 4.23 se exponen las reglas de equivalencia entre los EP y el diagrama de procesos (Zapata, 2012). Por ejemplo, a partir de la triada "asistente recibe pedido" se puede representar la responsabilidad del actor, es decir, el "asistente", y en su responsabilidad se puede representar el proceso "recibe pedido". Además, si el objeto "pedido" tiene atributos como "referencia", "fecha" y "estado", estos se pueden representar también en el diagrama de procesos al lado del "pedido" como se observa en la Tabla 4.23.

Tabla 4.23. Equivalencia de EP al diagrama de procesos. Basado en Zapata (2012)

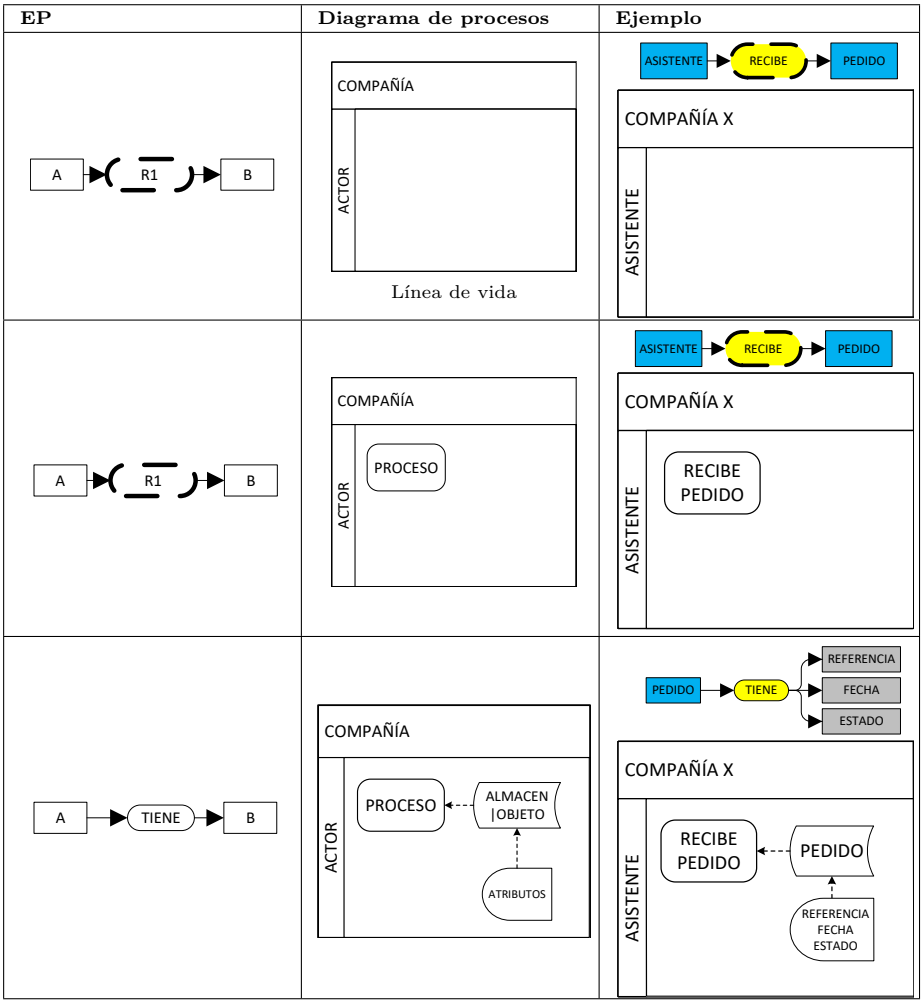
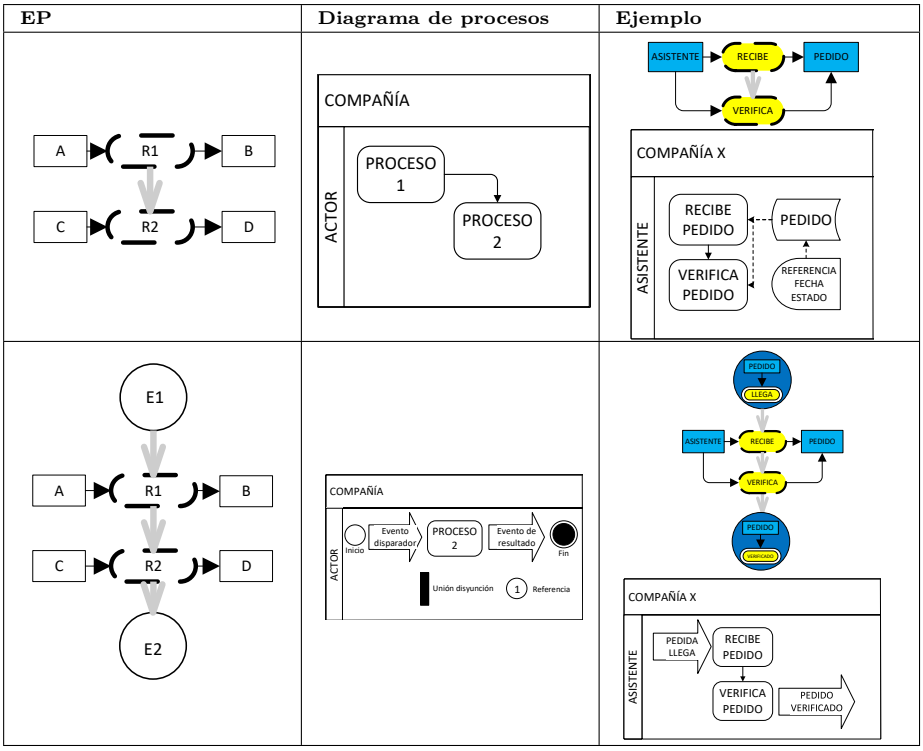


Tabla 4.23. (Continuación) Equivalencia de EP al diagrama de procesos. Basado en Zapata (2012)



Cuando se presenta la secuencia en las relaciones dinámicas, estas son equivalentes en el diagrama de procesos. Por ejemplo, las triadas "asistente recibe pedido" y luego, "asistente verifica pedido" se presentan como dos procesos "recibe pedido" y "verifica pedido" en secuencia en la línea de vida del actor "asistente" como se puede ver en la Tabla 4.23. Los eventos disparadores y de resultado también están presentes en la secuencia de flujo de procesos. Por ejemplo, el evento disparador "pedido llega" y el evento de resultado "pedido verificado"desde los EP, se representa de la misma manera dentro de una flecha según la notación del diagrama de procesos (véase la Tabla 4.23).

Equivalencia de EP al grafo de interacción de eventos: el *grafo de interacción de eventos* (EIG, por sus siglas en inglés, *event interaction graph*) a partir de la notación de EP, permite representar el flujo de interacción entre eventos y estados del sistema (Noreña-Cardona, 2014). Las equivalencias desde

el EP se basan en la interacción entre relaciones dinámicas (en las líneas de conjunción), eventos disparadores (círculos rojos) y eventos de resultado (círculos azules) presentes en el EP (véase la Tabla 4.24). Es importante señalar que un evento de resultado marca el cierre de una secuencia o flujo tanto en un EP como en un diagrama de procesos. No obstante, en el caso del EIG, sí es posible representar cómo un evento de resultado puede interactuar con un nuevo evento disparador.

Tabla 4.24. Equivalencia de EP al grafo de interacción de eventos. Basado en Noreña-Cardona (2014)

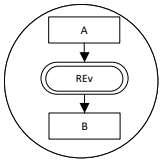
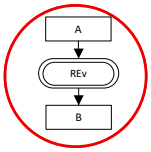
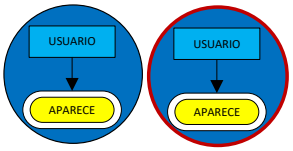
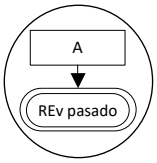
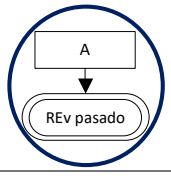
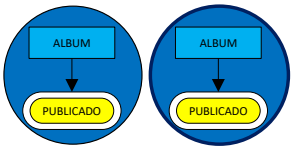
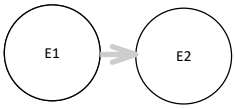
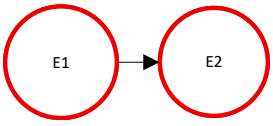
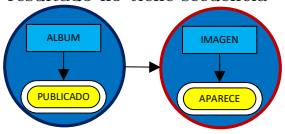
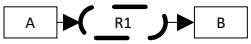
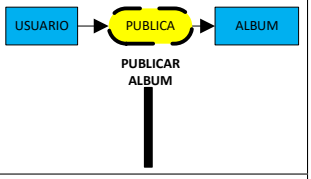
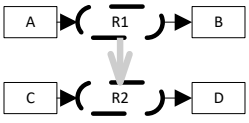
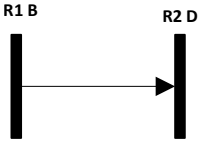
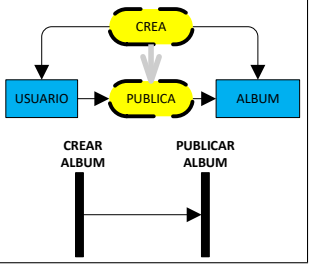
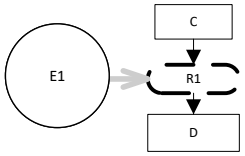
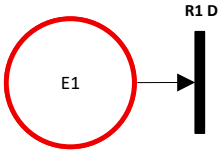
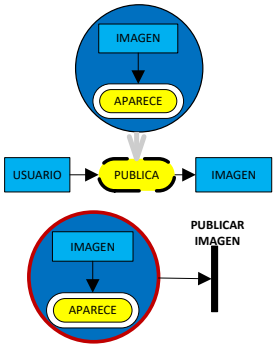
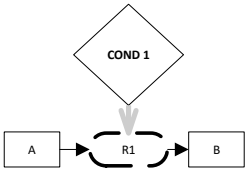
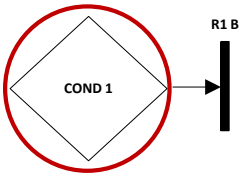
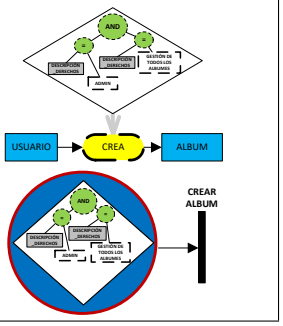
EP	EIG	Ejemplo
		
		
 E1 y E2 son disparadores	 E1 y E2 son disparadores, condicionales y de resultado	En EP un evento de resultado no tiene secuencia 
		
		

Tabla 4.24. (Continuación) Equivalencia de EP al grafo de interacción de eventos. Basado en Noreña-Cardona (2014)

EP	EIG	Ejemplo
		
		

En la Figura 4.1 se presenta la interacción completa de un EIG en el dominio de una red social hipótetica (Noreña-Cardona, 2014; Zapata, 2012).

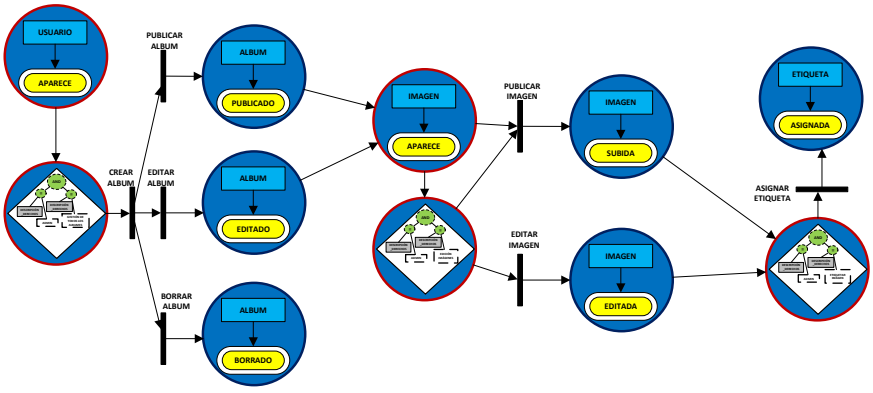


Figura 4.1. Grafo de interacción de eventos. Basado en Noreña-Cardona (2014)

Equivalencia de EP al diagrama entidad-relación (E/R): El *diagrama E/R* permite una representación estructurada de los datos, a partir de *entidades* que corresponden a clases, conceptos principales u objetos en el EP. Así, una clase en el EP se traduce en una entidad dentro del diagrama E/R, de forma similar al diagrama de clases, permitiendo incorporar atributos o campos propios de una base de datos. En la Tabla 4.25 se presentan las reglas de equivalencia entre los EP y el diagrama E/R y una posible traducción a bases de datos relacionales como SQL (Chaverra, 2011; Zapata et al., 2011c). Por ejemplo, la triada en EP "estudiante tiene nombre" se representa en el diagrama E/R mediante la entidad "estudiante" y su atributo "nombre", ubicado dentro del rectángulo correspondiente, junto con su tipo de dato.

Los *conceptos únicos*, que se definen en EP mediante la relación estructural "tiene_único" y una *conexión obligatoria* (doble), también se reflejan explícitamente en el E/R. Por ejemplo, en la triada "estudiante tiene_único código", el atributo "código" aparece dentro de la entidad "estudiante" como un *identificador único* (véase Tabla 4.25).

Tabla 4.25. Equivalencia de EP al diagrama entidad-relación. Basado en Chaverra (2011)

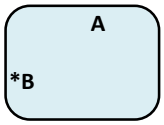
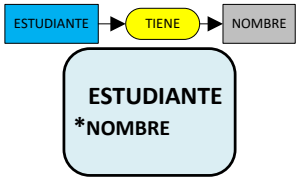
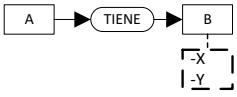
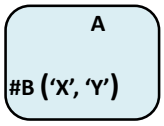
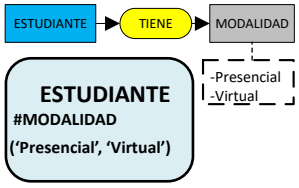
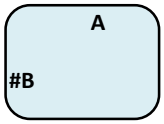
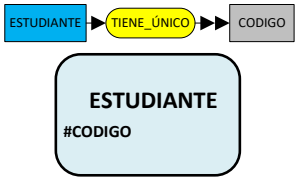
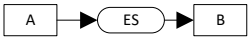
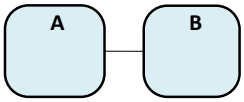
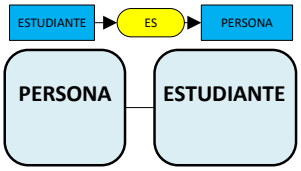
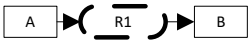
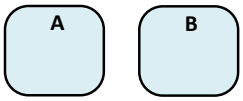
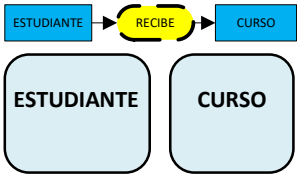
EP	Entidad-relación	Ejemplo en SQL
	 CREATE TABLE **A** (**B** varchar (30) default NULL,);	 CREATE TABLE **ESTUDIANTE** (**NOMBRE** varchar (30) default NULL);
	 CREATE TABLE **A** (**B** ENUM ('X', 'Y')); o CREATE TYPE **B** AS ENUM ('X', 'Y'); CREATE TABLE **A** (B B);	 CREATE TABLE **ESTUDIANTE** (**MODALIDAD** ENUM ('Presencial', 'Virtual')); o CREATE TYPE **MODALIDAD** AS ENUM ('Presencial', 'Virtual'); CREATE TABLE **ESTUDIANTE** (MODALIDAD MODALIDAD);

Tabla 4.25. (Continuación) Equivalencia de EP al diagrama entidad-relación. Basado en Chaverra (2011)

EP	Diagrama de procesos	Ejemplo
	 CREATE TABLE **A** (**B** varchar (30) default NULL, UNIQUE (B));	 CREATE TABLE **ESTUDIANTE** (**CODIGO** varchar (30) default NULL, UNIQUE (CODIGO));
	 CREATE TABLE **B** () CREATE TABLE **A** () INHERITS (**B**);	 CREATE TABLE **PERSONA** (); CREATE TABLE **ESTUDIANTE** INHERITS **PERSONA**();
	 CREATE TABLE **B** (); CREATE TABLE **B**();	 CREATE TABLE **ESTUDIANTE** (); CREATE TABLE **CURSO** ();

Asimismo, si un atributo posee valores, estos se pueden indicar entre paréntesis dentro del rectángulo de la entidad. Por ejemplo, en la relación "estudiante tiene modalidad" donde "modalidad" admite los valores "presencial" y "virtual", estos se deben registrar explícitamente entre paréntesis junto al atributo "modalidad" en el diagrama E/R (véase la Tabla 4.25).

Cuando en un EP se establece una relación de generalización entre dos clases utilizando la relación estructural "es", ambas se representan como entidades independientes en el diagrama E/R. Por ejemplo, en la triada "estudiante es persona", tanto "estudiante" como "persona" se visualizan como entidades. Del mismo modo, en una relación dinámica como "estudiante recibe curso", las clases también se representan como entidades en el diagrama E/R (véase la Tabla 4.25).

4.2.2. Equivalencias entre EP y Artefactos ágiles

Los esquemas preconceptuales son modelos de representación del conocimiento del dominio textual donde subyacen los requisitos del sistema. De esta manera, también tienen equivalencias con artefactos ágiles como el mapeo de historias de usuario, historias de usuario y diseños visuales.

Equivalencia entre el *user story mapping* y EP: El mapeo de historias de usuario o *user story mapping* es un tablero donde se representan los actores y las historias de usuario (HU) correspondientes a los requisitos funcionales y no funcionales del sistema mediante notas adhesivas. A partir de este tablero, es posible extraer directamente actores e HU de requisitos funcionales para representarlos en EP. En la Figura 4.2 se muestra el caso de una farmacéutica que necesita un sistema de alertas de fechas de vencimiento y un módulo de venta a droguerías. En la Tabla 4.26 se resumen las reglas de equivalencia entre los elementos de este tablero y los EP.

En la parte superior del tablero se encuentran los *usuarios* por lo que se pueden representar en los EP como *actores*. Por ejemplo, "administrador de farmacéutica" en la Tabla 4.26. Esto también aplica para las *tareas* o HU en cada *sprint*. Por ejemplo, si el usuario es "administrador farmacéutica" y el requisito funcional o tarea es "ingresar medicamento", entonces se puede representar como una triada de relación dinámica en los EP así: "administrador farmacéutica ingresa medicamento" (véase la Tabla 4.26).

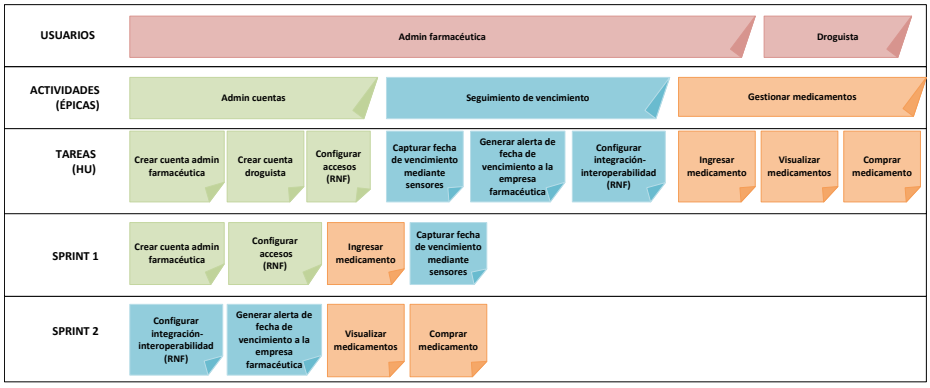


Figura 4.2. Mapeo de historias de usuario

Tabla 4.26. Equivalencia entre el *User story mapping* y EP

User story mapping	EP	Ejemplo
Usuario	A	Administrador farmacéutica / ADMINISTRADOR FARMACÉUTICA
Tarea (HU)	A → R1 → B	Administrador farmacéutica / Ingresar medicamentos; ADMINISTRADOR FARMACÉUTICA → INGRESA → MEDICAMENTO
Actividad (épica)	A, C → R1 (R2, R3) → B	Droguista / Gestionar medicamentos; GESTIONA (DROGUISTA; VISUALIZA, COMPRA; MEDICAMENTO)
Tarea (automática, HU)	E1	Capturar fecha de vencimiento (mediante sensores); MEDICAMENTO, FECHA DE VENCIMIENTO → CAPTURADA

En caso de que la tarea sea automática, se procede a representarla como un evento en EP. Por ejemplo, si se encuentra en la nota "capturar fecha de vencimiento (mediante sensores)", entonces se representa como un evento en EP así: "fecha de vencimiento del medicamento capturada" (véase la Tabla 4.26).

Si la HU se encuentra en la línea de *actividades*, esto significa que hay una jerarquía de HU *épicas* en el tablero. Por lo que, en EP se puede representar con el símbolo de marco. Por ejemplo, si la épica es "gestionar medicamento" y esta épica agrupa a las HU "visualizar medicamento" y "comprar medicamento" en el *story mapping*, entonces, se representan dos triadas de relaciones dinámicas "droguista visualiza medicamento" y "droguista compra medicamento" y se agrupan en un marco que se llama "gestiona" indicando que esta acción incluye las dos relaciones anteriores (véase la Tabla 4.26). Los colores se pueden incluir de acuerdo a la necesidad del analista.

Equivalencia entre historias de usuario y EP: las HUs son artefactos ágiles que permiten documentar de forma descriptiva los requisitos funcionales

y no funcionales de un sistema. Estas historias incluyen atributos como el actor, nombre, descripción, criterios de aceptación, código, módulo, entre otros elementos del formato. Las reglas de equivalencia entre HUs y EP se expresa en la Tabla 4.27 (Villamizar, 2019).

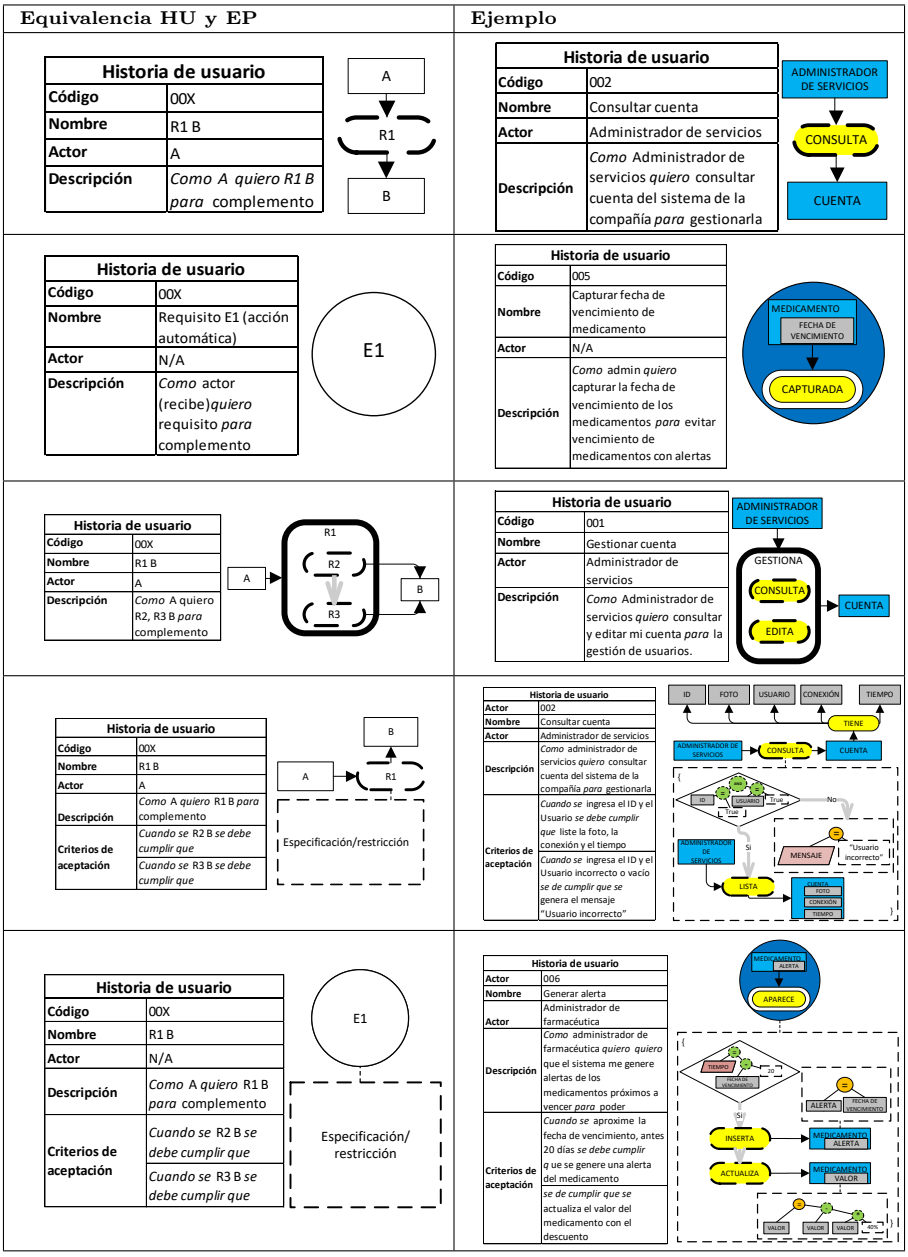
Al igual que en el *user story mapping*, los EP permiten representar los actores y requisitos funcionales de las HU. Teniendo en cuenta el actor y nombre de la HU. Por ejemplo, si el actor es "administrador de servicios", este se modela como un concepto en el EP (véase la Tabla 4.27). Cuando la HU especifica un actor y una acción, se representa como una triada de relación dinámica. Así, la HU 002 "consultar cuenta" y su actor "administrador de servicios" se traducen en EP como "administrador de servicios consulta cuenta".

