PA141-04
Framework para la verificación y corrección de modelos de procesos de negocio basado en técnicas de MDE y Ontologías

Oscar Ivan Vivas Reinoso

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
MAESTRÍA EN INGENIERÍA DE DE SISTEMAS Y COMPUTACIÓN
BOGOTÁ, D.C.
Año 2014

PA141-04
Framework para la verificación y corrección de modelos de procesos de negocio basado en técnicas de MDE y Ontologías

Autor:

Oscar Ivan Vivas Reinoso

MEMORIA DEL TRABAJO DE GRADO REALIZADO PARA CUMPLIR UNO
DE LOS REQUISITOS PARA OPTAR AL TITULO DE
MAGÍSTER EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

Director:
Ph.D. Jaime Pavlich Mariscal

Comité de Evaluación del Trabajo de Grado
Ph.D. Alexandra Pomares Quimbaya
Ph.D. Kelly Johany Garces Pernett

Página web del Trabajo de Grado
http://pegasus.javeriana.edu.co/~PA141-04-VerifModNegMDE

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
MAESTRÍA EN INGENIERÍA DE DE SISTEMAS Y COMPUTACIÓN
BOGOTÁ, D.C.
Noviembre, 2014

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
MAESTRÍA EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

Rector Magnífico
Jorge Humberto Peláez Piedrahita S.J.

Decano Académico Facultad de Ingeniería
Ingeniero Jorge Luis Sánchez Téllez

Director Maestría en Ingeniería de Sistemas y Computación
Ingeniero Enrique González Guerrero

Director Departamento de Ingeniería de Sistemas
Ingeniero Rafael Andrés González Rivera

Artículo 23 de la Resolución No. 1 de Junio de 1946

"La Universidad no se hace responsable de los conceptos emitidos por sus alumnos en sus proyectos de grado. Sólo velará porque no se publique nada contrario al dogma y la moral católica y porque no contengan ataques o polémicas puramente personales. Antes bien, que se vean en ellos el anhelo de buscar la verdad y la Justicia"

Dedicado a Dios y a mi madre que desde el cielo guían mis pasos

Esta tesis de grado la dedico en primera medida a Dios por haberme dado la oportunidad de estudiar en tan prestigiosa instutición.

A mi madre:

Quien siempre me apoyo y confio en mi, que desde el cielo guía mi camino y a quien dedico este esfuerzo con el cual doy este importante paso en mi vida.

Oscar Ivan Vivas Reinoso

La preocupación por el hombre y su destino siempre debe ser el interés primordial de todo esfuerzo técnico. Nunca olvides esto entre tus diagramas y ecuaciones.

Albert Einstein

Agradecimientos

Un agradecimiento especial a mi mamá Floralba Reinoso, que nunca dejó de apoyarme ni siquiera en sus últimos días y que de una u otra manera es la razón por la cual me veo en este punto de mi vida, a puertas del título de masgister tan anhelado.

A mi novia Mercedes Moreno, A mis hermanos Carlos Alberto Vivas Reinoso y Juan Pablo Vivas Reinoso con quienes siempre hemos sido muy unidos, y quienes me apoyaron y ayudaron en los momentos difíciles.

Finalmente, pero no menos importante, a mis profesores, que marcaron con sus enseñanzas el futuro de todos nosotros, especialmente para mi director Jaime Pavlich por todo el apoyo brindado, por su tiempo y por los conocimientos que me transmitió.

A todos con mucho cariño

Oscar Ivan Vivas Reinoso

Contenido

Agradecimientos vi

Resumen 2

Resumen Ejecutivo 3

1. INTRODUCCIÓN 5
1.1. Motivaciones . . . 5
1.2. Propuesta . . . 6
1.3. Estructura de la memoria . . . 7

2. ESTADO DEL ARTE 8
2.1. Business Process Management . . . 8
2.1.1. Business Process Modeling Notation . . . 9
2.2. Model Driven Architecture . . . 10
2.2.1. Domain-Specific Modeling . . . 11
2.2.2. Domain-Specific Language . . . 12
2.3. Ontologías . . . 13
2.3.1. Lenguajes . . . 13
2.3.2. Lógica descriptiva . . . 14
2.3.3. SWRL . . . 16
2.3.4. Razonamiento . . . 16
2.4. Trabajos Relacionados . . . 17