En caso de que el requisito funcional sea automático entonces se representa como un evento. Por ejemplo, la HU 005 en la Tabla 4.27 es "capturar fecha de vencimiento de medicamento", entonces se representa como un evento así: "fecha de vencimiento del medicamento capturada". Para estos requisitos, también se pueden apoyar en la descripción de la HU para detalles en los actores y el nombre de la HU. Por ejemplo, en la descripción de esta HU, se encuentra el actor que recibirá la notificación del evento *"como admin quiero capturar la fecha de vencimiento de los medicamentos para evitar vencimiento de medicamentos con alertas"*. Además, se encuentra el requisito después de la palabra "quiero".

También, se pueden agrupar funcionalidades en un *marco* a partir de HU épicas (categoría que agrupa otras HU). Por ejemplo, la HU 002 "gestionar cuenta" agrupa las HU "consultar cuenta" y "editar cuenta", de la misma manera, su equivalente en la representación de EP presenta ambas funcionalidades dentro de "gestionar cuenta" (véase la Tabla 4.27). Estas agrupaciones se suelen realizar en funcionalidades atómicas (insertar, seleccionar, actualizar, listar y borrar).

En un nivel de abstracción 2, los *criterios de aceptación* de las HU también se pueden representar en un EP ejecutable. Esto implica que se pueden representar mediante una especificación o restricción de una relación dinámica. Por ejemplo, los criterios de aceptación de la HU 002 "consultar cuenta" (véase la Tabla 4.27) se representan en la especificación de la RD en la triada "administrador de servicios consulta cuenta" donde la condición y la acción interna "administrador de servicios lista foto, conexión y tiempo" se describen en los criterios de aceptación.

Tabla 4.27. Equivalencia entre historias de usuario y EP. Basado en Villamizar (2019)



La funcionalidad anterior la realiza un actor del sistema, en caso contrario,

cuando la funcionalidad es automática y no presenta actor, igualmente se pueden representar sus criterios de aceptación mediante la especificación o restricción de un evento. Por ejemplo, los criterios de aceptación de la HU 006 "generar alerta" (véase la Tabla 4.27) se representan en la especificación del evento "alerta de medicamento aparece" donde la condición y las acciones internas y automáticas "generar alerta de medicamento" y "actualizar valor de medicamento" se describen en los criterios de aceptación.

Equivalencia entre EP y prototipos visuales: Los *sketches*, *wireframes* y *mock-ups* son prototipos visuales de la interfaz gráfica de usuario (GUI, por sus siglas en inglés) de una aplicación. Tanto la apariencia como la navegabilidad y la accesibilidad (experiencia usuario, UX por sus siglas en inglés) se pueden validar con pruebas de aceptación desde estos prototipos para el desarrollo front (lado del cliente) usando EP. En la Tabla 4.28 se puede observar la equivalencia entre los EP y estos prototipos con elementos de la GUI. Esta equivalencia facilita la consistencia entre las expectativas del cliente y la aplicación. Es importante mencionar que en la Tabla 4.28 se presentan varias opciones de representación desde los EP: (a) o (b) para un mismo elemento de la interfaz.

El *campo* es un elemento de la GUI, una caja de texto que permite incluir información. Este elemento se ejemplifica con los campos "correo" y "contraseña" del concepto-clase "usuario" utilizando la regla de equivalencia de la relación estructural en EP (ver la Tabla 4.28).

La *lista* es un elemento de la GUI que permite desplegar un conjunto de opciones asociadas a un concepto. Por ejemplo, el concepto "tipo de dispositivo" puede incluir las opciones: "computador", "celular" y "*tablet*" (véase la Tabla 4.28). Para representar una lista desde el EP se puede utilizar un concepto con sus valores, dicho concepto tiene la opción de especificar su tipo mediante un estereotipo «lista» como se observa en la Tabla 4.28.

Los siguientes son elementos de la GUI que se pueden utilizar para crear listas, en los prototipos y el desarrollo front a partir de esta representación de EP (véase la Figura 4.3):

- *Dropdown:* Permite al usuario seleccionar una única opción desde una lista desplegable (menú desplegable). Por ejemplo, en la selección del tipo de dispositivo.

- *Checkbox:* Permite seleccionar una o varias opciones simultáneamente de un conjunto. Este tipo de lista es útil cuando se requiere marcar múltiples valores.
- *Radio button:* Permite seleccionar sólo una opción entre varias posibles, excluyendo las demás automáticamente.
- *Switch:* Permite activar o desactivar una opción con un solo clic, facilitando decisiones binarias como encender o apagar.

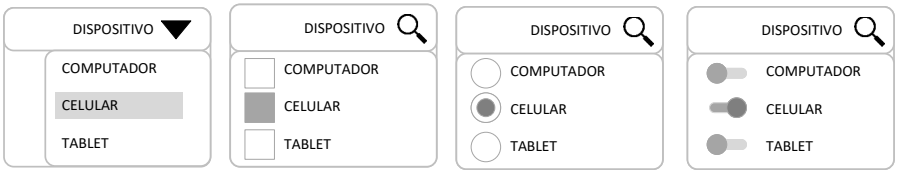


Figura 4.3. Tipos de listas en la GUI. Basado en Villamizar (2019)

Tabla 4.28. Equivalencia entre EP y prototipos visuales. Basado en Villamizar (2019)

EP	Equivalencia y elementos GUI
(a) A TIENE B (b) A TIENE <<CAMPO>> B Campo	USUARIO, TIENE, LOGIN, CORREO, <<CAMPO>> CONTRASEÑA, INGRESA LOGIN, CORREO, CONTRASEÑA, INGRESAR
(a) B -B1 -B2 -B3 (b) <<LISTA>> B -B1 -B2 -B3 Lista (*checkbox, dropdown, radio, switch,* entre otros)	TIENE, DISPOSITIVO (a) TIPO -COMPUTADOR -CELULAR -TABLET (b) <<LISTA>> TIPO DE DISPOSITIVO -COMPUTADOR -CELULAR -TABLET DISPOSITIVO, COMPUTADOR, CELULAR, TABLET

Tabla 4.28. (Continuación) Equivalencia entre EP y prototipos visuales. Basado en Villamizar (2019)

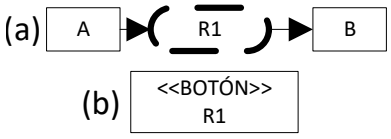
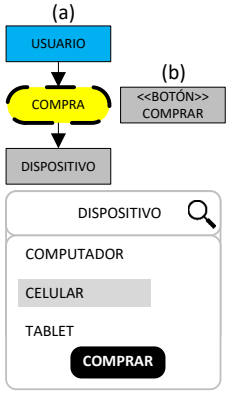
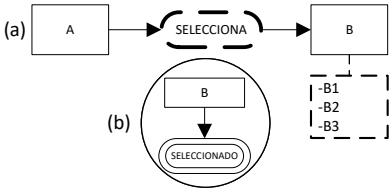
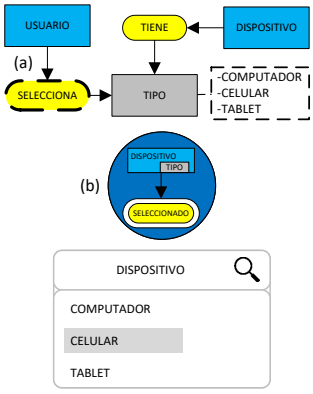
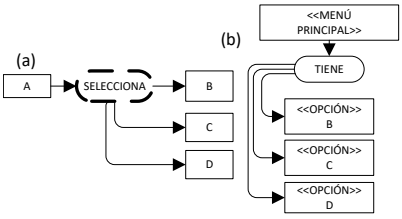
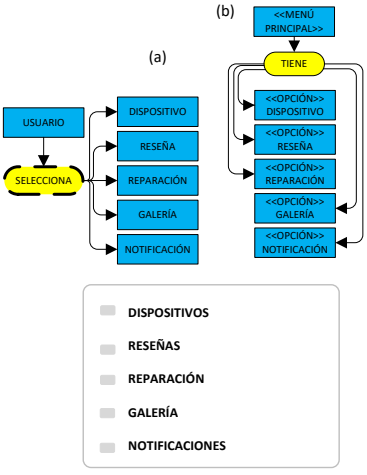
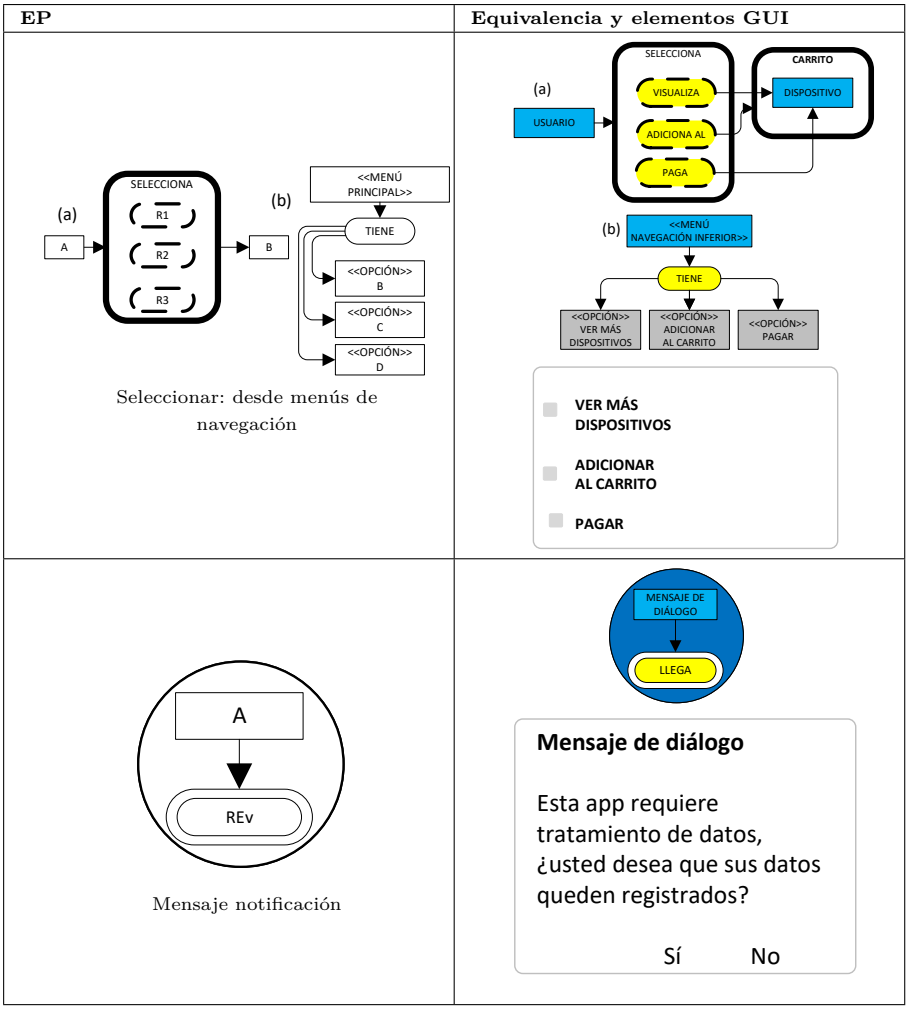
EP	Equivalencia y elementos GUI
 Botón	
 Seleccionar: mediante interacción directa	
 Seleccionar: desde menús principales y submenús	

Tabla 4.28. (Continuación) Equivalencia entre EP y prototipos visuales. Basado en Villamizar (2019)



En cuanto al *botón*, este es un elemento GUI para ejecutar acciones. En el contexto de los EP, el botón se representa mediante una relación dinámica. Por ejemplo, en la triada "usuario compra dispositivo", el verbo "comprar" corresponde a la acción que el botón activa (véase la Tabla 4.28).

Adicionalmente, esta acción se puede representar de forma explícita utilizando la etiqueta «botón» seguida del nombre de la acción, es decir, «botón» comprar, tal como se ilustra en la misma tabla.

Por otro lado, una de las acciones más comunes en la GUI es la acción de *seleccionar*, la cual se puede presentar en diferentes formas:

(i) *Mediante interacción directa:* el usuario puede realizar una selección mediante un evento *clic* o *tags* (toques en pantalla). Esta acción se puede representar en EP como una relación dinámica, por ejemplo, "usuario selecciona tipo de dispositivo". Alternativamente, si la selección es automática, se representa como un evento, por ejemplo, "tipo de dispositivo seleccionado" (véase la Tabla 4.28).

(ii) *Desde menús principales y submenús:* estos menús permiten acceder a los recursos principales de la aplicación y se representan en EP mediante conceptos clase. Por ejemplo, un usuario puede seleccionar en el menú principal opciones como: "dispositivo, reseña, reparación, galería, notificación". En este caso, "dispositivo" puede actuar como concepto principal que agrupa a los demás (véase la equivalencia entre EP y *sketch* en la Tabla 4.28). Sin embargo, cada uno de estos conceptos también puede ser principal si tiene sus propios conceptos hoja, como "reparación", que podría tener "fecha de reparación, técnico y observación" como atributos asociados.

(iii) *Desde menús de navegación:* estos permiten el acceso a funcionalidades adicionales y se representan mediante relaciones dinámicas en EP. Pueden aparecer como menús desplegables dentro de una ventana. Por ejemplo, en la ventana del concepto "dispositivo", el usuario puede seleccionar "compartir, cargar, copiar imagen del dispositivo" o "imprimir cotización" (véase la Tabla 4.28). Otra variante es la navegación inferior, donde se presentan opciones como "visualizar y pagar dispositivo" o "agregar al carrito".

Todos los tipos de menús se pueden representar con *estereotipos*. Por ejemplo, para las opciones "ver más dispositivos", "agregar al carrito" o "pagar", se puede utilizar el estereotipo «opción» sobre cada concepto o dentro de un marco que agrupe dichas acciones y el estereotipo «menú» con su clasificación por ejemplo «menú principal» (véase la Tabla 4.28).

Finalmente, los *mensajes de notificación* que se generan en la aplicación se representan en EP como eventos o diálogos modales. Un ejemplo sería "mensaje de diálogo llega", el cual aparece como evento automático para alertar al usuario (véase la Tabla 4.28).

4.2.3. Equivalencias de EP a código fuente

Las equivalencias entre los esquemas preconceptuales y el código fuente se fundamentan en los principios del paradigma orientado a objetos; no obstante, esta base no limita su aplicación a lenguajes de programación con otros enfoques o modelos de desarrollo. En términos generales, un elemento dentro de un sistema se representa mediante una estructura que contiene atributos y un conjunto de operaciones que se materializan en sus métodos. Debido a esta forma de organización, es posible establecer correspondencias entre los componentes que se definen en los EP y las construcciones presentes en diversos lenguajes de programación. Así, la representación conceptual que proporciona los esquemas sirve como punto de partida para generar equivalencias en cualquier lenguaje que permita describir datos, comportamientos o relaciones entre entidades, independientemente del paradigma que adopte.

Equivalencia de EP a lenguajes de *back-end*: El *back-end* (lado del servidor) permite gestionar la lógica de negocio y las interacciones con los datos. Las equivalencias desde el EP incluyen conceptos del dominio, los cuales se pueden traducir como clases o nuevos archivos de la aplicación, sus conceptos hoja como atributos de la clase y las relaciones dinámicas y eventuales como métodos en las clases y las relaciones de logro como restricciones. En la Tabla 4.29 se presentan las reglas de equivalencia desde los EP a código en Python, Java, C#.

Por ejemplo, el concepto “paciente” del dominio se traduce como una *clase* al código fuente en el lenguaje de programación, al igual que su atributo “id”. De esta misma clase, también se presenta una *herencia* desde la clase “usuario” y el *método* “ingresar paciente” que se traduce de la relación dinámica “recepcionista ingresa paciente”. El evento “tiempo pasa” al ser un proceso automático se programa como un *método*, las condiciones iniciales se traducen como *variables globales* mientras que las *restricciones* se relacionan con una especificación de un concepto y una relación de logro (véase la Tabla 4.29).

Consecuentemente, las especificaciones/restricciones de relaciones dinámicas y eventuales constituyen la *lógica interna de la funcionalidad*, por lo que los elementos que se encuentran dentro de la especificación se traducen dentro del *método* (véase la Tabla 4.29).

Tabla 4.29. Equivalencias EP a lenguajes *back-end*. Basado en Zapata y Cardona (2008), Manjarrés (2010), Chaverra (2011), Calle (2016), Velásquez (2019) y Noreña (2020)

Python	Java	C#
		
`class A:` `class Paciente:`	`public class A {}` `public class Paciente {}`	`public class A {}` `public class Paciente {}`
		

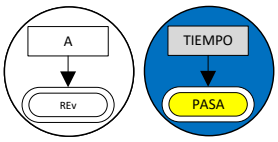
`class A(C):`
 	
`class Usuario:`
`class Paciente(Usuario):`
	`public class A extends C{}`
	`public class A implements C{`
	`}`
	
	`public class Paciente`
	`extends Usuario{}`
	`public class Paciente`
	`implements Usuario{}`
		`public class C{}`
		`public class A : C{}`
		
		`public class Usuario{}`
		`public class Paciente :`
		`Usuario{}`

`class A:`
`def __init__(self, B):`
`self.B=B`
 	
`class Paciente:`
`def __init__(self,id):`
`self.id = id`
	`public class A {`
	`visibilidad tipo B;`
	`}`
	
	`public class Paciente {`
	`private int id;`
	`}`
	`}`
		`public class A {`
		`visibilidad tipo B;}`
		
		`public class Paciente {`
		`public int Id {}`
		`}`

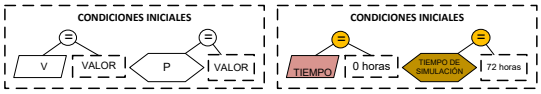
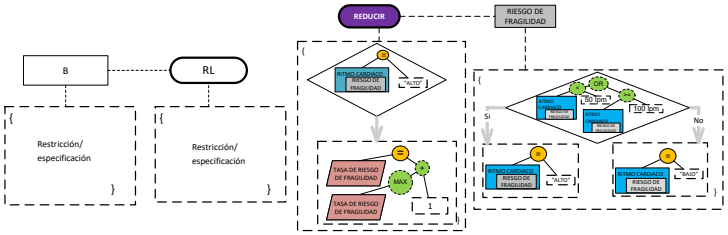
		

`def r1A():`
`def r1_A():`
 	
`def ingresarPaciente():`
`def ingresar_Paciente():`
	`public static void r1A() {}`
	`public static void r1_A(){}`
	
	`public static void`
	`ingresarPaciente() {}`
	`public static void ingresar_`
	`Paciente() {}`
		`public void r1A() {}`
		`public void r1_A() {}`
		
		`public void ingresarPaciente`
		`() {}`
		`public void ingresar_`
		`Paciente() {}`

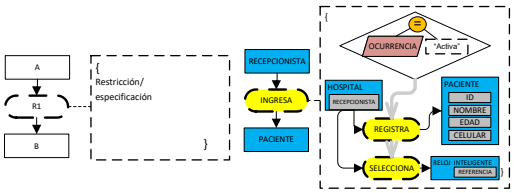
`def AE1()`
`def A_E1()`
 	
`def TIEMPO_PASA()`
	`public static void AREv(){}`
	`public static void A_REv(){}`
	
	`public static void TIEMPO_`
	`PASA() {}`
		`public void AREv() {}`
		`public void A_REv() {}`
		
		`public void TIEMPO_PASA () {`
		`}`

Tabla 4.29. (Continuación) Equivalencias EP a lenguajes *back-end*. Basado en Zapata y Cardona (2008), Manjarrés (2010), Chaverra (2011), Calle (2016), Velásquez (2019) y Noreña (2020)

Python	Java	C#
		
global V=VALOR **global** P=VALOR **global** TIEMPO= 0 **global** TIEMPO_SIMULACION=72	visibilidad tipo V=VALOR; visibilidad **final** tipo P= VALOR; **public static int** TIEMPO=0; **public static final int** TIEMPO_SIMULACION=72;	visibilidad tipo V=VALOR; visibilidad **const** tipo P= VALOR; **public static int** TIEMPO=0; **public static const int** TIEMPO_SIMULACION=72;
		

class B:
Restricción
 	
class RiesgoFragilidad:
 	
if valor<=50 **or** valor>=100:
riesgo="Alto"
else:
riesgo="Bajo"
 	
ritmo_cardiaco["riesgo_
fragilidad"]=riesgo
 	
if TIEMPO>1:
PROMEDIO_RITMOCARDIACO
=((PROMEDIO_
RITMOCARDIACO*(
TIEMPO-1))+valor)/
TIEMPO
 	
if riesgo=="Alto":
TASA_RIESGOFRAGILIDAD=
max(TASA_
RIESGOFRAGILIDAD+1,
TASA_
RIESGOFRAGILIDAD)
	public class B {
	Restricción;
	}
	
	Public **class**
	RiesgoFragilidad{
	
	String riesgo;
	
	if (valor<=50——valor>=100){
	riesgo="Alto";}
	else {
	riesgo="Bajo";}
	
	ritmoCardiaco.
	setRiesgoFragilidad(
	riesgo);
	
	if (TIEMPO>1){
	PROMEDIO_RITMOCARDIACO
	=((PROMEDIO_
	RITMOCARDIACO*(
	TIEMPO-1))+valor)/
	TIEMPO;}
	if (riesgo.equals("Alto")) {
	TASA_RIESGOFRAGILIDAD=
	Math.max(TASA_
	RIESGOFRAGILIDAD+1,
	TASA_
	RIESGOFRAGILIDAD);}
		public class B {
		Restricción;}
		
		public class
		RiesgoFragilidad{
		string riesgo;
		if (valor<=50——valor>=100){
		riesgo="Alto";}
		else{
		riesgo="Bajo";}
		
		ritmoCardiaco.
		RiesgoFragilidad=riesgo
		;
		if (TIEMPO>1){
		PROMEDIO_RITMOCARDIACO
		=((PROMEDIO_
		RITMOCARDIACO*(
		TIEMPO-1))+valor)/
		TIEMPO;}
		
		if (riesgo=="Alto"){
		TASA_RIESGOFRAGILIDAD=
		Math.Max(TASA_
		RIESGOFRAGILIDAD+1,
		TASA_
		RIESGOFRAGILIDAD);}

Tabla 4.29. (Continuación) Equivalencias EP a lenguajes *back-end*. Basado en Zapata y Cardona (2008), Manjarrés (2010), Chaverra (2011)...