3. DISEÑO DE LA SOLUCIÓN 19
3.1. Alcance . . . 19
3.1.1. Objetivo General . . . 19
3.1.2. Objetivos Específicos . . . 19
3.2. Metodología . . . 20
3.3. Arquitectura . . . 21
3.3.1. Lenguaje de dominio específico - DSL . . . 23
3.3.2. Generador OWL . . . 26
3.3.3. Razonador . . . 29
3.3.4. Transformador de resultados . . . 30

4. VALIDACION 32
4.1. Caso de estudio . . . 32

4.2. Proceso de Verificación 34
4.3. Corrección de los modelos 35

5. CONCLUSIONES Y TRABAJOS FUTUROS 39
5.1. Conclusiones 39
5.2. Trabajos futuros 40

A. Anexo: Especificación del Diseño del Framework 41
A.1. Lenguaje de dominio específico 42
A.2. Generador OWL 46
A.3. Razonador 50
A.4. Especificación Transformador de resultados 51

B. Anexo: Plantillas DSL 54

Bibliografía 57

RESUMEN

El presente trabajo propone un framework para la validación de diagramas BPMN (Notación para el Modelado de Procesos de Negocio) con respecto a las reglas de negocio que los rigen. Este framework está compuesto por un DSL (Lenguaje de Dominio Específico) que es capaz de representar reglas de un proceso de negocio y transformar tanto el diagrama BPMN como las reglas del proceso en una base de conocimiento. Esta base de conocimiento es representada por una ontología, esto con la intención de validar la consistencia del diagrama respecto a sus reglas de negocio. La utilidad de este framework radica en la capacidad de realizar validaciones sobre los modelos BPMN antes de ser implementados en sistemas BPM y almacenar las restricciones de negocio en un formato estándar fácil de comprender y compartir. Los resultados obtenidos en las pruebas de este framework demuestran la utilidad de este enfoque en el mantenimiento e implantación de sistemas BPM.

Palabras clave: arquitectura dirigida por modelos, lenguaje de dominio específico, notación de gestión de procesos de negocio, ontología, motor de inferencia. gestión de procesos de negocio.

ABSTRACT

This paper proposes a framework for validating BPMN diagrams (Business Process Modeling Notation) with respect to the business rules that govern it. This framework consists of a DSL (Domain Specific Language) that is capable of representing rules of a business process and transform both the BPMN diagram and process rules in a knowledge base. This knowledge base is represented by an ontology in order to validate the consistency of the diagram with respect to its business rules. The usefulness of this framework is the ability to perform validations on BPMN models before its implementation in BPM systems and store business constraints in a standard format that is easy to understand and share. The results of testing this framework demonstrate the utility of this approach in the maintenance and deployment of BPM systems.

Keywords: Model Driven Architecture, Domain Specific Language, Business process management notation, Ontology, Inference engine, Business process management

RESUMEN EJECUTIVO

En la actualidad las compañías centran sus prioridades en tener sus procesos de negocio alineados con sus objetivos organizacionales, persiguiendo un continuo mejoramiento y evolución de los procesos de negocio para asegurar que generen un valor agregado que les garantice competitividad [1]. Estas son algunas de las razones por las que una empresa se puede enfrentar a realizar proyectos de implementación o mejoramiento de sus sistemas BPM.

Consecuentemente los analistas de negocio de dichas compañías deben realizar tareas de análisis y diseño de los diagramas BPMN [2] que representan los procesos tácticos, operativos o de apoyo en la organización, los cuales finalmente se implementan en un sistema BPM. El conocimiento relacionado a las restricciones de negocio, que deben seguir los diagramas modelados normalmente queda en manos de los analistas de negocio, que no poseen una forma estándar de representar estas reglas o restricciones de negocio. Adicionalmente la transmisión de conocimiento del comportamiento del proceso no se apoya en algún artefacto estándar [2].

Razón por la cual este trabajo propone desarrollar un framework que permita modelar las restricciones de negocio de un proceso, con la intención de que este modelo sirva para verificar la consistencia de los diagramas BPMN que se relacionen al proceso de negocio que se está analizando. Para lograr este cometido el framework utiliza técnicas de ingeniería dirigida por modelos (MDE) [3], para modelar las restricciones de negocio. La forma en que MDE propone que se debe realizar este proceso es a través de un lenguaje de dominio específico (DSL) [4] que se enfoque en modelar los elementos del dominio de un proceso de negocio. Esto implica desarrollar un lenguaje de dominio específico que permita representar las restricciones de un proceso de negocio. Adicionalmente el framework posee un componente que verifica si el diagrama BPMN cumple con las restricciones representadas en el DSL, esto se logra transformando tanto el diagrama BPMN como el modelo DSL en una base de conocimiento que permita realizar el proceso de verificación propuesto.