Python

```
def r1A():
  Restricción/
  especificación

def ingresarPaciente():
 for i in range (0,2):
    id = int(input("Ingrese
         el ID del paciente:
          "))
    nombre = input("Ingrese
         el nombre del
         paciente: ")
    edad = int(input("
         Ingrese la edad del
          paciente: "))
    celular = int(input("
         Ingrese el celular
         del paciente: "))
    referencia = input("
         Ingrese la
         referencia del
         paciente: ")
    print ("")
    paciente = PACIENTE(id,
         nombre, edad,
         celula, referencia)
    LISTA_PACIENTES.append(
         paciente)
```

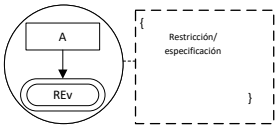
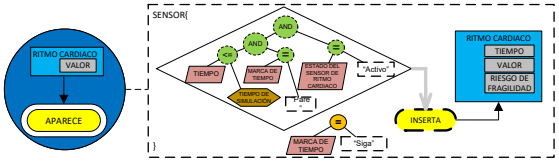
Java

```
public static void r1A()
  {Restricción/
   especificación}

public static void
     ingresarPaciente() {
 Scanner scanner = new
      Scanner(System.in);
 for (int i=0;i<2;i++) {
    System.out.println("
         Ingrese datos del
         paciente " + (i +
         1) + ":");
    System.out.print("ID: ")
         ;
    int id = Integer.
         parseInt(scanner.
         nextLine());
    System.out.print("Nombre
         : ");
    String nombre = scanner.
         nextLine();
    System.out.print("Edad:
         ");
    int edad = Integer.
         parseInt(scanner.
         nextLine());
    System.out.print("
         Celular: ");
    String celular = scanner
         .nextLine();
    System.out.print("
         Referencia reloj: "
         );
    String referencia =
         scanner.nextLine();
    Paciente paciente = new
         Paciente(id, nombre
         , edad, celular,
         referencia);
    listaPacientes.add(
         paciente);
   }
  }
```

C#

```
public static void r1A()
  {Restricción/
   especificación}

public static void
     IngresarPaciente(){
 for (int i= 0; i<2;i++){
    Console.WriteLine($"
         Ingrese los datos
         del paciente {i + 1
         }:");
    Console.Write("ID: ");
    int id = int.Parse(
         Console.ReadLine()
         ?? "0");
    Console.Write("Nombre: "
         );
    string nombre = Console.
         ReadLine() ??
         string.Empty;
    Console.Write("Edad: ");
    int edad = int.Parse(
         Console.ReadLine()
         ?? "0");
    Console.Write("Celular:
         ");
    string celular = Console
         .ReadLine() ??
         string.Empty;
    Console.WriteLine();
    Console.Write("
         Referencia reloj: "
         );
    string referencia =
         Console.ReadLine()
         ?? string.Empty;

    var paciente = new
         Paciente(id, nombre
         , edad, celular,
         referencia);
    listaPacientes.Add(
         paciente);
   }
}
```

Tabla 4.29. (Continuación) Equivalencias EP a lenguajes *back-end*. Basado en Zapata y Cardona (2008), Manjarrés (2010), Chaverra (2011)...

Python	Java	C#
		

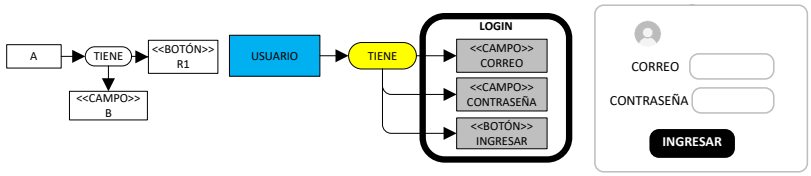
```
def A_REv():
 Restricción/
 especificación

def RITMOCARDIACO_APARECE():
 global PROMEDIO_
      RITMOCARDIACO, TASA_
      RIESGOFRAGILIDAD,
      MARCA_TIEMPO

 if TIEMPO<=TIEMPO_
      SIMULACION and MARCA_
      TIEMPO=="Pare" and
      ESTADOSENSOR_
      RITMOCARDIACO=="Activo
      ":
    valor=random.randint(50,
          120)
    riesgo="Alto" if valor>
         100 else "Bajo"

 evento={
 'Tiempo': TIEMPO,'Ritmo
      Cardiaco': valor,'
      Riesgo': riesgo}
 registro_eventos.append(
      evento)
```

```
public static void A_REv()
  {Restricción/
   especificación}

public static void
     RITMOCARDIACO_APARECE()
      {
 if (TIEMPO<=TIEMPO_
      SIMULACION && MARCA_
      TIEMPO.equals("Pare")
      && ESTADO_SENSOR.
      equals("Activo")) {

  int valor = new Random().
       nextInt(71)+50;
  String riesgo = (valor >
       100) ? "Alto" : "
       Bajo";

  Map<String,Object> evento
        = new HashMap<>();
  evento.put("Tiempo",
       TIEMPO);
  evento.put("Ritmo
       Cardiaco", valor);
  evento.put("Riesgo",
       riesgo);
  registroEventos.add(
       evento);}}
```

```
public static void A_REv()
 {Restricción/
  especificación}

public static void
     RITMOCARDIACO_APARECE()
     {
  if (Tiempo<=
      TiempoSimulacion &&
      MarcaTiempo == "Pare"
       && EstadoSensor == "
      Activo"){
   var rand = new Random();
   int valor = rand.Next
       (50, 121);
   string riesgo = valor >
       100 ? "Alto" : "
       Bajo";

   var evento = new
       Dictionary<string,
       object> {
   ["Tiempo"] = Tiempo,
   ["Ritmo Cardiaco"] =
       valor,
   ["Riesgo"] = riesgo};
   RegistroEventos.Add(
       evento);}
}
```

Equivalencia de EP a lenguajes de *front-end*: El *front-end* (lado del cliente) se encarga de gestionar la interacción directa del usuario y de proporcionar la representación visual de los elementos del sistema. En este contexto, los elementos de un esquema preconceptual se pueden expresar mediante *etiquetas, formularios, menús* y *campos de entrada* como se muestra en la Tabla 4.30 para HTML (hipertexto de etiquetas para páginas web), CSS (hojas de estilo) y JavaScript, los cuales permiten estructurar contenido, aplicar estilos y manejar comportamientos dinámicos en una interfaz. Los *conceptos clase* suelen corresponder a usuarios y elementos de la GUI como *formularios, paneles, menús* o *secciones* al igual que en el prototipado (véase la Tabla 4.28).

Tabla 4.30. Equivalencia de EP a lenguajes *front-end*: HTML, CSS y JavaScript. Basado en Zapata (2012) y Villamizar (2019)

<table>
<tr><th>HTML</th><th>CSS</th><th>JavaScript</th></tr>
<tr><td colspan="3"> 

Campo, Botón, formulario</td></tr>
<tr><td>

```
<div class="login-
    container">
  <h2>LOGIN</h2>
  <img src="user-icon.
      png" alt="User
      Icon" class="
      avatar">
  <form id="loginForm">
    <label for="correo"
        >CORREO</label
        >
    <input type="email"
        id="correo"
        name="correo"
        required>

    <label for="
        contrasena">
        CONTRASEÑA</
        label>
    <input type="
        password" id="
        contrasena"
        name="
        contrasena"
        required>

    <button type="
        submit" id="
        ingresar">
        INGRESAR</
        button>
  </form>
</div>
```

</td><td>

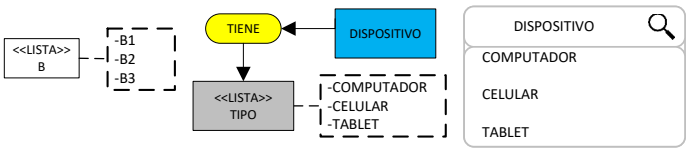
```
.login-container {
  width: 300px;
  padding: 20px;
  margin: auto;
  border-radius: 10px;
  font-family: sans-
      serif;
  background-color: #
      f9f9f9;
  text-align: center;}

.login-container h2 {
  margin-bottom: 10px;}

input {
  width: 90%;
  padding: 10px;
  margin: 10px 0;
  border-radius: 5px;}

button {
  background-color:
      #000;
  color: #fff;
  padding: 10px 20px;
  border: none;
  width: 100%;
  border-radius: 5px;
  cursor: pointer;}
```

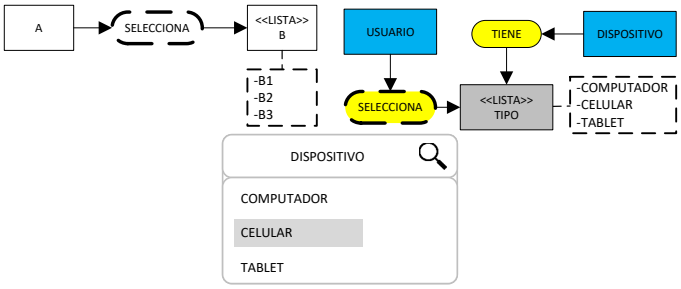
</td><td>

```
document.getElementById
    ("loginForm").
    addEventListener("
    submit", function(
    event) {
  event.preventDefault
      ();
  const correo =
      document.
      getElementById("
      correo").value;
  const contrasena =
      document.
      getElementById("
      contrasena").
      value;

  if (correo &&
      contrasena) {
    alert("Ingreso
        exitoso");
  } else {
    alert("Por favor
        completa todos
        los campos.")
        ; }});
```

</td></tr>
</table>

Tabla 4.30. (Continuación) Equivalencia de EP a lenguajes *front-end*: HTML, CSS y JavaScript. Basado en Zapata (2012) y Villamizar (2019)

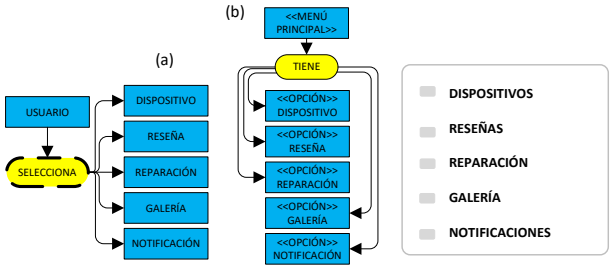
<table>
<tr><th>HTML</th><th>CSS</th><th>JavaScript</th></tr>
<tr><td colspan="3"> 

Lista</td></tr>
<tr><td><pre>
<select id="dispositivo
    ">
  <option>COMPUTADOR</
      option>
  <option>CELULAR</
      option>
  <option>TABLET</
      option>
</select>
</pre></td><td><pre>
#dispositivo {
  width: 200px;
  padding: 10px;
  font-size: 14px;
  border-radius: 8px;
  border: 1px solid #
      ccc;
}
</pre></td><td></td></tr>
<tr><td colspan="3"> 

Seleccionar</td></tr>
<tr><td><pre>
<select id="dispositivo
    ">
  <option>COMPUTADOR</
      option>
  <option>CELULAR</
      option>
  <option>TABLET</
      option>
</select>
</pre></td><td><pre>
#dispositivo {
  width: 200px;
  padding: 10px;
  font-size: 14px;
  border-radius: 8px;
  border: 1px solid #
      ccc;
}
</pre></td><td><pre>
document.getElementById
    ("dispositivo").
    addEventListener("
    change", function
    () {
  const seleccionado =
      this.value;
  console.log("
      Seleccionaste: "
       + seleccionado)
      ;
});
</pre></td></tr>
</table>

Tabla 4.30. (Continuación) Equivalencia de EP a lenguajes *front-end*: HTML, CSS y JavaScript. Basado en Zapata (2012) y Villamizar (2019)

HTML



Menús, submenús, seleccionar

```
<div class="menu">
  <h3>MENÚ PRINCIPAL</
      h3>
  <ul>
    <li onclick="
         seleccionar('
         DISPOSITIVO')"
         >DISPOSITIVO</
         li>
    <li onclick="
         seleccionar('
         RESEÑA')">RESE
         ÑA</li>
    <li onclick="
         seleccionar('
         REPARACIÓN')">
         REPARACIÓN</li
         >
    <li onclick="
         seleccionar('
         GALERÍA')">
         GALERÍA</li>
    <li onclick="
         seleccionar('
         NOTIFICACIÓN')
         ">NOTIFICACIÓN
         </li>
  </ul>
</div>

<p id="seleccion">
     Seleccione una
     opción del menú</p
     >
```

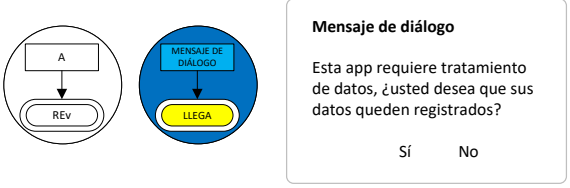
CSS

```
.menu {
  width: 250px;
  border: 1px solid #
       ccc;
  border-radius: 10px;
  padding: 15px;
  font-family: Arial,
       sans-serif;
  background-color: #
       fdfdfd;
}

.menu h3 {
  text-align: center;
  margin-bottom: 10px;
  font-size: 16px;
}

.menu ul {
  list-style: none;
  padding: 0;
}

.menu li {
  padding: 10px;
  margin-bottom: 5px;
  background-color: #
       e6f2ff;
  border-radius: 5px;
  cursor: pointer;
  text-align: center;
  font-weight: bold;
}
```

JavaScript

```
function seleccionar(
     opcion) {
  document.
       getElementById("
       seleccion").
       textContent = '
       Seleccionaste: $
       {opcion}';
}
```

Tabla 4.30. (Continuación) Equivalencia de EP a lenguajes *front-end*: HTML, CSS y JavaScript. Basado en Zapata (2012) y Villamizar (2019)

HTML	CSS	JavaScript
 Mensaje notificación		
`<button onclick="` `mostrarDialogo()">` `Mostrar diálogo</` `button>` `<!-- Diálogo modal -->` `<div id="dialogo" class` `="modal">` `<div class="modal-` `contenido">` `<h3><strong>Mensaje` `de diálogo</` `strong></h3>` `<p>Esta app` `requiere` `tratamiento de` `datos, ` `usted desea que sus` `datos queden` `registrados?</` `p>` `<div class="botones` `">` `<button onclick="` `respuesta('S` `í')">Sí</` `button>` `<button onclick="` `respuesta('` `No')">No</` `button>` `</div>` `</div>` `</div>` `<p id="resultado"></p>`	`.modal {` `display: none; /* Se` `oculta por` `defecto */` `position: fixed;` `z-index: 999;` `left: 0;` `top: 0;` `width: 100%;` `height: 100%;` `background-color:` `rgba(0,0,0,0.5);` `/* Fondo oscuro` `*/` `}` `.modal-contenido {` `background-color:` `white;` `padding: 20px;` `border-radius: 10px;` `width: 300px;` `margin: 15% auto;` `text-align: center;` `font-family: Arial;` `box-shadow: 0 0 10px` `gray;` `}` `.botones button {` `margin: 10px;` `padding: 8px 15px;` `font-weight: bold;` `border: none;` `border-radius: 5px;` `cursor: pointer;` `}`	`function mostrarDialogo` `() {` `document.` `getElementById("` `dialogo").style.` `display = "block` `";` `}` `function respuesta(` `valor) {` `document.` `getElementById("` `dialogo").style.` `display = "none"` `;` `document.` `getElementById("` `resultado").` `textContent = "` `Respuesta: " +` `valor;` `}`

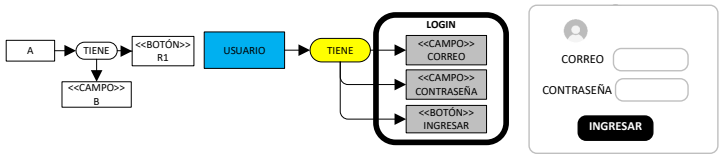
Los *conceptos hoja* representan atributos, *campos* o valores visibles, tales como *botones (con estereotipo), etiquetas* o *listas* (véase la Tabla 4.30). Para mantener la coherencia, se recomienda que los nombres de conceptos hoja en el EP coincidan con los identificadores (*ID* o *name*) de los elementos HTML o de los componentes visuales. Las RE entre conceptos se traducen en jerarquías visuales o agrupaciones dentro de la interfaz, como contenedores o grupos de campos. Es importante aclarar que todas las reglas de equivalencia entre EP y el prototipado visual (véase la Tabla 4.28) son aplicables al desarrollo *front*.

Por otro lado, las RD y RE se implementan como comportamientos o interacciones usando eventos GUI, como *onClick*, *onSubmit*, *onChange* o funciones controladoras en JavaScript. Estas relaciones permiten activar acciones específicas (Seleccionar opciones a través de botones, clics o *tags*) cuando el *usuario interactúa con los componentes visuales* o cuando se generan eventos internos automáticos (por ejemplo, el cambio de estado de un componente) derivando mensajes de notificación (véase la Tabla 4.30).

- **Equivalencia de EP a tecnologías de desarrollo web:** Con el fin de facilitar la implementación de los elementos de la GUI desde EP, se propone la Tabla 4.31 de equivalencias a tecnologías de desarrollo web como Node.js, React y TypeScript. Estas tecnologías se emplean en entornos de desarrollo de aplicaciones web o móviles para manejar componentes de la GUI como listas, menús, formularios, mensajes de notificación, entre otros.
  React es una biblioteca que permite construir interfaces mediante componentes reutilizables y gestión declarativa del estado, mientras que *TypeScript* añade un sistema de tipado estático sobre JavaScript que mejora la estructuración y validación del código durante el desarrollo. Por su parte, *Node.js* es un entorno de ejecución de JavaScript que facilita el desarrollo *front-end* proporcionando herramientas para la automatización de tareas, gestión de dependencias y ejecución de procesos: compilación de código y el despliegue de servidores de desarrollo en el *back-end*. La integración de Node.js con *frameworks* como React es común, permitiendo que el desarrollo *front-end* aproveche sus funcionalidades. En caso de que surjan nuevas tecnologías o librerías para el desarrollo *front-end*, es posible mantener la correspondencia entre los EP y estas tecnologías en tanto se definan equivalencias claras en cuanto a componentes, atributos y relaciones estructurales o dinámicas, asegurando así adaptabilidad y coherencia en el proceso de implementación.

- **Equivalencia de EP a tecnologías de desarrollo móvil:** Para facilitar la implementación de tecnologías como Kotlin Multiplatform, Flutter (Dart) y React Native, se presenta la Tabla 4.32. Estas tecnologías se utilizan en entornos de desarrollo de aplicaciones móviles para gestionar componentes de la GUI como listas, menús, formularios, botones y mensajes de notificación, permitiendo implementar la lógica y la interacción visual del sistema.
  Kotlin Multiplatform es un enfoque que permite compartir lógica de negocio y modelos de datos en distintos sistemas operativos, manteniendo la posibilidad de integrar interfaces nativas en Android e iOS. Por su parte, *Flutter*, basado en el lenguaje *Dart*, ofrece un *framework* multiplataforma que renderiza de forma nativa componentes visuales personalizables mediante *widgets*, garantizando una experiencia consistente entre dispositivos. Finalmente, *React Native* extiende el paradigma de componentes de *React* hacia el desarrollo móvil, permitiendo construir interfaces declarativas y reactivas en JavaScript/TypeScript que se comunican con componentes nativos de cada sistema operativo.

Tabla 4.31. Equivalencia de EP a tecnologías web: Node.js, React y TypeScript. Basado en Zapata (2012) y Villamizar (2019)

<table>
<tr><th>Node.js</th><th>React</th><th>TypeScript</th></tr>
<tr><td colspan="3"> 

Campo, Botón, Formulario</td></tr>
<tr><td>

```
const express = require
    ('express');
const app = express();
app.use(express.json()
);

app.post('/login', (req
    , res) => {
  const { correo,
      contrasena } =
      req.body;
  if (correo &&
      contrasena) {
    res.send("Login
        exitoso");
  } else {
    res.status(400).
        send("Datos
        incompletos");
  }
});

app.listen(3000, () =>
  console.log('Servidor
      en puerto 3000'
    )
);
```

</td><td>

```
import React, {useState
    } from 'react';
function Login() {
 const [correo,
     setCorreo] =
     useState('');
 const [contrasena,
     setContrasena] =
     useState('');
 const handleSubmit = (
     e) => {
  e.preventDefault();
  alert('Correo: ${
      correo},
      Contraseña: ${
      contrasena}');}
  return (<form
      onSubmit={
      handleSubmit}>
 <input
  type="email"
  placeholder="Correo"
  onChange={e =>
      setCorreo(e.
      target.value)}/>
 <input
  type="password"
  placeholder="Contrase
      ña"
  onChange={e =>
      setContrasena(e.
      target.value)}/>
 <button type="submit">
     Ingresar</button>
 </form>);}
export default Login;
```

</td><td>

```
interface Credenciales
    {
  correo: string;
  contrasena: string;
}

function login(data:
    Credenciales):
    void {
  console.log(
    'Correo: ${data.
        correo},
    Contraseña: ${data
        .contrasena}'
  );
}

login({
  correo: "usuario@mail
      .com",
  contrasena: "1234"
});
```

</td></tr>
</table>

Tabla 4.31. (Continuación) Equivalencia de EP a tecnologías web: Node.js, React y TypeScript. Basado en Zapata (2012) y Villamizar (2019)

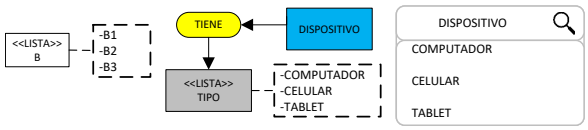
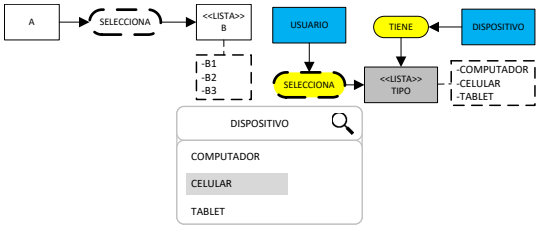
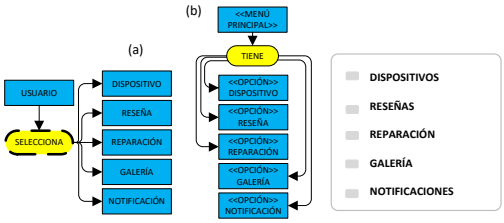
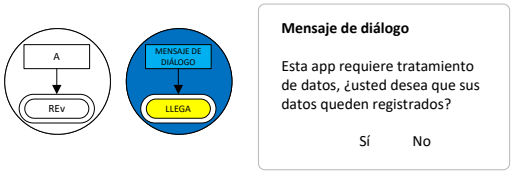
Node.js	React	TypeScript
 Lista		
`app.get('/dispositivos', (req, res) => { res.json(["Computador", "Celular", "Tablet"]); });`	`function Lista() { return ( <ul> <li>Computador</li> <li>Celular</li> <li>Tablet</li> </ul> ); }`	`const dispositivos: string[] = [ "Computador", "Celular", "Tablet" ]; dispositivos.forEach(d => console.log(d));`
 Seleccionar		
`app.post('/seleccionar', (req, res) => { const { opcion } = req.body; res.send('Opcion: ${opcion}'); });`	`function Seleccion() { const [opcion, setOpcion] = useState(''); return ( <select onChange={e => setOpcion(e.target.value)}> <option>Computador</option> <option>Celular</option> <option>Tablet</option> </select> );}`	`let opcion: string; function setOpcion(v: string) { opcion = v; } setOpcion("Celular");`

Tabla 4.31. (Continuación) Equivalencia de EP a tecnologías web: Node.js, React y TypeScript. Basado en Zapata (2012) y Villamizar (2019)

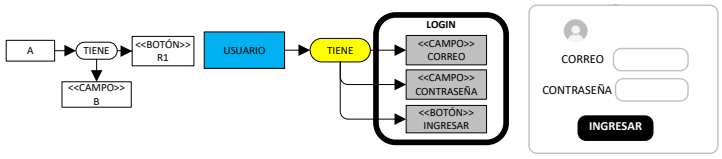
Node.js	React	TypeScript
		
Menús y submenús		

`app.get('/menu',`
`(req, res) => {`
`res.json([`
`"Dispositivos",`
`"Reseñas",`
`"Reparación",`
`"Galería",`
`"Notificaciones"]);`
`});`
	`function Menu() {`
	`return(<nav><ul>`
	`<li>Dispositivos</li>`
	`<li>Reseñas</li>`
	`<li>Reparación</li>`
	`<li>Galería</li>`
	`<li>Notificaciones</`
	`li></ul>`
	`</nav>);}`
		`type Menu = string[];`
		`const menu: Menu = [`
		`"Dispositivos",`
		`"Reseñas",`
		`"Reparación",`
		`"Galería",`
		`"Notificaciones"`
		`];`

		
Mensajes de notificación		

`app.get('/notificacion'`
`,`
`(req, res) => {`
`res.json({`
`titulo: "Mensaje de`
`diálogo",`
`texto: "Esta app`
`requiere`
`tratamiento de`
`datos, ¿usted`
`desea que sus`
`datos queden`
`registrados?"`
`});`
`});`
	`function Mensaje() {`
	`return(<div>`
	`<h3>Mensaje de diá`
	`logo</h3>`
	`<p>"Esta app`
	`requiere`
	`tratamiento de`
	`datos, ¿usted`
	`desea que sus`
	`datos queden`
	`registrados?"</`
	`p>`
	`<button>Sí</button>`
	`<button>No</button>`
	`</div>);}`
		`interface Mensaje {`
		`titulo: string;`
		`texto: string;}`
		`const mensaje: Mensaje`
		`= {`
		`titulo: "Mensaje de di`
		`álogo",`
		`texto: "Esta app`
		`requiere`
		`tratamiento de`
		`datos, ¿usted`
		`desea que sus`
		`datos queden`
		`registrados?"};`

Tabla 4.32. Equivalencia de EP a desarrollo móvil: Kotlin Multiplatform, Flutter (Dart) y React Native. Basado en Mosquera (2021) y Villamizar (2019)

<table>
<tr><th>Kotlin</th><th>Flutter(Dart)</th><th>React Native</th></tr>
<tr><td colspan="3"> 

Campo, Botón, Formulario</td></tr>
<tr><td>

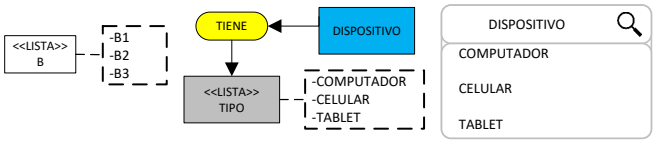
```
data class Credenciales
    (
  val correo: String,
  val contrasena:
      String
)

fun login(c:
    Credenciales):
    String {
  return "Login de ${c.
      correo}"
}
```

</td><td>

```
TextField(decoration:
  InputDecoration(
      labelText: "
      Correo")),
TextField(
  obscureText: true,
  decoration:
    InputDecoration(
        labelText: "
        Contraseña")),
ElevatedButton(
  onPressed: () { print
      ("Ingresar"); },
  child: Text("Ingresar
      "))
```

</td><td>

```
<TextInput placeholder=
    "Correo"/>
<TextInput
  secureTextEntry={true
      }
  placeholder="Contrase
      ña"/>
<Button title="Ingresar
    "
  onPress={() => alert(
      "Login")}/>
```

</td></tr>
<tr><td colspan="3"> 

Lista</td></tr>
<tr><td>

```
val dispositivos =
    listOf(
 "Computador","Celular"
     ,"Tablet")
```

</td><td>

```
ListView(children: [
 Text("Computador"),
 Text("Celular"),
 Text("Tablet")
])
```

</td><td>

```
<FlatList
 data={["Computador","
     Celular","Tablet"
     ]}
 renderItem={({item}) =
     >
   <Text>{item}</Text>}
     />
```

</td></tr>
</table>

Tabla 4.32. (Continuación) Equivalencia de EP a desarrollo móvil: Kotlin Multiplatform, Flutter (Dart) y React Native...

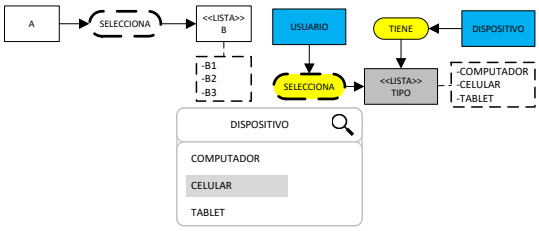
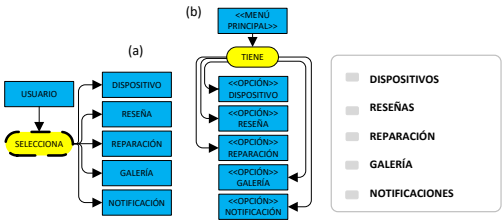
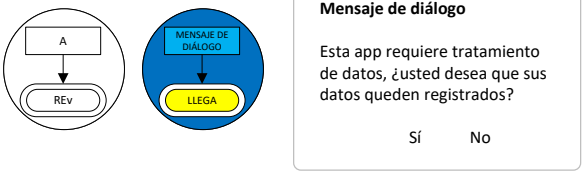
Kotlin	Flutter(Dart)	React Native
 Seleccionar		
`fun select(opcion: String) { println("Seleccion: $opcion") }`	`DropdownButton( items: ["PC","Celular", "Tablet"] .map((e) => DropdownMenuItem ( value: e, child: Text(e))) .toList(), onChanged: (v) { print (v); })`	`<Picker> <Picker.Item label="PC" value="pc"/> <Picker.Item label="Celular" value="cel"/> <Picker.Item label="Tablet" value="tab"/> </Picker>`
 Menús y submenús		
`val menu = listOf("Dispositivos", "Reseñas","Reparación","Galería","Notificación")`	`Drawer( child: ListView( children: [ ListTile(title: Text ("Dispositivos")), ListTile(title: Text ("Reseñas")) ]))`	`<Drawer.Navigator> <Drawer.Screen name="Dispositivos"/> <Drawer.Screen name="Reseñas"/> </Drawer.Navigator>`

Tabla 4.32. (Continuación) Equivalencia de EP a desarrollo móvil: Kotlin Multiplatform, Flutter (Dart) y React Native...

Kotlin	**Flutter(Dart)**	**React Native**
 Mensajes de notificación		

`data class Mensaje(`
`val titulo: String,`
`val texto: String)`
	`showDialog(context:ctx,`
	`builder: (_) =>`
	`AlertDialog(`
	`title: Text("Mensaje de diálogo"),`
	`content: Text("¿Desea que sus datos queden registrados?"),`
	`actions: [`
	`TextButton(onPressed:(){},child: Text("Sí")),`
	`TextButton(onPressed:(){},child: Text("No"))]));`
		`Alert.alert(`
		`"Mensaje",`
		`""Esta app requiere tratamiento de datos, ¿usted desea que sus datos queden registrados?",`
		`[{text:"Sí"}, {text:"No"}])`

Equivalencia de EP a lenguajes de Bases de datos: La base de datos almacena la información estructurada del esquema. Las equivalencias entre EP se representan con las relaciones estructurales mediante los conceptos principales o clases y sus conceptos hoja como se puede observar previamente en la Tabla 4.25, con la especificación del diagrama entidad/relación y la notación SQL (lenguaje de consulta estructurado para bases de datos). También, los conceptos principales se traducen a tablas en bases de datos relacionales (SQL) o en los archivos de almacenamiento que se utilicen en bases de datos no relacionales (NoSQL), por ejemplo, documentos, columnas, entre otras, mediante la consistencia del nombramiento de la tabla o archivo.

A partir de estas reglas se presenta la Tabla 4.33 de equivalencias entre EP a bases de datos. En la tabla se presentan las tecnologías MySQL, PostgresSQL y MongoDB. *MySQL* es un sistema de gestión de bases de datos relacional, que se caracteriza por su rapidez, simplicidad y eficiencia. Utiliza el lenguaje SQL estándar para la definición y manipulación de datos en aplicaciones web y empresariales. Por su parte, *PostgreSQL* es un sistema de gestión de bases de datos relacional y orientado a objetos, que se caracteriza por su extensibilidad y cumplimiento estricto de los estándares SQL. Soporta tipos de datos avanzados, consultas

complejas y características como funciones definidas por el usuario, permitiendo flexibilidad y escalabilidad en los proyectos de software. Desde las bases de datos NoSQL, *MongoDB* es una base de datos orientada a documentos, que almacena la información en formato JSON/BSON. Su modelo flexible permite trabajar con datos no estructurados o semiestructurados, ofreciendo escalabilidad horizontal y un enfoque ágil para aplicaciones que requieren alta disponibilidad y gestión eficiente de grandes volúmenes de datos. Existen diversas bases de datos en el grupo NoSQL, los cuales también se pueden beneficiar de esta equivalencia.

Los *tipos de datos* para los conceptos hoja son: texto (sin símbolo), fecha(//), número (#), booleano (?) y correo (@), los cuatro últimos se definen después de usar el símbolo de dos puntos (Chaverra, 2011). Por ejemplo, el concepto hoja tipo *número* "id" y el concepto hoja tipo *correo* "email" en la Tabla 4.33. Los conceptos hoja también permiten ser conceptos obligatorios usando la conexión de doble enlace, que indica que este valor no puede ser nulo, por ejemplo el "id" y el "email" del "paciente", siendo este último además de obligatorio único al usar la relación estructural *tiene_único* en la Tabla 4.33.

Las acciones atómicas que se representan como una relación dinámica pueden ser *insert, select, update* y *delete*, las cuales se definen con verbos similares, por ejemplo, si la relación dinámica es "ingresar" la acción atómica es *insert* (véase la Tabla 4.33).

Tabla 4.33. Equivalencias EP a bases de datos. Basado en Chaverra (2011)

MySQL	**PostgreSQL**	**MongoDB**
		

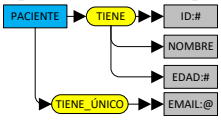
`CREATE TABLE A (`
`);`
 	
`CREATE TABLE Paciente (`
`);`
	`CREATE TABLE A (`
	`);`
	
	`CREATE TABLE Paciente (`
	`);`
		`db.A.insertOne({`
		`});`
		
		`db.paciente.insertOne({`
		`});`

`CREATE TABLE C (`
`B INT PRIMARY KEY);`
 	
`CREATE TABLE A (`
`FOREIGN KEY (B)`
`REFERENCES A(B));`
 	
`CREATE TABLE Usuario (`
`id INT PRIMARY KEY);`
 	
`CREATE TABLE Paciente (`
`FOREIGN KEY (id)`
`REFERENCES Usuario(id));`
	`CREATE TABLE C (`
	`B INT PRIMARY KEY);`
	
	`CREATE TABLE A (`
	`FOREIGN KEY (B)`
	`REFERENCES A(B));`
	
	`CREATE TABLE Usuario (`
	`id INT PRIMARY KEY);`
	
	`CREATE TABLE Paciente (`
	`FOREIGN KEY (id)`
	`REFERENCES Usuario(id));`
		`db.C.insertOne({`
		`_B: valor);`
		
		`db.A.insertOne({`
		`C_B: valor`
		`});`
		`db.usuario.insertOne({`
		`_id: 1, nombre: "Juan`
		`Perez"});`
		
		`db.paciente.insertOne({`
		`_id: 1,`
		`usuario_id: 1,`
		`edad: 65,`
		`celular: "3001234567"});`

Tabla 4.33. (Continuación) Equivalencias EP a bases de datos...

<table>
<tr><th>MySQL</th><th>PostgreSQL</th><th>MongoDB</th></tr>
<tr><td colspan="3"> 

Concepto hoja con tipo de dato, concepto obligatorio y único

</td></tr>
<tr><td>

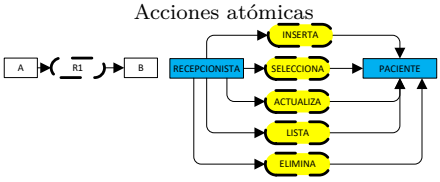
```
CREATE TABLE A (
  B,
);

CREATE TABLE Paciente (
  id INT PRIMARY KEY NOT
       NULL,
  nombre VARCHAR(100),
  edad INT,
  email VARCHAR(100) UNIQUE
       NOT NULL
);
```

</td><td>

```
CREATE TABLE A (
  B,
);

CREATE TABLE Paciente (
  id INT PRIMARY KEY NOT
       NULL,
  nombre VARCHAR(100),
  edad INT,
  email VARCHAR(100) UNIQUE
       NOT NULL
);
```

</td><td>

```
db.A.createIndex({"A":1}, {
     unique: true});
db.A.insertOne({
  B: valor,});

db.paciente.createIndex({"id
     ":1}, {unique: true});
db.paciente.createIndex({"
     email":1}, {unique:
     true});

db.paciente.insertOne({
  id: 1,
  nombre: "Juan Perez",
  edad: 65,
  email: "juan@eafit.edu.co"
       });
```

</td></tr>
<tr><td colspan="3">Acciones atómicas

</td></tr>
<tr><td>

```
INSERT INTO Paciente
(id,nombre,edad,email)
VALUES
(1027,'Leticia',75,'
     leti@eafit.edu.co')
     ;

SELECT * FROM Paciente
     WHERE id = 1027;

UPDATE Paciente
SET email = 'leti2@eafit
     .edu.co', edad = 66
WHERE id = 1027;

SELECT * FROM Paciente;

DELETE FROM Paciente
     WHERE id = 1027;
```

</td><td>

```
INSERT INTO Paciente
(id,nombre,edad,email)
VALUES
(1027,'Leticia',75,'
     leti@eafit.edu.co');

SELECT * FROM Paciente WHERE
      id = 1;

UPDATE Paciente
SET email = 'leti2@eafit.edu
     .co', edad = 66
WHERE id = 1027;

SELECT * FROM Paciente;

DELETE FROM Paciente WHERE
     id = 1027;
```

</td><td>

```
db.Paciente.insertOne({
  nombre: "Leticia",
  email: "leti2@eafit.edu.co
       ",
  edad: 66
});

db.Paciente.findOne({ email:
      "leti2@eafit.edu.co" }
     );

db.A.updateOne(
  { id: 1027 },
  { $set: { edad: 66 } }
);
db.Paciente.find();

db.Paciente.deleteOne({
     email: "leti2@eafit.edu
     .co" });
```

</td></tr>
</table>

Tabla 4.33. (Continuación) Equivalencias EP a bases de datos...

<table>
<tr><th>MySQL</th><th>PostgreSQL</th><th>MongoDB</th></tr>
<tr><td colspan="3">
<table>
<tr><th colspan="4">B</th></tr>
<tr><th>D</th><th>E</th><th>F</th><th>G</th></tr>
<tr><td>valor1</td><td>valor1</td><td>valor1</td><td>valor1</td></tr>
<tr><td>valor2</td><td>valor2</td><td>valor2</td><td>valor2</td></tr>
</table>
<table>
<tr><th colspan="4">PACIENTE</th></tr>
<tr><th>ID</th><th>NOMBRE</th><th>EDAD</th><th>EMAIL</th></tr>
<tr><td>1027</td><td>Leticia Londoño</td><td>75</td><td>leti2@eafit.edu.co</td></tr>
<tr><td>1028</td><td>Ottoniel Noreña</td><td>68</td><td>otto3@eafit.edu.co</td></tr>
</table>
</td></tr>
<tr><td>

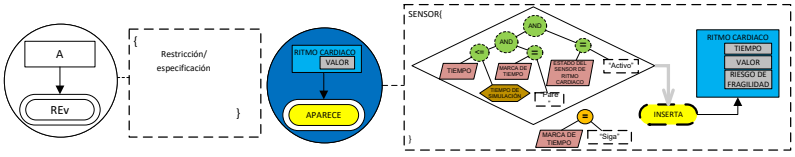
```
INSERT INTO A (D,E,F,G)
     VALUES
(valor1, valor1, valor1,
     valor1),
(valor2, valor2, valor2,
     valor2);

INSERT INTO Paciente (id,
     nombre, edad, email)
     VALUES
(1027, 'Leticia Londoño',
     75, 'leti2@eafit.edu.co
     '),
(1028, 'Ottoniel Noreña',
     68, 'otto3@eafit.edu.co
     ');
```

</td><td>

```
INSERT INTO A (D,E,F,G)
     VALUES
(valor1, valor1, valor1,
     valor1),
(valor2, valor2, valor2,
     valor2);

INSERT INTO Paciente (id,
     nombre, edad, email)
     VALUES
(1027, 'Leticia Londoño',
     75, 'leti2@eafit.edu.co
     '),
(1028, 'Ottoniel Noreña',
     68, 'otto3@eafit.edu.co
     ');
```

</td><td>

```
db.A.insertMany([
  {
    D: valor1,
    E: valor1,
    F: valor1,
    G: valor1
  };
db.Paciente.insertMany([
  {
    id: 1027,
    nombre: "Leticia Londoño
         ",
    edad: 75,
    email: "leti2@eafit.edu.
         co"
  },
  {
    id: 1028,
    nombre: "Ottoniel Noreña
         ",
    edad: 68,
    email: "otto3@eafit.edu.
         co"
  }
]);
```

</td></tr>
<tr><td colspan="3"> 
</td></tr>
<tr><td>

```
CREATE TRIGGER A_REv
BEFORE INSERT ON A
FOR EACH ROW
BEGIN
 Restricción/especificación
END;
```

</td><td>

```
CREATE OR REPLACE FUNCTION A
     _REv()
RETURNS TRIGGER AS $$
BEGIN
 Restricción/especificación
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER A_REv
BEFORE INSERT ON A
FOR EACH ROW
EXECUTE FUNCTION A_REv();
```

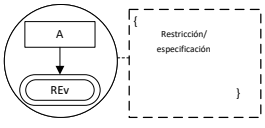
</td><td>

```
//Mongo no tiene triggers
     pero se pueden simular
     usando funciones
     Mongoose}

const mongoose = require('
     mongoose');

const A = new mongoose.
     Schema({
   B: valor
});

const A = mongoose.model('A'
     , ASchema);
Restricción/especificación
```

</td></tr>
</table>

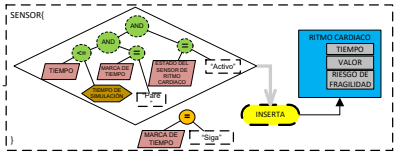
Tabla 4.33. (Continuación) Equivalencias EP a bases de datos...

MySQL




```
CREATE TRIGGER RitmoCardiaco
     _Aparece
BEFORE INSERT ON
     RitmoCardiaco
FOR EACH ROW
BEGIN
  DECLARE riesgo VARCHAR(10)
  DECLARE tiempo INT
  DECLARE valor INT;

  IF NEW.tiempo <= New.
       TiempoSimulacion OR
       NEW.marcaTiempo ='
       Pare' AND
       estadoSensor = '
       Activo' THEN
 SET valor=NEW.valor;
 IF valor <= 50 OR valor >=
      100 THEN
    SET riesgo = 'Alto';
  ELSE
    SET riesgo = 'Bajo';
  END IF;
  SET Tiempo;
END;
```

PostgreSQL




```
CREATE OR REPLACE FUNCTION
     RitmoCardiaco_Aparece()
RETURNS TRIGGER AS $$
BEGIN
  DECLARE
  riesgo VARCHAR(10)
  tiempo INT
  valor INT;

IF NEW.tiempo <= New.
     TiempoSimulacion OR NEW
     .marcaTiempo ='Pare'
     AND estadoSensor = '
     Activo' THEN

 SET valor=NEW.valor;
 IF valor <= 50 OR valor >=
      100 THEN
    SET riesgo = 'Alto';
  ELSE
    SET riesgo = 'Bajo';
  END IF;
  SET Tiempo;
 RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER RitmoCardiaco
     _Aparece
BEFORE INSERT ON
     RitmoCardiaco
FOR EACH ROW
EXECUTE FUNCTION
     RitmoCardiaco_Aparece()
     ;
```

MongoDB

```
const mongoose = require('
     mongoose');

const ritmoCardiacoSchema =
     new mongoose.Schema({
 valor: { type: Number,
      required: true},
 marcaTiempo: {type: String,
      required: true},
 riesgo: {type: String,
      default: null},});

ritmoCardiacoSchema.pre('
     save', function(next) {
 const nuevoValor = this.
      valor;
 const nuevaMarcaTiempo =
      this.marcaTiempo;
    if (nuevaMarcaTiempo ===
         'Pare') {
       this.riesgo = 'N/A';
    } else {
        if (nuevoValor <= 50
             -- nuevoValor
            >= 100) {
           this.riesgo = '
               Alto';
        } else {
           this.riesgo = '
               Bajo';}}
    next();});
const RitmoCardiaco =
     mongoose.model('
     RitmoCardiaco',
     ritmoCardiacoSchema);
if (require.main === module)
     {
    conectarDB();}
```

Desde los EP ejecutables se derivan tablas que también representan los posibles valores que tendrán las tablas de la base de datos por lo que se pueden entender como una acción atómica de *insert*. Por otro lado, los eventos son equivalentes a *triggers* en bases de datos por lo que se pueden crear de acuerdo con la restricción o especificación que contenga un evento (véase la Tabla 4.33).

Capítulo 5

Caso de laboratorio

Un caso de laboratorio en el contexto del diseño de sistemas y software se presenta en un ambiente controlado y planificado para aplicar, evaluar y refinar artefactos (modelos, métodos, herramientas o sistemas) mediante un ciclo estructurado de análisis, diseño, desarrollo y evaluación. Está orientado a demostrar la utilidad del artefacto propuesto y a generar conocimiento científico aplicable al dominio (Wieringa, 2014).

El caso de laboratorio propuesto es un sistema de salud IoT (internet de las cosas, por sus siglas en inglés), para la evaluación de riesgo de fragilidad de los adultos mayores. Este caso laboratorio tiene como objetivo demostrar cómo los esquemas preconceptuales se pueden utilizar en las fases de análisis y diseño mediante sus características estructurales, dinámicas, télicas y eventuales, permitiendo observar la lógica de la aplicación necesaria para la fase del desarrollo.

Tabla 5.1. Metodología para desarrollar el caso de laboratorio. Basado en Wieringa (2014) y Wohlin *et al.* (2012)

Fase	**Definición**
Análisis	Entender y definir lo que el sistema debe hacer. Incluye la identificación de necesidades, requisitos funcionales y restricciones del dominio del problema. Se busca comprender el dominio del sistema y sus actores.
Diseño	Seleccionar técnicas de modelado (EP, UML, prototipos, etc.), definir elementos, datos y arquitectura del sistema. El diseño facilita la comprensión del sistema y proporciona una guía para transformar los requisitos en una solución técnica viable.
Desarrollo	Es donde las especificaciones y el diseño se transforman, se implementa la solución propuesta por medio de la codificación. En esta fase se construyen los artefactos del sistema.
Ejecución & Evaluación	Analizar los resultados y evaluación frente a los objetivos y requisitos definidos. También se considera el aprendizaje obtenido.

5.1. Análisis

La *fragilidad* en pacientes, especialmente en adultos mayores, es un fenómeno que se viene estudiando en la medicina moderna. Este concepto está estrechamente relacionado con la capacidad reducida de los pacientes para resistir y recuperarse de eventos estresantes. En este contexto, el monitoreo continuo de indicadores de salud como el *ritmo cardíaco* puede ser una herramienta que ayuda a la detección de riesgos y la toma de decisiones oportunas sobre el tratamiento y manejo de los pacientes (Rodríguez-Fórtiz *et al.* 2019).

Para el análisis de los requisitos del sistema, se emplean artefactos como el *user story mapping*, el cual permite organizar y visualizar de manera estructurada a los posibles usuarios del sistema como la "recepcionista", el "paciente" y el personal del "hospital", así como las distintas funcionalidades a desarrollar mediante HU. Este enfoque facilita la identificación de funcionalidades como "ingresar paciente" y "aparecer ritmo cardíaco", este último se integra en la HU épica "monitoreo del riesgo de fragilidad" junto con el "cálculo de riesgo de fragilidad" y la "notificación del riesgo de fragilidad" que surgen automáticamente (véase la Figura 5.1).

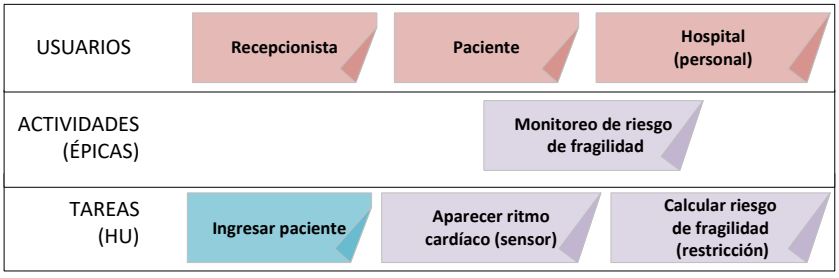


Figura 5.1. *User story mapping* del caso de laboratorio

Además, se describen las HU para especificar en lenguaje natural la lógica de negocio y posteriormente representar las funcionalidades en EP. A partir de los criterios de aceptación y restricciones, se puede representar respectivamente la lógica interna del elemento y la forma en que será medida la funcionalidad. Por ejemplo, la HU 001 "Ingresar paciente" describe una interacción dinámica en la que el actor "recepcionista" ingresa los datos del paciente en el sistema; esta funcionalidad se representa como una triada de relación dinámica en EP (véase la Tabla 5.2).