Para realizar este proceso de verificación se utilizaron técnicas basadas en ontologías [5] que permiten representar los modelos en una única notación y verificar que se cumplan las restricciones de negocio. Esto se logra transformando las restricciones de negocio y el diagrama BPMN en notación del Web Ontology Language (OWL) [6] y usando un motor de inferencia que realice consultas y razonamientos sobre esta ontología.

Finalmente los resultados entregados por el razonador son transformados en notación del

DSL la cual es más comprensible por el analista de negocio. De esta forma el analista podrá realizar cambios sobre el diagrama BPMN hasta obtener un modelo consistente. Para validar el funcionamiento de este framework se propuso un caso de estudio que posee varios escenarios mediante los cuales se explica cuál es el comportamiento del framework y los procesos de modelado y verificación que se pueden realizar.

Mediante el caso de estudio propuesto se demostró el funcionamiento del framework de verificación, donde se evidencia la capacidad de identificar inconsistencias en los diagramas BPMN antes de que se implementen en sistemas BPM, con base en un modelo que represente las restricciones de negocio de un proceso, lo cual constituye un ahorro en costos y tiempo, que se convierten en un ganancia potencial para las organizaciones. Adicionalmente este framework se complementa con el uso de plantillas de procesos de negocio estándar que pretenden ahorrar tiempo a la hora de realizar un modelo con el DSL propuesto.

Otra característica importante es que las restricciones de negocio quedan representadas en un modelo de fácil comprensión y que puede ser reutilizado para evolucionar los procesos de negocio constantemente, esto permite que el conocimiento no se pierda y pueda compartirse en una organización a todos los analistas de negocio y usuarios involucrados con el mantenimiento de los sistemas BPM.

1. INTRODUCCIÓN

Este primer apartado describe las razones que motivan el desarrollo de esta tesis, la propuesta de valor de este enfoque y finaliza con la estructura de la memoria.

1.1. Motivaciones

Una empresa exitosa se caracteriza por tener procesos de negocio estandarizados, que apoyen la visión estratégica de la compañía, consiguiendo de esta forma los objetivos planteados por la empresa; por esta razón son tan comunes en una organización los proyectos de implementación de BPMS (Business Process Management System o Sistemas de Gestión de Procesos de Negocio) que manejan los procesos vitales para el negocio [1]. El grado de éxito de estos proyectos está relacionado con la adecuada implementación de los modelos de negocio en el BPMS, como lo muestran los análisis realizadas por Gartner (líder mundial en la investigación de tecnologías de información y asesoría) [7].

No obstante cuando una compañía se enfrenta a la necesidad de implementar o mejorar un BPMS, realiza una representación gráfica de sus procesos de negocio en notación BPMN (Business Process Modeling Notation), siendo estos diagramas uno de los principales insumos para estos proyectos; estos normalmente resultan del trabajo colaborativo de los analistas de negocio [2], ya que los procesos de negocio son transversales en la organización y los analistas generalmente son especialistas en un departamento o área de la organización.

La representación de un proceso de negocio mediante un diagrama BPMN, se realiza a través de un conjunto de actividades lógicamente dependientes, ejecutadas para obtener un resultado de negocio, adicionalmente en dicha notación se describe el uso de datos, modelos de decisiones y se estipula que roles interactúan con dichas actividades [8].

Sin embargo las reglas del negocio que describen los objetivos de la compañía quedan implícitas en el sistema implementado, pero no se pueden observar en el diagrama BPMN y solo las conocen los analistas de negocio, esto dificulta la ejecución de proyectos que tengan como objetivo mejorar los procesos vitales en las empresas. No obstante aunque la gestión de procesos de negocio BPM [1] se enfoca en el mejoramiento de procesos en una compañía, no describe ningún artefacto mediante el cual se representen las reglas de negocio, estos son prácticamente idóneos en cada organización [2].

[1] Business Process Management

De aquí surge la importancia de mantener adecuadamente las reglas de negocio que rigen los procesos vitales en una organización y que deberían servir para validar la definición y mejoramiento de sus procesos.

1.2. Propuesta

El presente trabajo propone generar un lenguaje de modelado que represente las reglas de un proceso de negocio, conservando el conocimiento de los analistas en dicho modelo y que permita validar los diagramas BMPN que representan los procesos críticos en una compañía.