Por otro lado, la HU 002 "Monitoreo del ritmo cardíaco" corresponde a una historia épica, ya que integra el registro del ritmo cardíaco, el cálculo de riesgo de fragilidad y las notificaciones. Esta historia resume una funcionalidad más compleja que incluye condiciones, restricciones y reglas de negocio que se representan mediante aglutinadores, árboles de decisión y especificaciones dentro de los EP (véase la Tabla 5.3).

Tabla 5.2. Historia de usuario: Ingresar paciente

Historia de usuario	
ID	001
Nombre	Ingresar paciente
Actor	Recepcionista
Descripción	Como recepcionista quiero ingresar la información del paciente para que quede almacenada en el sistema del hospital.
Criterios de aceptación	▪ Cuando se activa la ocurrencia y se ingresa el ID, nombre, edad, celular y se selecciona la referencia del paciente y el nit del hospital, se debe cumplir que se guarde correctamente. ▪ Cuando no se ingresa algún campo, se debe cumplir que el sistema muestre un mensaje de "campo obligatorio".

Tabla 5.3. Historia de usuario épica: Monitoreo del riesgo de fragilidad

Historia de usuario	
ID	002
Nombre	Monitoreo de riesgo de fragilidad
Actor	Sistema automático
Descripción	Como sistema, quiero monitorear continuamente el ritmo cardíaco del paciente para calcular el riesgo de fragilidad y enviar una notificación cuando se detecte un riesgo alto.
Criterios de aceptación	▪ Se debe cumplir que se incluya la historia 002: "Aparecer ritmo cardíaco". ▪ Cuando el ritmo cardíaco es menor o igual a 50 lpm o mayor o igual a 100 lpm, se debe cumplir que el valor del riesgo sea "Alto". ▪ Cuando el riesgo es "Alto", se debe cumplir que se actualice la tasa de riesgo de fragilidad y se envíe una notificación. ▪ En los demás casos, el riesgo se considera "Bajo".
Restricción	▪ Si el ritmo cardíaco es $\leq$ 50 lpm o $\geq$ 100 lpm, entonces el sistema debe activar la notificación y sumar a la tasa de riesgo. ▪ La tasa de riesgo solo se actualiza si la marca de tiempo lo permite (por ejemplo, estado del sensor activo). ▪ La ocurrencia de que un paciente llegue es aleatoria para la simulación. ▪ El tiempo de simulación y otras variables como estado del sensor, promedio y tasa de ritmo cardíaco se deben inicializar como variables globales. ▪ El tiempo incrementa cada hora.

Finalmente, la HU 003 "Aparecer ritmo cardíaco" se refiere a una funcionalidad automática en la que un sensor, ubicado en un dispositivo como un reloj inteligente, genera lecturas del ritmo cardíaco del paciente. Esta operación no involucra un actor humano, sino un evento

que se activa bajo ciertas condiciones y es representada como una triada de evento en los EP (véase la Tabla 5.2).

Tabla 5.4. Historia de usuario: Aparecer ritmo cardíaco

Historia de usuario	
ID	003
Nombre	Aparecer ritmo cardíaco (sensor)
Actor	Sistema automático
Descripción	Como sistema, quiero capturar el ritmo cardíaco desde el sensor para almacenarlo y evaluar el riesgo de fragilidad.
Criterios de aceptación	▪ Cuando el tiempo $\leq$ tiempo de simulación y la marca de tiempo es "Pare" y el estado del sensor es "Activo", se debe cumplir que se inserte el valor de ritmo cardíaco. ▪ Cuando no se cumplen las condiciones, no se debe insertar ningún dato y se debe mantener la marca de tiempo como "Siga".

5.2. Diseño

Al emplear una representación visual, los EP permiten a analistas, diseñadores, desarrolladores y científicos comprender los elementos del dominio de un producto de software, facilitando la comunicación entre el equipo de desarrollo y los interesados, en la identificación de actores, aquellos que interactúan o activan funcionalidades en el sistema, las funcionalidades y conceptos del dominio. Los actores del caso de laboratorio son adultos mayores (pacientes) y personal administrativo (recepcionista) de los hospitales.

5.2.1. Esquema preconceptual general

En la Figura 5.2 se presenta un primer EP, que ofrece una vista general o una abstracción de nivel uno, integrando las características estructurales, dinámicas, télicas y eventuales del software.

Características estructurales: Las características estructurales en la Figura 5.2 incluyen los conceptos clase: "hospital", "usuario", "recepcionista", "paciente", "reloj inteligente" y "ritmo cardíaco", junto con sus relaciones estructurales.

El concepto "hospital" se detalla con tres conceptos hoja ("nit", "nombre" y "dirección"). De manera similar, un "usuario" cuenta con dos conceptos hoja ("id" y "nombre"), de los cuales "recepcionista" hereda estos mismos conceptos. Por su parte, el "paciente" además de heredar "id", y "nombre" tiene dos conceptos hoja adicionales ("edad" y "celular").

Además, el "paciente" está asociado al concepto clase "ritmo cardíaco", que tiene tres conceptos hoja obligatorios (no vacíos) que se identifican con un doble enlace ("tiempo", "valor" y "riesgo de fragilidad", este último clasificado como "alto" o "bajo"). También el "reloj inteligente" que se relaciona a un "paciente" tiene una única dirección (tiene_único) "IP" y obligatoria (doble enlace) y adicionalmente tiene una "referencia" y "marca".

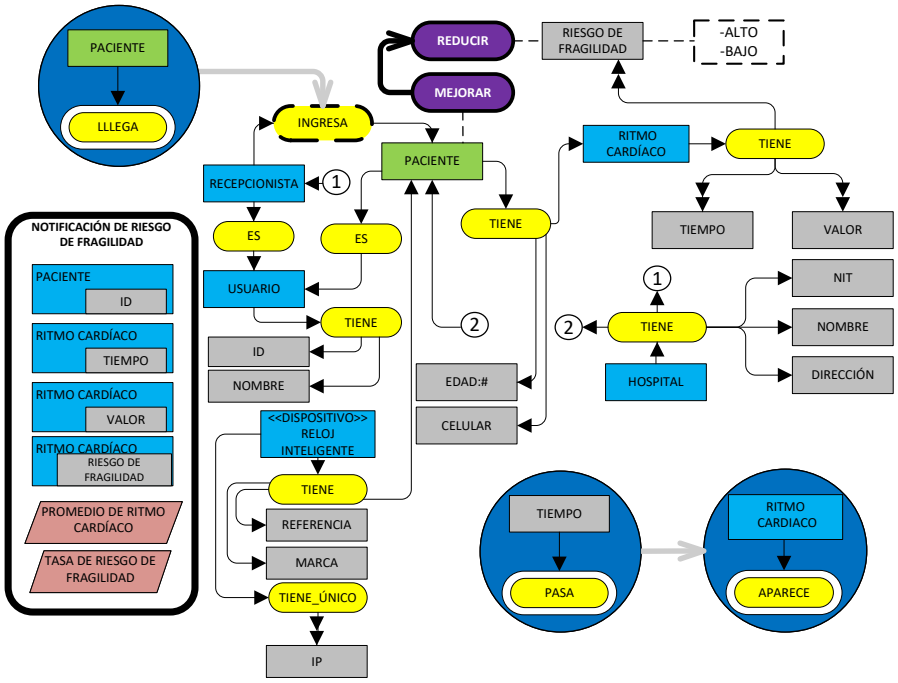


Figura 5.2. Esquema preconceptual general

Nótese que los conceptos principales se encuentran en color azul. Una excepción es el concepto "paciente", que aparece en color verde, lo cual señala que es una clase que relaciona con varias entidades: en este caso, se conecta con "hospital" y "reloj inteligente", evidenciando una relación estructural de agregación desde ambas clases hacia él.

El concepto "reloj inteligente" se destaca además por utilizar un *concepto estereotipado*, representado como «dispositivo», lo que permite especificar su naturaleza dentro del dominio y facilitar su interpretación, por ejemplo, al integrarse con elementos de interfaz o en un contexto técnico.

Por su parte, los conceptos hoja se visualizan en color gris, lo que indica que representan atributos o propiedades de una clase. Dentro de estos, el concepto "edad" ejemplifica el uso del *tipo de dato*, marcado con el símbolo (#), correspondiente a un valor numérico.

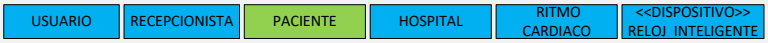
Adicionalmente, se incluye la representación de una "notificación" del proceso de monitoreo

del “riesgo de fragilidad”, que se agrupa dentro de un *marco*. Este marco agrupa los conceptos hoja “tiempo”, “valor” y “riesgo de fragilidad” del “ritmo cardíaco”. Asimismo, se integran dentro del marco dos variables independientes (paralelogramos rosados): “promedio de ritmo cardíaco” y “tasa de riesgo de fragilidad”. Estas variables permiten capturar valores derivados del sistema que se calculan y usan en diferentes etapas del monitoreo.

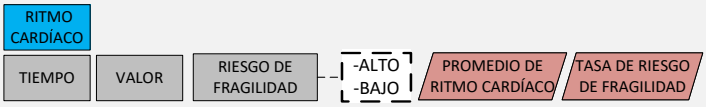
Resumen guía para construir características estructurales

Para modelar las características estructurales de un sistema en los EP, se recomienda seguir los siguientes pasos:

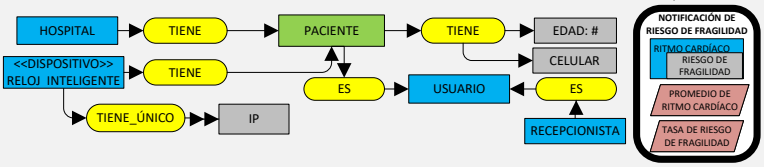
(1) **Incluir actores y conceptos principales:** Estos definen las entidades del dominio (sustantivos o sintagma nominal en singular). Por ejemplo, los conceptos “recepcionista” y “ritmo cardíaco” (color azúl, verde si tiene varias conexiones hacia el concepto). En caso de requerir más detalle, se utiliza un concepto estereotipado, por ejemplo, «dispositivo» en el concepto “reloj inteligente”.



(2) **Agregar los conceptos hoja y variables:** Estos representan atributos (color gris) de los conceptos principales, por ejemplo, “tiempo”, “valor” y “riesgo de fragilidad” son conceptos hoja de “ritmo cardíaco”. También se relacionan a este concepto, las variables (color rosado), por ejemplo, “promedio de ritmo cardíaco” y “tasa de riesgo de fragilidad”. Si el concepto hoja tiene valores determinados, se conectan mediante un enlace *concepto-nota* y se integran en un *valor-nota* como lista. Por ejemplo, los valores “alto” o “bajo” del concepto hoja “riesgo de fragilidad”.



(3) **Establecer las relaciones y agrupaciones:** Una vez definidos los elementos, se generan las relaciones estructurales (por ejemplo, “tiene” o “tiene_único”(clase a concepto hoja) y “es” (herencia) en color amarillo) y se aplican las agrupaciones mediante *marcos* o *conexiones* (doble cuando es obligatorio) siguiendo las reglas del libro. Al generar las relaciones se verifican los conceptos clase que tienen varias conexiones (color verde).



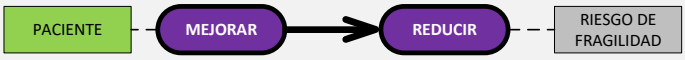
Características télicas: Las características télicas en la Figura 5.2 involucran las expectativas “reducir riesgo de fragilidad” y “mejorar paciente”. Ambos objetivos se unen mediante un enlace *implicación* (color negro) para indicar que si se reduce el riesgo de fragilidad el paciente puede mejorar.

Resumen guía para construir características télicas

Para modelar las características télicas de un sistema en los EP, se recomienda seguir los siguientes pasos:

(1) **Definir los objetivos/expectativas o problemas:** Estos se definen mediante el uso de las relaciones de logro (verbo de logro en infinitivo). Las relaciones de logro se pueden relacionar a conceptos, relaciones de dinámicas y estructurales mediante un enlace *concepto-nota*. Por ejemplo, los objetivos "reducir riesgo de fragilidad y "mejorar paciente" (color morado, letra blanca). El uso de adverbios, negación y connotación negativa para esta definición junto con las demás reglas se pueden revisar en el libro.

(2) **Establecer la secuencia:** Las relaciones causa-efecto entre los objetivos y problemas se representan mediante un enlace *implicación* (color negro) desde una relación de logro hacia otra.



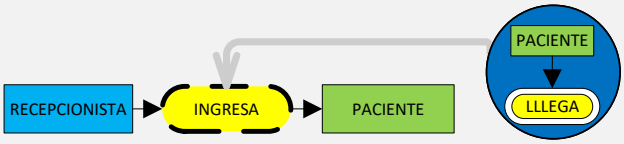
Características dinámicas: Las características dinámicas en la Figura 5.2 se expresan mediante la triada "recepcionista ingresa paciente" donde "ingresar" es una acción atómica que permite insertar información del paciente ("id", "nombre", "edad" y "celular") en la base de datos.

Resumen guía para construir características dinámicas

Para modelar las características dinámicas de un sistema en los EP, se recomienda seguir los siguientes pasos:

(1) **Representar las funcionalidades:** A partir del análisis de los requisitos funcionales, revisar cuáles de estos los realizan los diferentes actores del sistema. Las relaciones dinámicas se conforman mediante una triada entre concepto (rol de agente), verbo dinámico (conjugado, color amarillo) y otro concepto sobre el que recae la acción, mediante el enlace *conexión*. Por ejemplo, la relación dinámica "recepcionista ingresa paciente".

(2) **Establecer la secuencia:** Las relaciones causa-efecto entre las relaciones dinámicas, condicionales y los eventos permiten observar el flujo o la secuencia de acciones del sistema mediante un enlace *implicación* (color gris). Revisar las diferentes reglas de las características dinámicas en el libro.



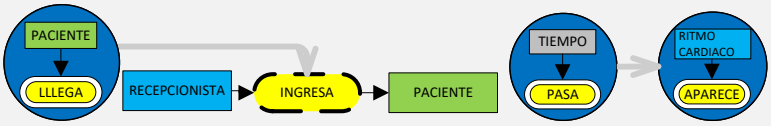
Características eventuales: Las características eventuales en la Figura 5.2 se expresan mediante los eventos "paciente llega", "tiempo pasa", "paciente llega" y "ritmo cardíaco aparece" permitiendo el cálculo y la "notificación del ritmo cardíaco" en el monitoreo de estos pacientes.

Resumen guía para construir características eventuales

Para modelar las características eventuales de un sistema en los EP, se recomienda seguir los siguientes pasos:

(1) **Representar las funcionalidades automáticas:** A partir del análisis de los requisitos funcionales, revisar cuáles de estos se realizan de forma automática (sin un actor). Los eventos se conforman mediante un círculo que agrupa un concepto y una relación eventual usualmente, aunque también puede tener dos conceptos o no tener ninguno mediante el enlace *conexión*. Por ejemplo, los eventos "paciente llega", "tiempo pasa", "paciente llega" y "ritmo cardíaco aparece".

(2) **Establecer la secuencia:** Las relaciones causa-efecto entre las relaciones dinámicas y los eventos ppermiten observar el flujo o la secuencia de acciones del sistema mediante un enlace *implicación* (color gris). Revisar las diferentes reglas de las características eventuales en el libro.



5.2.2. Esquema preconceptual detallado

En la Figura 5.3, se muestra el nivel de abstracción dos, un esquema preconceptual detallado derivado del EP general en la Figura 5.2. Este esquema permite observar la lógica completa de la aplicación y su simulación mediante el uso de las especificaciones y restricciones, las cuales se utilizan en las condiciones iniciales, los conceptos, las relaciones dinámicas, las relaciones de logro y las relaciones eventuales.

Características estructurales: Las características estructurales del EP detallado (véase la Figura 5.3) permanecen iguales a las de la vista general (véase la Figura 5.2), es decir, que se incluyen los conceptos clase: "hospital", "usuario", "recepcionista", "paciente", "reloj inteligente" y "ritmo cardíaco", junto con sus respectivas relaciones estructurales y conceptos hoja. De los cuales se relaciona una tabla por cada concepto principal (véanse las Tablas 5.5, 5.7, 5.8, 5.9, 5.10 y 5.11). Además, se introduce una restricción al concepto hoja "riesgo de fragilidad", que asigna el valor "alto" cuando el ritmo cardíaco es inferior a 50 latidos por minuto (lpm) o superior a 100 lpm, y "bajo" en caso contrario. Esta restricción permite al desarrollador programar la condición que permite generar el resultado del "riesgo de fragilidad".

Además, se incorporan las *condiciones iniciales* dentro de las características estructurales de la aplicación, estableciendo los valores iniciales de los parámetros y variables independientes que se utilizan para controlar la simulación. Entre las condiciones iniciales, se encuentra el "tiempo" inicializado en "0", el "tiempo de simulación" configurado en "72 horas" y la "marca de tiempo" con el valor "pare". Estas condiciones se usan para gestionar el flujo temporal, ya que la simulación tiene una duración total de 72 horas, mientras que la "marca de tiempo"

detiene el avance temporal para permitir el registro de datos del sistema hasta que su estado cambie a “siga”.

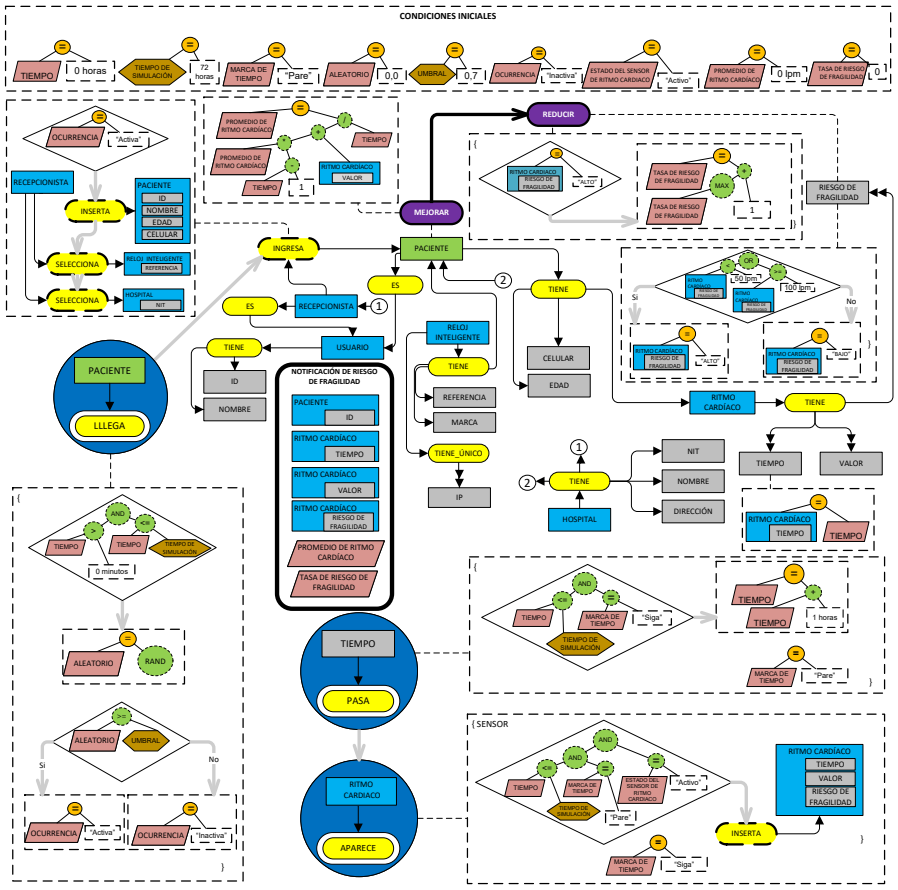


Figura 5.3. Esquema preconceptual detallado

Adicionalmente, las condiciones “aleatorio” con valor “0,0”, “umbral” con valor “0,7” y “ocurrencia” establecida como “inactiva” se utilizan para generar de manera aleatoria un evento de llegada. Este evento, identificado como “paciente llega”, se activa cuando el valor de “ocurrencia” cambia a “activa”. Finalmente, las tres variables: “estado del sensor de ritmo cardíaco” configurada en “activo”, “promedio de ritmo cardíaco” con valor inicial de “0 lpm” y “tasa de riesgo de fragilidad” igual a “0” se emplean para almacenar la información que surge a partir de las lecturas del sensor de ritmo cardíaco.

Tabla 5.5. Análisis de datos de la tabla HOSPITAL

HOSPITAL		
NIT	**NOMBRE**	**DIRECCIÓN**
890901826-2	Pablo Tobón Uribe	Diagonal 80 #76-08, Robledo
890906347-9	Manuel Uribe Ángel	Diagonal 31 36ª sur 80, Envigado

Tabla 5.6. Análisis de datos de la tabla USUARIO

USUARIO	
ID	**NOMBRE**
RE1	Luz Elena Villa
RE2	Natalia Beltrán
1027	Leticia Londoño
1028	Ottoniel Noreña

Tabla 5.7. Análisis de datos de la tabla RECEPCIONISTA

RECEPCIONISTA	
ID	**HOSPITAL.NIT**
RE1	890901826-2
RE2	890906347-9

Tabla 5.8. Análisis de datos de la tabla RELOJ INTELIGENTE

RELOJ INTELIGENTE			
CÓDIGO	**REFERENCIA**	**MARCA**	**IP**
M1	GEAR S7	Samsung	216.58.152.97
M2	GEAR S11	Samsung	216.58.152.97

Tabla 5.9. Análisis de datos de la tabla PACIENTE

PACIENTE (ADULTO MAYOR)					
USUARIO. ID	**USUARIO. NOMBRE**	**EDAD**	**CELULAR**	**RELOJ.REFE-RENCIA**	**HOSPITAL. NIT**
1027	Leticia Londoño	75	3156782399	M1	890906347-9
1028	Ottoniel Noreña	69	3102004534	M2	890901826-2

Tabla 5.10. Análisis de datos de la tabla RITMO CARDÍACO, RELOJ INTELIGENTE **M1**

RITMO CARDÍACO		
TIEMPO (Horas)	**VALOR(lpm)**	**RIESGO DE FRAGILIDAD**
1	90	Bajo
2	95	Bajo
3	120	Alto

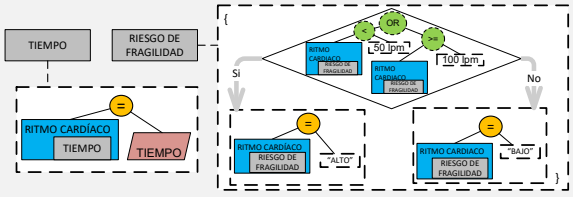
Tabla 5.11. Análisis de datos de la tabla RITMO CARDÍACO, RELOJ INTELIGENTE **M2**

RITMO CARDÍACO		
TIEMPO (Horas)	**VALOR(lpm)**	**RIESGO DE FRAGILIDAD**
1	50	Bajo
2	48	Bajo
3	68	Bajo

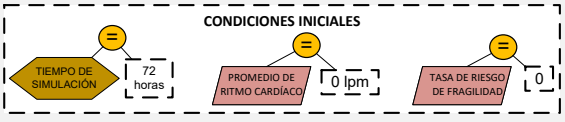
Resumen guía para especificar características estructurales

Para especificar las características estructurales de un sistema en los EP, se recomienda seguir los siguientes pasos:

(1) **Definir la restricción/especificación de los conceptos:** Algunos conceptos tienen restricciones o especificaciones. Por ejemplo, el concepto hoja "tiempo" se relaciona con una especificación que incluye internamente la asignación del valor "tiempo" desde la variable "tiempo". El concepto hoja "riesgo de fragilidad" presenta una restricción que involucra una condición para asignar el "riesgo de fragilidad".



(2) **Incluir condiciones iniciales:** Se incluyen condiciones iniciales como variables y parámetros con sus respectivos valores iniciales. Por ejemplo, el parámetro "tiempo de simulación" y las variables "promedio de ritmo cardíaco" y "tasa de riesgo de fragilidad".



(3) **Relacionar datos:** Para la comprensión de los datos que pueden tener los conceptos hoja en la base de datos, se pueden utilizar *tablas* con los posibles valores del sistema. Por ejemplo, la Tabla 5.8 que relaciona los datos del concepto principal "reloj inteligente" con sus conceptos hoja.

Características télicas: El flujo culmina con las relaciones de logro “mejorar paciente” y “reducir el riesgo de fragilidad”, que representan las características télicas del sistema. Estas relaciones permiten evaluar los indicadores de desempeño clave del “paciente” y su “riesgo de fragilidad”.

La relación de logro “reducir el riesgo de fragilidad” incluye una especificación detallada que cuantifica la “tasa de riesgo de fragilidad” en estado “alto”, estableciendo como objetivo su reducción, y el “promedio de ritmo cardíaco” para medir el progreso del paciente durante el tiempo de evaluación. Este proceso está orientado a lograr una mejora en el estado del paciente. El “promedio de ritmo cardíaco” se calcula mediante la ecuación 5.1, promedio aritmético que se aplica a mediciones de tiempo discreto en horas y se traduce conceptualmente en la ecuación 5.2, para su representación en la especificación. Los datos resultantes se registran en las variables correspondientes y se integran dentro de la “notificación de riesgo de fragilidad”.

$$\bar{x}_n = \frac{\bar{x}_{n-1}(n-1) + x_n}{n} \tag{5.1}$$

- $\bar{x}_n$: promedio actualizado
- $\bar{x}_{n-1}$: promedio anterior
- x_n: variable medida
- n: índice de medición

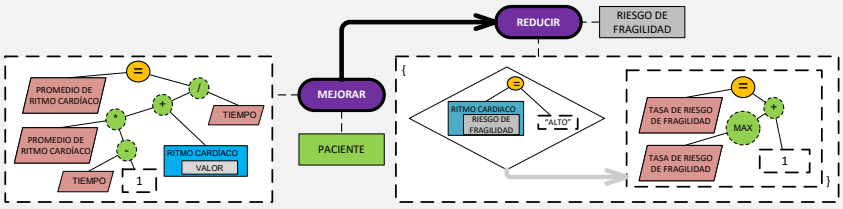
$$\text{PROMEDIO\_RITMOCARDIACO} = \frac{(\text{PROMEDIO\_RITMOCARDIACO} * (\text{TIEMPO} - 1)) + \text{VALOR}}{\text{TIEMPO}} \tag{5.2}$$

- PROMEDIO_RITMOCARDIACO: promedio actualizado del ritmo cardíaco
- $\text{PROMEDIO\_RITMOCARDIACO}_{n-1}$: promedio anterior
- VALOR: valor actual del ritmo cardíaco
- TIEMPO: índice discreto de medición (hora)

Resumen guía para especificar características télicas

Para especificar las características télicas de un sistema en los EP, se recomienda:

Definir la restricción/especificación de las relaciones de logro: Se deben incluir los indicadores de rendimiento clave mediante condiciones u operaciones matemáticas dentro de la restricción o especificación de la relación de logro. Por ejemplo, la especificación del objetivo “mejorar paciente” y la restricción del objetivo “riesgo de fragilidad”.



Características dinámicas: Cuando un “paciente llega” al sistema, se activa la relación eventual que evalúa si existe una “ocurrencia activa”, es decir, una condición que determina la ejecución de los procesos subsiguientes. Una vez verificada esta ocurrencia, se activa la relación dinámica “recepcionista ingresa paciente”, que representa la acción del actor “recepcionista” sobre el concepto “paciente”.

Esta relación desencadena dos acciones dinámicas principales que se agrupan dentro de un marco: la acción de “insertar” los datos del paciente y la de “seleccionar” un “reloj inteligente” disponible. En la acción de inserción se capturan los conceptos hoja asociados al paciente: “id”, “nombre”, “edad” y “celular”, los cuales son necesarios para su identificación y contacto. Posteriormente, mediante la acción de selección, la recepcionista asocia un “reloj inteligente” mediante su “referencia”.

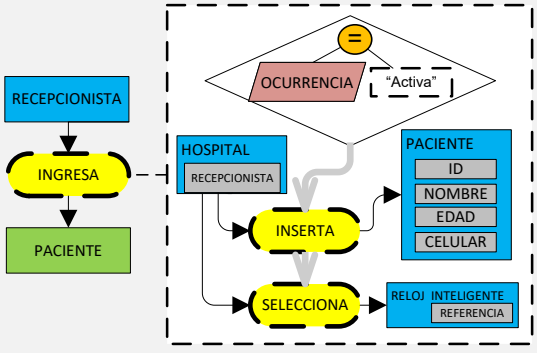
Por ejemplo, la “paciente” con id = 1027, nombre = Leticia Londoño, edad = 75 años y celular = 3156782399, es ingresado por el recepcionista del Hospital Pablo Tobón Uribe, como se describe en la Tabla 5.9. A este paciente se le asigna un “reloj inteligente” con referencia = M1, el cual permitirá capturar señales vitales como el ritmo cardíaco, de acuerdo con la información detallada en la Tabla 5.10.

Este ejemplo evidencia cómo los EP permiten representar detalladamente la lógica del sistema, articulando relaciones entre actores, conceptos principales y conceptos hoja, junto con las acciones asociadas que deben ejecutarse bajo ciertas condiciones.

Resumen guía para especificar características dinámicas

Para especificar las características dinámicas de un sistema en los EP, se recomienda:

Definir la especificación de las relaciones dinámicas: Se debe incluir la lógica interna que cumple el requisito funcional; el analista se puede guiar por los criterios de aceptación de una historia de usuario, donde se describe el paso a paso del requisito. Por ejemplo, la relación dinámica “recepcionista ingresa paciente” se relaciona con la restricción que indica que primero se debe “insertar paciente” y luego “seleccionar referencia del reloj inteligente” para asignarlo.



Características eventuales: El flujo que articula las relaciones dinámicas y eventuales comienza con el evento "paciente llega", que representa el inicio de la interacción con el sistema, tal como se muestra también en el EP general. Este evento se encuentra asociado con una especificación que describe su lógica interna: si el "tiempo" es mayor que 0 y menor o igual al "tiempo de simulación", entonces el sistema genera un valor "aleatorio". Dicho valor se compara con un "umbral", y si es mayor o igual a éste, la "ocurrencia" se marca como "activa", indicando que llegó un nuevo paciente. En caso contrario, el estado permanece como "inactivo" y el flujo no se ejecuta.

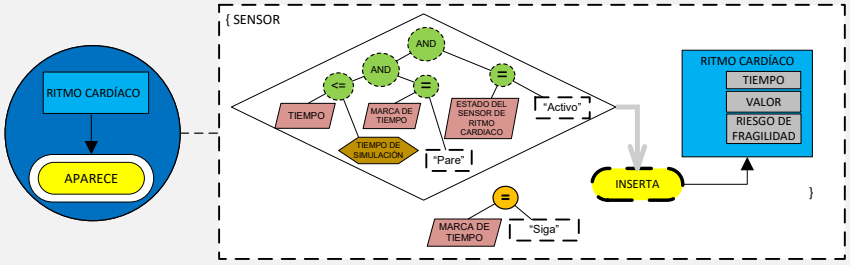
Posteriormente, el evento "tiempo pasa" controla el avance temporal dentro del modelo de simulación. Este evento también cuenta con una especificación interna que define su comportamiento cíclico: mientras el tiempo actual sea menor que el "tiempo de simulación" y la "marca de tiempo" esté en estado "siga", el "tiempo" incrementa en intervalos definidos (por ejemplo, una hora). Una vez actualizado, la marca de tiempo se cambia a "pare" para detener momentáneamente la ejecución, permitiendo que los eventos sensoriales y de monitoreo puedan capturar los datos correspondientes.

Uno de esos eventos es "valor del ritmo cardíaco aparece", que representa la recolección automática de señales vitales del paciente por medio del "sensor del reloj inteligente". Este evento incluye una especificación que permite insertar los datos del "ritmo cardíaco", como el tiempo de la medición, el valor obtenido en latidos por minuto (lpm), y el resultado del análisis del "riesgo de fragilidad". Con base en esta información, se puede generar automáticamente una *notificación* cuando se detecta un estado de riesgo alto, cerrando así el ciclo de retroalimentación en tiempo real dentro del sistema simulado.

Resumen guía para especificar características eventuales

Para especificar las características eventuales de un sistema en los EP, se recomienda:

Definir la restricción/especificación de los eventos: Se debe incluir la lógica interna que cumple el requisito funcional automático; el analista se puede guiar por los criterios de aceptación de una historia de usuario, donde se describe el paso a paso del requisito. Por ejemplo, el evento "ritmo cardíaco aparece" se relaciona con la restricción que indica que se debe cumplir una condición para insertar los datos del "ritmo cardíaco".



La representación mediante EP permite una comprensión del sistema al capturar las características estructurales (actores, conceptos y atributos del dominio), dinámicas (interacciones y acciones realizadas por los actores), télicas (propósitos, metas o logros esperados del sistema) y eventuales (comportamientos autónomos). Esta caracterización se puede desarrollar desde

una vista general—útil para la comunicación temprana entre interesados—y desde una vista detallada, indispensable para la posterior implementación y validación del sistema. En conjunto, ambas vistas forman una base para avanzar hacia representaciones más técnicas como el diseño arquitectónico, asegurando la trazabilidad desde los requisitos hasta la ejecución técnica.

En caso de requerir una representación de la arquitectura para este caso de laboratorio, se puede considerar la propuesta de Rodríguez-Fortíz *et al.* (2019), la cual plantea una arquitectura basada en microservicios y orientada a eventos. Esta perspectiva complementa el EP al ofrecer una visión estructural del sistema que facilita su implementación distribuida y escalable.

5.3. Desarrollo: Equivalencia desde EP a código fuente

El proceso de traducción de un EP a código implica mapear los conceptos y relaciones hacia clases, atributos y métodos en el código fuente. Esta traducción se realiza con el objetivo de preservar la estructura lógica del sistema, asegurando que cada concepto, relación y restricción del modelo se representen de manera consistente en el código, por lo que se recomienda mantener la consistencia en el nombramiento de cada elemento. No obstante, el código resultante no se limita estrictamente a dicha traducción, ya que se pueden adicionar líneas de código según las necesidades específicas de implementación, siempre que estas extensiones no comprometan la coherencia con el EP.

A partir del EP de la Figura 5.3, se desarrolla la aplicación de riesgo de fragilidad utilizando el lenguaje de programación Python. Además, para la simulación, se utiliza la información de las Tablas 5.5, 5.6, 5.7, 5.8, 5.9, 5.10 y 5.11.

Enlace al código fuente:

5.3.1. Equivalencia de características estructurales

Los conceptos clase “Hospital”, “Usuario”, “Recepcionista”, “Paciente”, “Reloj Inteligente” y “Ritmo Cardíaco” se convierten en clases o archivos individuales en el código, siguiendo

una estructura modular y organizada, como se detalla en la Figura 5.4. Cada uno de estos conceptos se traduce en una clase que encapsula sus atributos y métodos, representando las características y comportamientos que se ven en el EP. Cada concepto se implementa como un archivo Python separado con el nombre correspondiente al concepto de la clase, por ejemplo: "Hospital.py" define la clase "Hospital".

Durante la traducción del EP a código, los conceptos hoja se representan como atributos dentro de las clases. Por ejemplo, en la clase "Paciente.py", sus atributos (*"id, nombre, celular, edad"*) corresponden directamente a los conceptos hoja del esquema, teniendo en cuenta que "id" y "nombre" vienen por la herencia de "usuario" (véase la Figura 5.3). Esta correspondencia estructural facilita la traducción a otras estructuras de datos, como bases de datos.

Figura 5.4. App Riesgo de Fragilidad

En una base de datos, el concepto se convierte en el nombre de la tabla y sus atributos en los campos de la misma. Siguiendo el ejemplo de la clase "Paciente", la tabla de datos contiene los mismos campos que se definen en el esquema. Esto asegura que la información sea consistente en todo el sistema. Por ejemplo, la Tabla 5.9 muestra los valores del primer paciente de una lista, que coinciden con la instancia de objeto p1 = Paciente(1027, "Leticia Londoño", 75, 3156782399, "M1", "890906347-9").

Las relaciones estructurales del EP se implementan en el código mediante diferentes tipos de relaciones entre clases (véase la vista estructural en la Figura 5.3). Estas relaciones se basan en la naturaleza del vínculo: la relación "tiene" se modela mediante la asociación ("paciente tiene ritmo cardíaco") y la composición o agregación ("hospital tiene paciente").

La relación "es" se implementa mediante la herencia. Esto crea una jerarquía lógica, donde las características comunes se definen en una clase padre (por ejemplo, "usuario"), mientras que las clases hijas ("paciente es usuario", "recepcionista es usuario") heredan esas propiedades

(véase la Figura 5.3). La herencia, además de establecer una jerarquía, también promueve el principio de responsabilidad única de los principios SOLID, al permitir que la lógica de las clases sea más específica y modular. Además, fomenta la reutilización del código, evitando la duplicación de funcionalidades comunes.

A continuación, se presenta el código de las clases "Hospital.py", "Recepcionista.py", "Paciente.py", "Reloj Inteligente.py" y "Ritmo Cardiaco.py", donde se puede observar la relación de herencia entre las clases "Usuario", "Recepcionista" y "Paciente".

Código 5.1. Hospital.py

```
class Hospital:
  def __init__ (self, nit, nombre, direccion):
        self.NIT = nit
        self.NOMBRE = nombre
        self.DIRECCION = direccion

  LISTA_HOSPITALES = []

h1 = Hospital(890901826-2, "Pablo Tobón Uribe", "Diagonal 80 76-08, Robledo")
h2 = Hospital(890906347-9, "Manuel Uribe Ángel", "Diagonal 31 36a sur 80,
     Envigado")
LISTA_HOSPITALES = [h1, h2]

def imprimirHospital():
  for hospital in LISTA_HOSPITALES:
    print ("Nit: ", hospital.NIT)
    print ("Nombre: ", hospital.NOMBRE)
    print ("Dirección: ", hospital.DIRECCION)
    print ("")
```

Código 5.2. Recepcionista.py

```
from Usuario import Usuario
class Recepcionista(Usuario):
    def __init__(self, id, nombre, nit):
        super().__init__(id, nombre)
        self.NIT_HOSPITAL = nit

LISTA_RECEPCIONISTAS = []

r1 = Recepcionista("RE1", "Natalia Beltrán", "890901826-2")
r2 = Recepcionista("RE2", "Luz Elena Villa", "890906347-9")
LISTA_RECEPCIONISTAS = [r1, r2]

def imprimirRecepcionista():
  for recepcionista in LISTA_RECEPCIONISTAS:
    print ("ID: ", recepcionista.ID)
    print ("Nombre: ", recepcionista.NOMBRE)
    print ("Nit Hospital: ", recepcionista.NIT_HOSPITAL)
    print ("")
```

Código 5.3. Paciente.py

```
from Usuario import Usuario

class Paciente(Usuario):
 def __init__(self, id, nombre, edad, celular, referencia, nit):
        super().__init__(id, nombre)
        self.EDAD = edad
        self.CELULAR = celular
        self.REFERENCIA_RELOJINTELIGENTE = referencia
        self.NIT_HOSPITAL = nit

LISTA_PACIENTES = []

p1 = Paciente(1027, "Leticia Londoño", 75, 3102004534,"M1","890901826-2")
p2 = Paciente(1028, "Ottoniel Noreña", 68, 3156782399,"M1","890906347-9")
LISTA_PACIENTES = [p1, p2]

def imprimirPaciente():
    for paciente in LISTA_PACIENTES:
      print ("ID: ", paciente.ID)
      print ("Nombre: ", paciente.NOMBRE)
      print ("Edad: ", paciente.EDAD)
      print ("Celular: ", paciente.CELULAR)
      print ("RelojInteligente: ", paciente.REFERENCIA_RELOJINTELIGENTE)
      print ("Nit: ", paciente.NIT_HOSPITAL)
      print ("")
```

Código 5.4. Usuario.py

```
class Usuario:
    def __init__(self, id, nombre):
        self.ID = id
        self.NOMBRE = nombre
```

Código 5.5. RelojInteligente.py

```
class RelojInteligente:
  def __init__ (self, codigo, referencia, marca, ip):
        self.CODIGO = codigo
        self.REFERENCIA= referencia
        self.MARCA = marca
        self.IP=ip

LISTA_RELOJES = []

re1 = RelojInteligente("M1", "GEAR S3", "Samsung", "216.58.152.97")
re2 = RelojInteligente("M2", "GEAR S5", "Samsung", "216.58.152.98")
LISTA_RELOJES = [re1, re2]

def imprimirRelojInteligente():
  for reloj in LISTA_RELOJES:
    print ("Código: ", reloj.CODIGO)
    print ("Referencia: ", reloj.REFERENCIA)
```

```
    print ("Marca: ", reloj.MARCA)
    print ("IP: ", reloj.MARCA)
    print ("")
```

Código 5.6. RitmoCardiaco.py

```
import random

class RitmoCardiaco:
  def __init__ (self, tiempo,valor,riesgofragilidad):
    self.TIEMPOR= tiempo
    self.VALOR = valor
    self.RIESGOFRAGILIDAD =riesgofragilidad
```

En la clase "NotificacionRiesgoFragilidad.py" se implementa el *marco* de la vista estructural "Notificación de riesgo de fragilidad"del EP (véase la Figura 5.3). Esta clase alberga la lógica para las *condiciones iniciales*, cuyo código se presenta a continuación:

Código 5.7. NotificacionRiesgoFragilidad.py

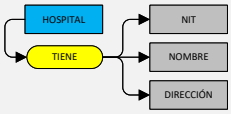
```
import random
import time
import matplotlib.pyplot as plt

from Hospital import imprimirHospital
from Paciente import imprimirPaciente
from Recepcionista import imprimirRecepcionista
from RelojInteligente import imprimirRelojInteligente

# Condiciones iniciales y parámetros globales
TIEMPO_SIMULACION = 72  # Horas
UMBRAL = 0.7  # Probabilidad en el rango [0 - 1]
TIEMPO = 0  # Horas
MARCA_TIEMPO = "Siga"  # Puede ser "Siga" o "Pare"
ESTADOSENSOR_RITMOCARDIACO = "Activo"  # Activo o Inactivo
PROMEDIO_RITMOCARDIACO = 0
TASA_RIESGOFRAGILIDAD = 0
```

Resumen guía para desarrollar características estructurales

Para codificar las características estructurales de un sistema, se recomienda seguir los siguientes pasos:

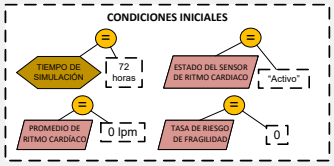
(1) **Crear las clases:** Los diferentes *conceptos clase* pueden pasar como clases o entidades a un lenguaje de programación. Por ejemplo, el concepto clase "Hospital" pasa a ser una clase en lenguaje de Python.

(2) **Definir los atributos:** Los *conceptos hoja* se definen como atributos dentro de una clase.




```
class Hospital:
def init (self, nit, nombre, direccion):
   self.NIT = nit
   self.NOMBRE = nombre
   self.DIRECCION = direccion
```

(3) **Definir variables:** Se incluyen condiciones iniciales como variables y parámetros en las clases del código. Por ejemplo las variables y parámetros del caso de laboratorio se usan en la clase principal "NotificacionRiesgoFragilidad.py"




```
# Condiciones iniciales
TIEMPO SIMULACION = 72 # Horas
ESTADOSENSOR RITMOCARDIACO = "Activo"
TASA RIESGOFRAGILIDAD = 0
```

(4) **Relacionar datos:** Para relacionar datos de los conceptos hoja se pueden utilizar desde estructuras de datos o bases de datos en el lenguaje desde las *tablas* con los posibles valores del sistema. Por ejemplo, la Tabla 5.8 que relaciona los datos del concepto principal "reloj inteligente" con sus conceptos hoja, los cuales en el caso de laboratorio se presentan en una lista (estructura de datos).

5.3.2. Equivalencia de características télicas

La traducción de las relaciones de logro "mejorar paciente" y "reducir riesgo de fragilidad" se expresa en el EP (Figura 5.3) mediante restricciones, las cuales se implementan como condiciones en el código. Dichas condiciones forman parte de la lógica de la clase "NotificacionRiesgoFragilidad.py", encargada de gestionar la notificación del riesgo de fragilidad. Según el valor obtenido, el sistema actualiza los indicadores de desempeño en las variables promedio de ritmo cardíaco y tasa de riesgo de fragilidad, tal como se muestra a continuación:

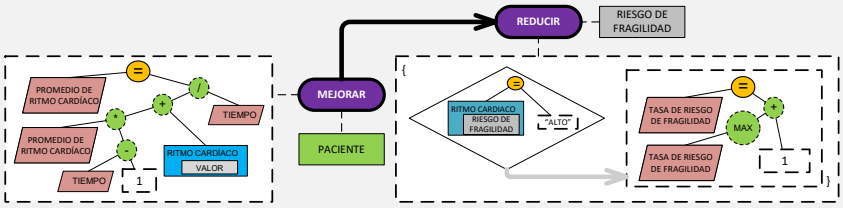
Código 5.8. Indicadores

```
# Actualizar KPI
    PROMEDIO_RITMOCARDIACO = ((PROMEDIO_RITMOCARDIACO * (TIEMPO - 1)) +
         valor) / TIEMPO

if riesgo == "Alto":
    TASA_RIESGOFRAGILIDAD += 1
```

Resumen guía para desarrollar características télicas

Para codificar las características télicas de un sistema, se recomienda:

Definir indicador: Se deben incluir los indicadores de rendimiento clave mediante condiciones u operaciones matemáticas desde la restricción/especificación de la relación de logro. Por ejemplo, la especificación del objetivo "mejorar paciente" y la restricción del objetivo "riesgo de fragilidad".




```
# Actualizar KPI
    PROMEDIO_RITMOCARDIACO = ((PROMEDIO_RITMOCARDIACO * (TIEMPO -
          1)) + valor) / TIEMPO

if riesgo == "Alto":
    TASA_RIESGOFRAGILIDAD += 1
```

5.3.3. Equivalencia de características dinámicas

La traducción de la relación dinámica "Recepcionista ingresa Paciente" desde el EP de la Figura 5.3 se traduce a código en la función "ingresarPaciente()" y su respectiva especificación en la lógica interna de la función. A pesar de que en el código se determinan datos iniciales de pacientes, al utilizar esta función una "Recepcionista" puede ingresar un nuevo "Paciente" desde la clase "Paciente.py" con el siguiente código:

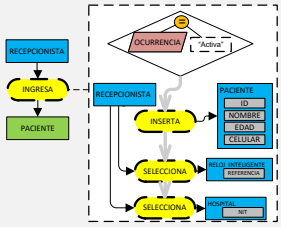
Código 5.9. ingresarPaciente()

```
def ingresarPaciente():
     for i in range (0,2):
       id = int(input("Ingrese el ID del paciente: "))
       nombre = input("Ingrese el nombre del paciente: ")
       edad = int(input("Ingrese la edad del paciente: "))
       celular = int(input("Ingrese el celular del paciente: "))
       referencia = input("Seleccione la referencia del reloj Inteligente: ")
       nit = int(input("Ingrese el nit del Hospital: "))
       print ("")
       paciente = PACIENTE(id, nombre, edad, celula, referencia, nit)
       LISTA_PACIENTES.append(paciente)
```

Resumen guía para desarrollar características dinámicas

Para codificar las características dinámicas de un sistema, se recomienda:

Crear funciones: A partir de las relaciones dinámicas se definen las funciones que ejecuta un usuario. La especificación permite traducir la lógica interna que satisface el requisito funcional. Para ello, el analista se puede apoyar en los criterios de aceptación de una historia de usuario, donde se detalla el paso a paso del requisito. Por ejemplo, la función "ingresarPaciente" se deriva de la relación dinámica "recepcionista ingresa paciente" de la Figura 5.3. Su implementación responde a la restricción que establece que, en primer lugar, se debe insertar el paciente, luego seleccionar la referencia del reloj inteligente y, finalmente, ingresar dicha referencia para asignarla correctamente. Es importante aclarar que se pueden adicionar las líneas adicionales que se requieran en el código.




```
def ingresarPaciente():
   for i in range (0,2):
      id = int(input("Ingrese el ID del paciente: "))
      nombre = input("Ingrese el nombre del paciente: ")
      edad = int(input("Ingrese la edad del paciente: "))
      celular = int(input("Ingrese el celular del paciente: "))
      referencia = input("Seleccione la referencia del reloj Inteligente: ")
      nit = int(input("Ingrese el nit del Hospital: "))
```

5.3.4. Equivalencia de características eventuales

La traducción de las relaciones eventuales "paciente llega", "tiempo pasa" y "ritmo cardíaco aparece" se realiza desde el EP representado en la Figura 5.3. En el código fuente, estas relaciones se implementan como funciones automáticas, ya que corresponden a procesos del sistema que no dependen de la interacción directa de un actor humano para ser ejecutados. En la especificación, se consideran requisitos funcionales de carácter automático, cuyo disparador está condicionado por eventos internos del sistema o por el estado de las variables globales, en lugar de por un rol explícito.

De esta manera, el código refleja la lógica del sistema al responder de forma autónoma a los cambios en el entorno simulado: la llegada de un paciente, el avance del tiempo y la detección de un valor de ritmo cardíaco por parte del sensor del reloj inteligente. Estas funciones permiten mantener la coherencia con el modelo en EP, garantizando que los procesos eventuales se gestionen como parte de un flujo continuo de ejecución.

En el siguiente fragmento de código se presenta la clase "NotificacionRiesgoFragilidad.py", dentro de la cual se encuentran implementadas las funciones correspondientes a estos eventos, asegurando la captura de datos, la actualización de variables y la notificación del riesgo de fragilidad cuando corresponde.

Código 5.10. NotificacionRiesgoFragilidad.py

```
def PACIENTE_LLEGA():
    """Determina si un paciente llega basándose en un valor aleatorio."""
```

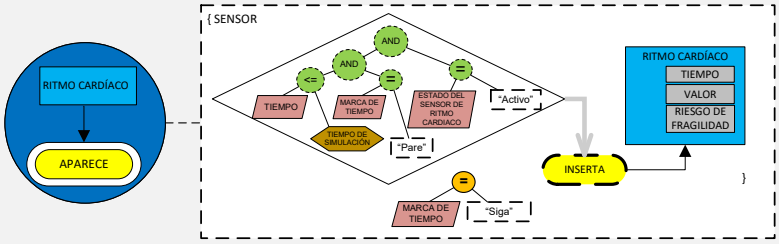
```
    global ALEATORIO, OCURRENCIA

    if 0 < TIEMPO <= TIEMPO_SIMULACION:
        ALEATORIO = random.random()
        OCURRENCIA = "Activa" if ALEATORIO >= UMBRAL else "Inactiva"
        print(f"[Evento PACIENTE_LLEGA] Ocurrencia: {OCURRENCIA}, Aleatorio: {
            ALEATORIO:.2f}")


def TIEMPO_PASA():
    """Controla el avance del tiempo dentro de la simulación."""
    global TIEMPO, MARCA_TIEMPO

    if TIEMPO <= TIEMPO_SIMULACION:
        if MARCA_TIEMPO == "Siga":
            TIEMPO += 1
            MARCA_TIEMPO = "Pare"
            print(f"[Evento TIEMPO_PASA] Tiempo actual: {TIEMPO} horas")
        else:
            print(f"[TIEMPO_PASA] Esperando marca de tiempo 'Siga'. Tiempo: {
                TIEMPO}")

registro_eventos = []

def RITMOCARDIACO_APARECE():
    """Genera y registra un valor de ritmo cardiaco."""
    global PROMEDIO_RITMOCARDIACO, TASA_RIESGOFRAGILIDAD, MARCA_TIEMPO

    if TIEMPO <= TIEMPO_SIMULACION and MARCA_TIEMPO == "Pare" and ESTADOSENSOR_
        RITMOCARDIACO == "Activo":
        valor = random.randint(50, 120)  # Ritmo aleatorio
        riesgo = "Alto" if valor > 100 else "Bajo"

        # Actualizar KPI
        if TIEMPO >= 1:  # Evita dividir por 0
            PROMEDIO_RITMOCARDIACO = ((PROMEDIO_RITMOCARDIACO * (TIEMPO - 1)) +
                valor) / TIEMPO

        if riesgo == "Alto":
            TASA_RIESGOFRAGILIDAD += 1

         # Registrar el evento
        evento = {
            'Tiempo': TIEMPO,
            'Ritmo Cardiaco': valor,
            'Riesgo': riesgo
        }
        registro_eventos.append(evento)

        # Registrar datos
        print(f"[Evento RITMOCARDIACO_APARECE] Tiempo: {TIEMPO} horas, Ritmo: {
            valor} lpm, Riesgo: {riesgo}")
        print(f"[Estado] Promedio ritmo cardiaco: {PROMEDIO_RITMOCARDIACO:.2f}
            lpm, Tasa riesgo fragilidad: {TASA_RIESGOFRAGILIDAD}")
```

```
        MARCA_TIEMPO = "Siga"
    else:
        print(f"[RITMOCARDIACO_APARECE] No se cumplen las condiciones para
             generar ritmo cardiaco.")


def main():
    """Controla la ejecución de la simulación."""
    print("Hospitales:")
    imprimirHospital()

    print("Recepcionistas:")
    imprimirRecepcionista()

    print("Relojes Inteligentes:")
    imprimirRelojInteligente()

    print("Pacientes:")
    imprimirPaciente()

    while TIEMPO <= TIEMPO_SIMULACION:
        PACIENTE_LLEGA()
        TIEMPO_PASA()
        RITMOCARDIACO_APARECE()
        print(f"Fin del ciclo para el tiempo {TIEMPO} horas"n")
        time.sleep(0.5)  # Pausa de medio segundo

    print(""n--- Resumen de Eventos ---")
    for evento in registro_eventos:
        print(evento)
    print("registro_eventos:", registro_eventos)

  # Simulación terminada, graficar resultados
    tiempos = [evento['Tiempo'] for evento in registro_eventos]
    ritmos = [evento['Ritmo Cardiaco'] for evento in registro_eventos]
    riesgos = [evento['Riesgo'] for evento in registro_eventos]

    # graficar la tasa de riesgo de fragilidad
    for tiempo, ritmo, riesgo in zip(tiempos, ritmos, riesgos):
        color = 'red' if riesgo == 'Alto' else 'green'
        plt.scatter(tiempo, ritmo, color=color, s=100, label=f"Riesgo: {riesgo}"
             if tiempo == tiempos[0] else "")
        plt.text(tiempo, ritmo + 2, riesgo, color=color, fontsize=9, ha='center')
               # Etiquetas "Alto" o "Bajo"

    plt.legend(loc='upper left')
    plt.legend()
    plt.show()

if __name__ == "__main__":
    main()
```

Resumen guía para desarrollar características eventuales

Para codificar las características eventuales de un sistema, se recomienda:

Crear funciones automáticas: Se deben definir las funciones automáticas que se traducen desde los eventos y su especificación. Se debe incluir la lógica interna de esta especificación, que cumple el requisito funcional automático; el analista se puede guiar por los criterios de aceptación de una historia de usuario, donde se describe el paso a paso del requisito. Por ejemplo, la función "RITMOCARDIACO_APARECE" desde el evento "ritmo cardíaco aparece" se relaciona con la restricción que indica que se debe cumplir una condición para insertar los datos del "ritmo cardíaco". Es importante aclarar que se pueden adicionar las líneas adicionales que se requieran en el código.




```
def RITMOCARDIACO_APARECE():
    """Genera y registra un valor de ritmo cardiaco."""
    global PROMEDIO_RITMOCARDIACO, TASA_RIESGOFRAGILIDAD, MARCA_TIEMPO

    if TIEMPO <= TIEMPO_SIMULACION and MARCA_TIEMPO == "Pare" and
         ESTADOSENSOR_RITMOCARDIACO == "Activo":
        valor = random.randint(50, 120)  # Ritmo aleatorio
        riesgo = "Alto" if valor > 100 else "Bajo"

         # Registrar el evento
        evento = {
            'Tiempo': TIEMPO,
            'Ritmo Cardiaco': valor,
            'Riesgo': riesgo
        }
        registro_eventos.append(evento)

         # Registrar datos
        print(f"[Evento RITMOCARDIACO_APARECE] Tiempo: {TIEMPO} horas,
             Ritmo: {valor} lpm, Riesgo: {riesgo}")
        print(f"[Estado] Promedio ritmo cardiaco: {PROMEDIO_RITMOCARDIACO
             :.2f} lpm, Tasa riesgo fragilidad: {TASA_RIESGOFRAGILIDAD}")
        MARCA_TIEMPO = "Siga"
    else:
        print(f"[RITMOCARDIACO_APARECE] No se cumplen las condiciones
             para generar ritmo cardiaco.")
```

5.4. Ejecución & evaluación

La implementación del código desde el EP de la Figura 5.3 permite generar consistencia desde el diseño del dominio. Los datos que se generan a partir de la simulación demuestran cómo los conceptos principales (*hospital*, *recepcionista*, *reloj inteligente* y *paciente*) interactúan en una aplicación guiada por esquemas preconceptuales (véanse las Figuras 5.2 y 5.3) evidenciando 72 horas de simulación. Esto permite, en el caso de laboratorio, monitorear eventos de riesgo de fragilidad en adultos mayores dentro del sector salud. La visualización de los resultados de la simulación se realiza por consola y se genera un gráfico de resultados en la Figura 5.5.

Para ejecutar el código fuente y simular los resultados puede ingresar al siguiente enlace:

Simulación por consola

Hospitales

```
Nit: 890901824
Nombre: Pablo Tobón Uribe
Dirección: Diagonal 80 N 76-08, Robledo

Nit: 890906338
Nombre: Manuel Uribe Ángel
Dirección: Diagonal 31 36a sur 80, Envigado
```

Recepcionistas

```
ID: RE1
Nombre: Natalia Beltrán
Nit Hospital: 890901826-2

ID: RE2
Nombre: Luz Elena Villa
Nit Hospital: 890906347-9
```

Relojes Inteligentes

```
Código: M1
Referencia: GEAR S3
Marca: Samsung
IP: Samsung
```

```
Código: M2
Referencia: GEAR S5
Marca: Samsung
IP: Samsung
```

Pacientes

```
ID: 1027
Nombre: Leticia Londoño
Edad: 75
Celular: 3102004534
RelojInteligente: M1
Nit: 890901826-2

ID: 1028
Nombre: Ottoniel Noreña
Edad: 68
Celular: 3156782399
RelojInteligente: M1
Nit: 890906347-9

[Evento TIEMPO_PASA] Tiempo actual: 1 horas
[Evento RITMOCARDIACO_APARECE] Tiempo: 1 horas, Ritmo: 92 lpm, Riesgo: Bajo
[Estado] Promedio ritmo cardiaco: 92.00 lpm, Tasa riesgo fragilidad: 0
Fin del ciclo para el tiempo 1 horas

[Evento PACIENTE_LLEGA] Ocurrencia: Inactiva, Aleatorio: 0.06
[Evento TIEMPO_PASA] Tiempo actual: 2 horas
[Evento RITMOCARDIACO_APARECE] Tiempo: 2 horas, Ritmo: 55 lpm, Riesgo: Bajo
[Estado] Promedio ritmo cardiaco: 73.50 lpm, Tasa riesgo fragilidad: 0
Fin del ciclo para el tiempo 2 horas

[Evento PACIENTE_LLEGA] Ocurrencia: Inactiva, Aleatorio: 0.48
[Evento TIEMPO_PASA] Tiempo actual: 3 horas
[Evento RITMOCARDIACO_APARECE] Tiempo: 3 horas, Ritmo: 92 lpm, Riesgo: Bajo
[Estado] Promedio ritmo cardiaco: 79.67 lpm, Tasa riesgo fragilidad: 0
Fin del ciclo para el tiempo 3 horas

[Evento PACIENTE_LLEGA] Ocurrencia: Activa, Aleatorio: 0.77
[Evento TIEMPO_PASA] Tiempo actual: 4 horas
[Evento RITMOCARDIACO_APARECE] Tiempo: 4 horas, Ritmo: 50 lpm, Riesgo: Bajo
[Estado] Promedio ritmo cardiaco: 72.25 lpm, Tasa riesgo fragilidad: 0
Fin del ciclo para el tiempo 4 horas

[Evento PACIENTE_LLEGA] Ocurrencia: Inactiva, Aleatorio: 0.01
[Evento TIEMPO_PASA] Tiempo actual: 5 horas
[Evento RITMOCARDIACO_APARECE] Tiempo: 5 horas, Ritmo: 69 lpm, Riesgo: Bajo
[Estado] Promedio ritmo cardiaco: 71.60 lpm, Tasa riesgo fragilidad: 0
Fin del ciclo para el tiempo 5 horas

[Evento PACIENTE_LLEGA] Ocurrencia: Inactiva, Aleatorio: 0.51
[Evento TIEMPO_PASA] Tiempo actual: 6 horas
[Evento RITMOCARDIACO_APARECE] Tiempo: 6 horas, Ritmo: 104 lpm, Riesgo: Alto
[Estado] Promedio ritmo cardiaco: 77.00 lpm, Tasa riesgo fragilidad: 1
```

```
Fin del ciclo para el tiempo 6 horas

[Evento PACIENTE_LLEGA] Ocurrencia: Inactiva, Aleatorio: 0.21
[Evento TIEMPO_PASA] Tiempo actual: 7 horas
[Evento RITMOCARDIACO_APARECE] Tiempo: 7 horas, Ritmo: 82 lpm, Riesgo: Bajo
[Estado] Promedio ritmo cardiaco: 77.71 lpm, Tasa riesgo fragilidad: 1
Fin del ciclo para el tiempo 7 horas

[Evento PACIENTE_LLEGA] Ocurrencia: Inactiva, Aleatorio: 0.57
[Evento TIEMPO_PASA] Tiempo actual: 8 horas
[Evento RITMOCARDIACO_APARECE] Tiempo: 8 horas, Ritmo: 99 lpm, Riesgo: Bajo
[Estado] Promedio ritmo cardiaco: 80.38 lpm, Tasa riesgo fragilidad: 1
Fin del ciclo para el tiempo 8 horas

[Evento PACIENTE_LLEGA] Ocurrencia: Inactiva, Aleatorio: 0.55
[Evento TIEMPO_PASA] Tiempo actual: 9 horas
[Evento RITMOCARDIACO_APARECE] Tiempo: 9 horas, Ritmo: 50 lpm, Riesgo: Bajo
[Estado] Promedio ritmo cardiaco: 77.00 lpm, Tasa riesgo fragilidad: 1
Fin del ciclo para el tiempo 9 horas

[Evento PACIENTE_LLEGA] Ocurrencia: Activa, Aleatorio: 0.74
[Evento TIEMPO_PASA] Tiempo actual: 10 horas
[Evento RITMOCARDIACO_APARECE] Tiempo: 10 horas, Ritmo: 63 lpm, Riesgo: Bajo
[Estado] Promedio ritmo cardiaco: 75.60 lpm, Tasa riesgo fragilidad: 1
Fin del ciclo para el tiempo 10 horas

[Evento PACIENTE_LLEGA] Ocurrencia: Activa, Aleatorio: 0.86
[Evento TIEMPO_PASA] Tiempo actual: 11 horas
[Evento RITMOCARDIACO_APARECE] Tiempo: 11 horas, Ritmo: 66 lpm, Riesgo: Bajo
[Estado] Promedio ritmo cardiaco: 74.73 lpm, Tasa riesgo fragilidad: 1
Fin del ciclo para el tiempo 11 horas

[Evento PACIENTE_LLEGA] Ocurrencia: Activa, Aleatorio: 0.86
[Evento TIEMPO_PASA] Tiempo actual: 12 horas
[Evento RITMOCARDIACO_APARECE] Tiempo: 12 horas, Ritmo: 118 lpm, Riesgo: Alto
[Estado] Promedio ritmo cardiaco: 78.33 lpm, Tasa riesgo fragilidad: 2
Fin del ciclo para el tiempo 12 horas
...

[Evento PACIENTE_LLEGA] Ocurrencia: Inactiva, Aleatorio: 0.14
[Evento TIEMPO_PASA] Tiempo actual: 70 horas
[Evento RITMOCARDIACO_APARECE] Tiempo: 70 horas, Ritmo: 63 lpm, Riesgo: Bajo
[Estado] Promedio ritmo cardiaco: 84.57 lpm, Tasa riesgo fragilidad: 21
Fin del ciclo para el tiempo 70 horas

[Evento PACIENTE_LLEGA] Ocurrencia: Inactiva, Aleatorio: 0.13
[Evento TIEMPO_PASA] Tiempo actual: 71 horas
[Evento RITMOCARDIACO_APARECE] Tiempo: 71 horas, Ritmo: 115 lpm, Riesgo: Alto
[Estado] Promedio ritmo cardiaco: 85.00 lpm, Tasa riesgo fragilidad: 22
Fin del ciclo para el tiempo 71 horas

[Evento PACIENTE_LLEGA] Ocurrencia: Inactiva, Aleatorio: 0.22
[Evento TIEMPO_PASA] Tiempo actual: 72 horas
[Evento RITMOCARDIACO_APARECE] Tiempo: 72 horas, Ritmo: 76 lpm, Riesgo: Bajo
```

```
[Estado] Promedio ritmo cardiaco: 84.88 lpm, Tasa riesgo fragilidad: 22
Fin del ciclo para el tiempo 72 horas

[Evento PACIENTE_LLEGA] Ocurrencia: Inactiva, Aleatorio: 0.59
[Evento TIEMPO_PASA] Tiempo actual: 73 horas
[RITMOCARDIACO_APARECE] No se cumplen las condiciones para generar ritmo cardiaco
Fin del ciclo para el tiempo 73 horas

--- Resumen de Eventos ---
{'Tiempo': 1, 'Ritmo Cardiaco': 92, 'Riesgo': 'Bajo'}
{'Tiempo': 2, 'Ritmo Cardiaco': 55, 'Riesgo': 'Bajo'}
{'Tiempo': 3, 'Ritmo Cardiaco': 92, 'Riesgo': 'Bajo'}
{'Tiempo': 4, 'Ritmo Cardiaco': 50, 'Riesgo': 'Bajo'}
{'Tiempo': 5, 'Ritmo Cardiaco': 69, 'Riesgo': 'Bajo'}
{'Tiempo': 6, 'Ritmo Cardiaco': 104, 'Riesgo': 'Alto'}
{'Tiempo': 7, 'Ritmo Cardiaco': 82, 'Riesgo': 'Bajo'}
{'Tiempo': 8, 'Ritmo Cardiaco': 99, 'Riesgo': 'Bajo'}
{'Tiempo': 9, 'Ritmo Cardiaco': 50, 'Riesgo': 'Bajo'}
{'Tiempo': 10, 'Ritmo Cardiaco': 63, 'Riesgo': 'Bajo'}
{'Tiempo': 11, 'Ritmo Cardiaco': 66, 'Riesgo': 'Bajo'}
{'Tiempo': 12, 'Ritmo Cardiaco': 118, 'Riesgo': 'Alto'}
{'Tiempo': 13, 'Ritmo Cardiaco': 75, 'Riesgo': 'Bajo'}
{'Tiempo': 14, 'Ritmo Cardiaco': 83, 'Riesgo': 'Bajo'}
{'Tiempo': 15, 'Ritmo Cardiaco': 92, 'Riesgo': 'Bajo'}
{'Tiempo': 16, 'Ritmo Cardiaco': 118, 'Riesgo': 'Alto'}
{'Tiempo': 17, 'Ritmo Cardiaco': 74, 'Riesgo': 'Bajo'}
{'Tiempo': 18, 'Ritmo Cardiaco': 116, 'Riesgo': 'Alto'}
{'Tiempo': 19, 'Ritmo Cardiaco': 103, 'Riesgo': 'Alto'}
{'Tiempo': 20, 'Ritmo Cardiaco': 108, 'Riesgo': 'Alto'}
{'Tiempo': 21, 'Ritmo Cardiaco': 99, 'Riesgo': 'Bajo'}
{'Tiempo': 22, 'Ritmo Cardiaco': 113, 'Riesgo': 'Alto'}
{'Tiempo': 23, 'Ritmo Cardiaco': 61, 'Riesgo': 'Bajo'}
{'Tiempo': 24, 'Ritmo Cardiaco': 107, 'Riesgo': 'Alto'}
{'Tiempo': 25, 'Ritmo Cardiaco': 108, 'Riesgo': 'Alto'}
{'Tiempo': 26, 'Ritmo Cardiaco': 56, 'Riesgo': 'Bajo'}
{'Tiempo': 27, 'Ritmo Cardiaco': 63, 'Riesgo': 'Bajo'}
{'Tiempo': 28, 'Ritmo Cardiaco': 69, 'Riesgo': 'Bajo'}
{'Tiempo': 29, 'Ritmo Cardiaco': 61, 'Riesgo': 'Bajo'}
{'Tiempo': 30, 'Ritmo Cardiaco': 57, 'Riesgo': 'Bajo'}
{'Tiempo': 31, 'Ritmo Cardiaco': 86, 'Riesgo': 'Bajo'}
{'Tiempo': 32, 'Ritmo Cardiaco': 99, 'Riesgo': 'Bajo'}
{'Tiempo': 33, 'Ritmo Cardiaco': 90, 'Riesgo': 'Bajo'}
{'Tiempo': 34, 'Ritmo Cardiaco': 91, 'Riesgo': 'Bajo'}
{'Tiempo': 35, 'Ritmo Cardiaco': 76, 'Riesgo': 'Bajo'}
{'Tiempo': 36, 'Ritmo Cardiaco': 103, 'Riesgo': 'Alto'}
{'Tiempo': 37, 'Ritmo Cardiaco': 95, 'Riesgo': 'Bajo'}
{'Tiempo': 38, 'Ritmo Cardiaco': 104, 'Riesgo': 'Alto'}
{'Tiempo': 39, 'Ritmo Cardiaco': 75, 'Riesgo': 'Bajo'}
{'Tiempo': 40, 'Ritmo Cardiaco': 64, 'Riesgo': 'Bajo'}
{'Tiempo': 41, 'Ritmo Cardiaco': 52, 'Riesgo': 'Bajo'}
{'Tiempo': 42, 'Ritmo Cardiaco': 102, 'Riesgo': 'Alto'}
{'Tiempo': 43, 'Ritmo Cardiaco': 52, 'Riesgo': 'Bajo'}
{'Tiempo': 44, 'Ritmo Cardiaco': 114, 'Riesgo': 'Alto'}
{'Tiempo': 45, 'Ritmo Cardiaco': 68, 'Riesgo': 'Bajo'}
```

```
{'Tiempo': 46, 'Ritmo Cardiaco': 66, 'Riesgo': 'Bajo'}
{'Tiempo': 47, 'Ritmo Cardiaco': 94, 'Riesgo': 'Bajo'}
{'Tiempo': 48, 'Ritmo Cardiaco': 86, 'Riesgo': 'Bajo'}
{'Tiempo': 49, 'Ritmo Cardiaco': 56, 'Riesgo': 'Bajo'}
{'Tiempo': 50, 'Ritmo Cardiaco': 95, 'Riesgo': 'Bajo'}
{'Tiempo': 51, 'Ritmo Cardiaco': 65, 'Riesgo': 'Bajo'}
{'Tiempo': 52, 'Ritmo Cardiaco': 64, 'Riesgo': 'Bajo'}
{'Tiempo': 53, 'Ritmo Cardiaco': 90, 'Riesgo': 'Bajo'}
{'Tiempo': 54, 'Ritmo Cardiaco': 112, 'Riesgo': 'Alto'}
{'Tiempo': 55, 'Ritmo Cardiaco': 100, 'Riesgo': 'Bajo'}
{'Tiempo': 56, 'Ritmo Cardiaco': 73, 'Riesgo': 'Bajo'}
{'Tiempo': 57, 'Ritmo Cardiaco': 96, 'Riesgo': 'Bajo'}
{'Tiempo': 58, 'Ritmo Cardiaco': 58, 'Riesgo': 'Bajo'}
{'Tiempo': 59, 'Ritmo Cardiaco': 109, 'Riesgo': 'Alto'}
{'Tiempo': 60, 'Ritmo Cardiaco': 116, 'Riesgo': 'Alto'}
{'Tiempo': 61, 'Ritmo Cardiaco': 53, 'Riesgo': 'Bajo'}
{'Tiempo': 62, 'Ritmo Cardiaco': 116, 'Riesgo': 'Alto'}
{'Tiempo': 63, 'Ritmo Cardiaco': 105, 'Riesgo': 'Alto'}
{'Tiempo': 64, 'Ritmo Cardiaco': 114, 'Riesgo': 'Alto'}
{'Tiempo': 65, 'Ritmo Cardiaco': 108, 'Riesgo': 'Alto'}
{'Tiempo': 66, 'Ritmo Cardiaco': 112, 'Riesgo': 'Alto'}
{'Tiempo': 67, 'Ritmo Cardiaco': 62, 'Riesgo': 'Bajo'}
{'Tiempo': 68, 'Ritmo Cardiaco': 83, 'Riesgo': 'Bajo'}
{'Tiempo': 69, 'Ritmo Cardiaco': 50, 'Riesgo': 'Bajo'}
{'Tiempo': 70, 'Ritmo Cardiaco': 63, 'Riesgo': 'Bajo'}
{'Tiempo': 71, 'Ritmo Cardiaco': 115, 'Riesgo': 'Alto'}
{'Tiempo': 72, 'Ritmo Cardiaco': 76, 'Riesgo': 'Bajo'}
```

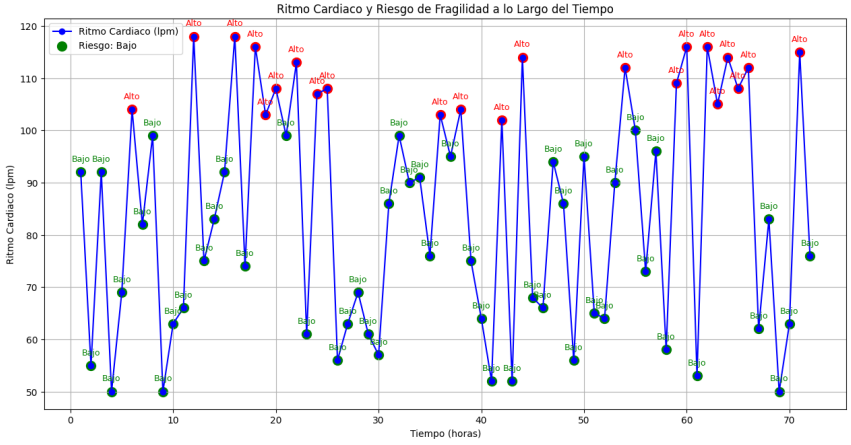


Figura 5.5. Ejecución

Capítulo 6

Conclusiones y desafíos futuros

6.1. Conclusiones

Hemos llegado al final de una exploración detallada de características estructurales, dinámicas, télicas y eventuales de los esquemas preconceptuales. En un momento en que analistas, desarrolladores y científicos requieren alternativas para abordar de manera más efectiva las etapas iniciales del desarrollo de software, los EP ofrecen una solución. Esperamos que este libro haya proporcionado una comprensión profunda de los EP y una hoja de ruta para su uso. Como hemos mostrado a lo largo de libro, sabemos que, una especificación definitiva y revisada de los EP constituye un avance significativo hacia la unificación de sus características estructurales, dinámicas, télicas y eventuales. Al integrarlas en una guía homogénea, se reduce la ambigüedad y se facilita su uso en el diseño de sistemas de software.

Los EP se consolidan como una herramienta clave en el proceso de educción de requisitos, permitiendo capturar y representar el conocimiento tácito y explícito de los interesados de manera clara y comprensible. Es por eso que, después de leer este libro, esperamos que se evidencie cómo la especificación mediante los EP proporciona insumos y lineamientos concretos que son valiosos en etapas tempranas del desarrollo de software, resultando útiles en el proceso de desarrollo de software.

Nuestro objetivo al escribir este libro era compilar los aportes más relevantes en términos de los EP para que resultados logrados se puedan reutilizar. Esperamos motivar el estudio e iniciativas de uso de los EP y respaldar la aplicación de los mismos. Por esto, realizamos un cierre del ciclo de especificación-implementación con la generación de código fuente a partir de los EP, ejemplificada en un caso de laboratorio, que demuestra la viabilidad de cerrar el ciclo desde la representación conceptual hasta la implementación técnica. Esto mejora la trazabilidad entre requisitos y código, fomenta una mayor eficiencia en el desarrollo de software. En complemento, la aplicación de los EP en un sistema de salud para evaluar el riesgo de fragilidad en adultos mayores valida su utilidad en un contexto real. Este caso demuestra que los EP se pueden emplear para abordar problemas complejos en dominios específicos, aprovechando arquitecturas modernas como los microservicios dirigidos por eventos.

Por último, ya que has llegado hasta aquí, cabe mencionar que la especificación revisada establece las bases para la automatización del proceso de educción de requisitos y generación de productos de trabajo en software. Esta automatización reduce el esfuerzo manual, asegura una mayor consistencia y calidad en los productos generados. También, al ofrecer una visión integral de las características de los EP en una única fuente, este libro minimiza las ambigüedades

previamente presentes debido al tratamiento aislado de dichas características. Esto mejora la comprensión y adopción de los EP en el diseño y desarrollo de sistemas de software.

En conclusión, la consolidación de las investigaciones previas y la propuesta de nuevas metodologías y herramientas para el uso de los EP representan un recurso valioso tanto para la academia como para la industria del software. Este libro se perfila como una referencia esencial para quienes deseen explorar, aplicar o perfeccionar los EP. Tambien, este trabajo contribuye a establecer una base sólida para la adopción, uso y mejora continua de los EP en el ámbito del diseño de software, promoviendo su estandarización y potenciando su impacto en proyectos futuros.

6.2. Desafíos futuros

Al abordar el tema de los EP en el contexto del diseño y desarrollo de software, dado que, como se mencionó a lo largo de este libro, se realizan en etapas tempranas del ciclo de vida y sirven como puente entre los requisitos y el diseño, los retos y desafíos actuales incluyen aspectos técnicos, metodológicos, culturales y organizacionales. En términos de diseño de software, tanto de alto nivel como detallado, la complejidad creciente de los sistemas, la velocidad de desarrollo requerida y la necesidad de mantener calidad y escalabilidad es imperativo y dichos aspectos deben ser considerados y tratados, lo que también compete al uso de los EP.

Al representar sistemas lo suficientemente complejos se puede correr el riesgo de omitir elementos, malinterpretar conceptos, que la representación no sea completa lo suficiente para abordar lo esencial del sistema, subespecificar la representación con los EP, malinterpretar las necesidades del cliente e incumplir estas especificaciones con productos de trabajo derivados. Para abordar estos retos se podría incorporar validaciones frecuentes con los interesados para garantizar la fidelidad de los EP respecto del dominio y se pueden establecer procesos de validación y verificación claros, incluyendo el uso de pruebas automáticas basadas en EP. También, en proyectos de larga duración los requisitos evolucionan constantemente, lo que obligaría a actualizar los EP para reflejar el estado actual del sistema. Esto obliga a implementar sistemas de gestión de cambios y versionamiento para los EP.

Como mencionamos, a medida que un sistema software crece en tamaño y complejidad, los EP se pueden volver extensos y difíciles de gestionar. Es una realidad desde la perspectiva de las representaciones gráficas y/o productos de trabajo de modelado y para esto se puede pensar en estandarizar los niveles de abstracción y descomponer los EP en módulos manejables. Este reto también se podría relacionar con la creciente adopción de los enfoques ágiles que priorizan la adaptabilidad, rapidez, entregas rápidas y evolución continua. Para ello, se puede pensar en adaptar los EP para que sean iterativos y evolucionen en paralelo con las prácticas ágiles. Introducir los EP en un equipo o proyecto que ya utiliza otras metodologías y herramientas puede generar resistencia al cambio, como en su momento sucedió y puede que aún suceda con los enfoques ágiles. Para ello, y de forma similar a como se presentó en este libro, se recomienda mostrar casos de éxito y beneficios concretos, destacando cómo los EP complementan y no reemplazan los enfoques existentes.

Por último y no menos importante, como presentamos en esta obra, los EP tienen oportunidad de seguir evolucionado, mejorando y adaptando a nuevos contexto, Por ejemplo, su uso en arquitecturas modernas como microservicios o soluciones de computación en la nube no está completamente explorado. Para esto, como autores de esta obra, altamente recomendamos fomentar la investigación aplicada para extender los EP y demostrar su valor en nuevos dominios y contextos tecnológicos. Algunos de ellos altamente especializados o técnicos como inteligencia artificial o biotecnología, sería interesante capturar y modelar el conocimiento tácito con EP y medir su pertinencia y/o dificultad, a la vez, desarrollar plantillas y guías específicas para diferentes dominios, facilitando la representación de conceptos complejos.

Referencias

Calle, J. M. (2016). Identificación de patrones de diseño para software científico a partir de esquemas preconceptuales. Master's thesis, Universidad Nacional de Colombia-Medellín Campus, Colombia.

Chaverra, J. J. (2011). *Generación automática de prototipos funcionales a partir de esquemas preconceptuales*. PhD thesis, Universidad Nacional de Colombia, Medellín Campus, Colombia.

Manjarrés, R. A. (2010). Generación de la especificación declarativa a partir de la especificación gráfica de un metamodelo. Master's thesis, Universidad Nacional de Colombia, Medellín Campus, Colombia.

Mosquera, J. D. (2021). Model-driven development of cross-platform mobile applications by using a set of heuristic rules based on pre-conceptual schemas. Master's thesis, Universidad Nacional de Colombia, Medellín Campus, Colombia.

Noreña-Cardona, P., Zapata, C., and Villamizar, A. (2019). Representing chemical events by using mathematical notation from pre-conceptual schemas. *IEEE Latin America Transactions*, 17(01):46–53.

Noreña-Cardona, P. A. (2014). Un mecanismo de consistencia en los eventos disparador y de resultado para los artefactos de unc-method. Master's thesis, Universidad Nacional de Colombia, Medellín Campus, Colombia.

Noreña-Cardona, P. A. (2020). *An Extension to Pre-conceptual Schemas for Refining Event Representation and Mathematical Notation*. PhD thesis, Universidad Nacional de Colombia, Medellín Campus, Colombia.

OMG (2003). *MDA (Model Driven Architecture) Guide Version 1.0.1*. OMG, Object Management Group. `http://www.omg.org/mda`.

OMG (2011). *Superestructure 2.4.1*. OMG, Object Management Group. `http://www.omg.org/spec/UML/2.4.1`.

Rodríguez-Fórtiz, M. J., García, F. M., Bermúdez, M., and Garrido, J. L. (2019). Una arquitectura orientada a microservicios y dirigida por eventos para el desarrollo de sistemas de esalud avanzados: Caso de evaluación de fragilidad en mayores. *Sistedes, biblioteca digital*, pages 1–8.

Vargas, F. A. (2016). *Modelo para la especificación de requisitos iniciales de software a partir de la relación sintáctica y semántica entre objetivos y problemas*. PhD thesis, Universidad Nacional de Colombia, Medellín Campus, Colombia.

Velásquez, S. (2019). Un modelo ejecutable para la simulación multi-física de procesos de recobro mejorado en yacimientos de petróleo basado en esquemas preconceptuales. Master's thesis, Universidad Nacional de Colombia, Medellín Campus, Colombia.

Villamizar, K. (2019). Definición de equivalencias entre historias de usuario y especificaciones en un-lencep para el desarrollo ágil de software. Master's thesis, Universidad Nacional de Colombia, Medellín Campus, Colombia.

Wieringa, R. (2014). *Design science methodology for information systems and software engineering*. Springer.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., Wesslén, A., et al. (2012). *Experimentation in software engineering*, volume 236. Springer.

Zapata, C. and Cardona, D. (2008). Implementación en c# de las reglas heurísticas de conversión de esquemas preconceptuales a diagramas uml 2.0. *Revista Facultad de Ingeniería Universidad de Antioquia*, (44):119–136.

Zapata, C. M. (2007). *Definición de un esquema preconceptual para la obtención automática de esquemas conceptuales de UML*. PhD thesis, Universidad Nacional de Colombia, Medellín Campus, Colombia.

Zapata, C. M. (2012). *The UNC-Method Revisited: Elements of the New Approach*. Saarbrücken: Lambert Academic Publishing.

Zapata, C. M. and Castro, L. F. (2017). Pre-conceptual-schema-based patterns for deriving key performance indicators from strategic objectives. *Ingeniería e Investigación*, 37(2):120–128.

Zapata, C. M. and Garcés, G. L. (2008). Generación del diagrama de secuencias de uml 2.1.1 desde esquemas preconceptuales. *Revista EIA*, 5(10):89–103.

Zapata, C. M., Giraldo, G. L., and Londoño, S. (2011a). Esquemas preconceptuales ejecutables. *Avances en sistemas e informática*, 8(1):15–24.

Zapata, C. M., Gómez, G. G., and Chaverra, J. J. (2011b). Generación automática de código bajo el patrón mvc a partir de esquemas preconceptuales. In *de XIX LACCEI Latin American and Caribbean Conference, Engineering for a Smart Planet, Innovation, Information, Colombia*.

Zapata, C. M., González, G., and Chaverra, J. J. (2011c). Generación automática del diagrama entidad-relación y su representación en sql desde un lenguaje controlado (un-lencep). *Revista Ingenierías Universidad de Medellín*, 10(18):127–136.

Zapata, C. M., Lezcano, L. A., and Tamayo, P. A. (2007a). Validación del método para la obtención automática del diagrama de objetivos desde esquemas preconceptuales. *Revista EIA*, (8):21–39.

Zapata, C. M., Lezcano, L. A., and Tamayo, P. A. (2011d). Preconceptual-schema-based representation of kaos goal diagram. In *2011 6th Colombian Computing Congress (CCC)*, pages 1–6. IEEE.

Zapata, C. M., Tamayo, P. A., and Arango, F. (2007b). Conversión de esquemas preconceptuales a diagrama de casos de uso empleando atom^3. *Dyna*, 74(153):237–251.

Zapata-Tamayo, J. S. (2019). Generación semiautomática de código pl/sql a partir de representaciones de eventos basadas en esquemas preconceptuales. Master's thesis, Universidad Nacional de Colombia, Medellín Campus, Colombia.

Zapata-Tamayo, J. S. and Zapata-Jaramillo, C. M. (2018). Pre-conceptual schemas: Ten years of lessons learned about software engineering teaching. *Developments in Business Simulation and Experiential Learning*, 45:250–257.

Apéndice A

Anexos. Otros usos de esquemas preconceptuales

Una trivia funciona como acertijo visual que tiene asociación de conceptos, deducción lógica y comprensión de relaciones. En las siguientes Figuras A.1, A.2, A.3, A.4, A.5 y A.6, se presentan ejemplos de cómo usar los EP en el diseño de trivias.

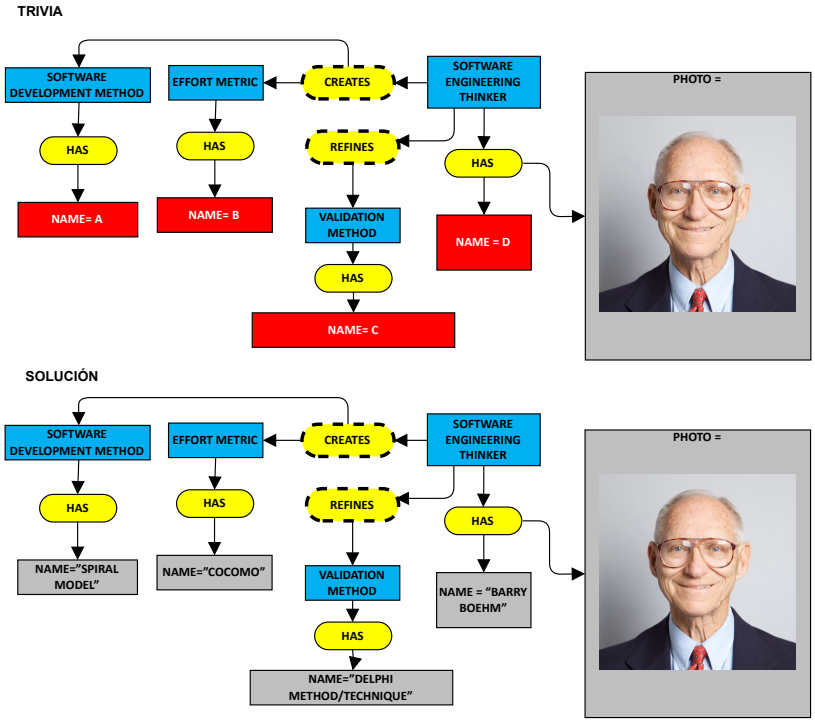


Figura A.1. Trivia 1

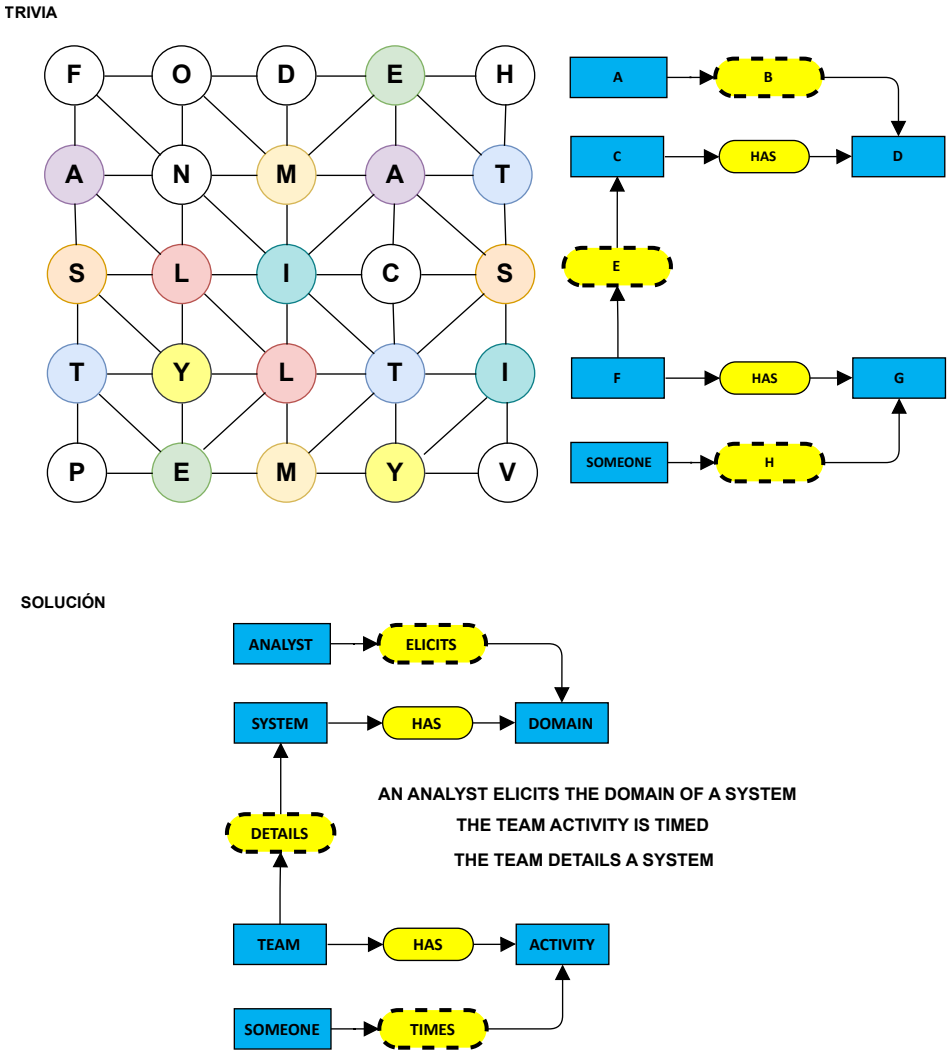
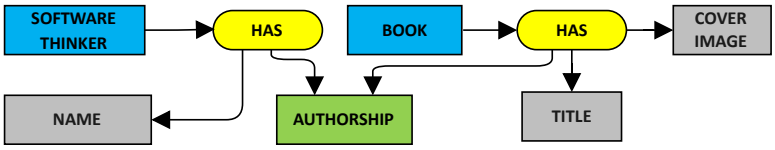


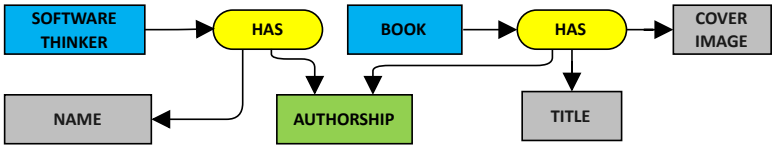
Figura A.2. Trivia 2

TRIVIA



AUTHORSHIP		
SOFTWARE THINKER.NAME	BOOK.TITLE	BOOK.COVER IMAGE
A	B	
C	D	

SOLUCIÓN



AUTHORSHIP		
SOFTWARE THINKER.NAME	BOOK.TITLE	BOOK.COVER IMAGE
ROBERT C. MARTÍN	CLEAN CODE	
FREDERICK BROOKS	THE MITICAL MAN-MONTH	

Figura A.3. Trivia 3

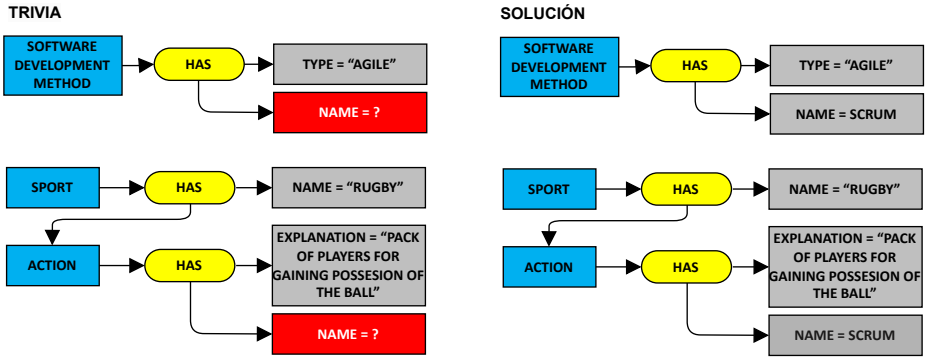


Figura A.4. Trivia 4

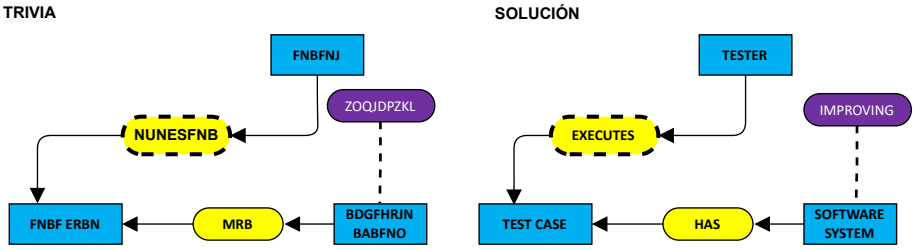


Figura A.5. Trivia 5

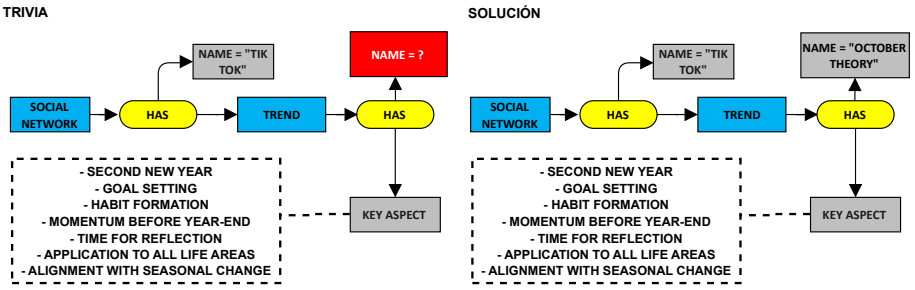
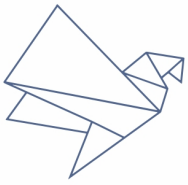


Figura A.6. Trivia 6

Índice alfabético



EDITORIAL

Manual para la creación de Esquemas Preconceptuales

Este manual ofrece una guía completa para la creación de esquemas preconceptuales (EP), un modelo intermedio entre el lenguaje natural y los esquemas conceptuales o lenguajes formales. Los EP permiten representar sistemas de software en etapas tempranas de análisis y diseño mediante una notación que integra características estructurales (elementos estáticos y sus relaciones), dinámicas (interacciones, cambios en el tiempo y comportamiento), télicas (objetivos y problemas) y eventuales (eventos que desencadenan comportamientos).

A lo largo del manual, se establece la equivalencia de los EP con esquemas conceptuales, artefactos ágiles y código fuente, lo que facilita su implementación en proyectos de software. Este libro es especialmente útil para ingenieros, diseñadores, desarrolladores y científicos, siendo una guía para modelar y comunicar sistemas de software en la academia y en la industria de manera sistémica, promoviendo así la comprensión, colaboración y generación de soluciones de software más cercanas a la necesidad del interesado.

9 786287 691780

Uniautónoma
DEL CAUCA