Para lograr esto se plantea utilizar las técnicas de MDE (Model Driven Engineering o Ingeniería Dirigida por modelos) [3], para desarrollar un DSL (Domain Specific Language o Lenguaje de Dominio específico) [4] que sea capaz de representar la reglas de un proceso de negocio y de esta forma asegurar que se conserve el conocimiento del proceso; en un lenguaje estándar que facilite su comprensión y trabajo colaborativo.

Dicha herramienta no sólo permitirá representar en un diagrama las reglas de negocio, sino que estas reglas servirán de insumo para validar los diagramas BPMN; esto garantizará la creación de diagramas consistentes y coherentes con los objetivos de la compañía. Para realizar esta validación se propone transformar el modelo BPMN y la reglas de negocio en una base de conocimiento, sobre la cual se verificará su consistencia.

Ya que las ontologías están compuestas por una serie de estándares y lenguajes que permiten representar el conocimiento y realizar procesos de inferencia no sólo para validar su consistencia sino para inferir nuevo conocimiento desde el ya existente [5], se plantea representar la base de conocimiento del diagrama BPMN y las reglas de negocio en una ontología.

Las ontologías representan el conocimiento a través de objetos, atributos, categorías de objetos y las relaciones entre ellos; permitiendo verificar la coherencia entre estos conceptos [5], característica mediante la cual se asegurará la consistencia de los diagramas con respecto a las reglas de negocio.

El DSL desarrollado tendrá entonces 2 funciones; la primera es permitir diagramar las reglas de negocio y la segunda es que mediante el uso de transformadores genere las ontologías que serán sometidas a una validación de consistencia.

Esta aproximación permitirá aprovechar toda la infraestructura desarrollada por la web semántica para gestionar el conocimiento [6]. Entre estos aspectos tenemos:

- La existencia de lenguajes estándar y repositorios que facilita el modelado de domi-

nios y posibilita usar otros dominios similares.

- Los razonadores existentes que permiten detectar y corregir inconsistencias en la definición de diagramas BPMN y reglas de negocio. Adicionalmente los razonadores que existen en la actualidad son capaces de gestionar grandes volúmenes de información de manera eficiente.

El uso de este enfoque no solo mejorará la capacidad de desarrollar diagramas BPMN que apoyen consistentemente los objetivos de la organización, sino que adicionalmente tiene todos los beneficios de las tecnologías involucradas:

- Facilidad para modelar y comprender el conocimiento del negocio de la compañía.
- Facilidad para centralizar y compartir el conocimiento del modelo de negocio.
- Capacidad de explicar las inconsistencias encontradas en los diagramas.

Con esta propuesta se espera generar un framework que apoye los proyectos de implementación y mejoramiento de sistemas BPM, mediante el cual se identifiquen inconsistencias en los diagramas de BPMN antes de su implementación y que aseguren conservar el conocimiento o reglas de negocio relacionadas a los diagramas BPMN.

1.3. Estructura de la memoria

El capítulo 2 describe el marco teórico sobre el cual se basa la investigación, el cual incluye las tecnologías y herramientas relacionadas a los conceptos de BPM, MDE y ontologías. El capítulo 3 apoyado en los conceptos del marco teórico ilustra en detalle el diseño y arquitectura del framework propuesto en este trabajo. A continuación el capítulo 4 presenta los casos de estudio y analiza los resultados del proceso de validación. Finalmente el capítulo 5 realiza las conclusiones del proyecto y propone una visión de los trabajos futuros que se pueden desprender de esta investigación.

2. ESTADO DEL ARTE

Este capítulo explica las tecnologías y herramientas que serán usadas a lo largo de la memoria para implementar el framework de validación. La primera sección 2.1 describe los conceptos usados por las organizaciones para realizar gestión de procesos de negocio y la notación mediante la cual se describen estos procesos [9] en un sistema BPM [10], los diagramas realizados en esta notación comprenden el objetivo a ser validado por el framework. La sección 2.2 describe los conceptos de la ingeniería dirigida por modelos [3] necesarios para diseñar un lenguaje [11] que describirá restricciones de negocio, el cual es otro componente del framework de validación propuesto. La sección 2.3 se centra en la forma de representar el conocimiento mediante ontologías [5] y como pueden ser usadas para realizar razonamientos, método que es utilizado por el framework para validar los modelos que representan procesos de negocio. Finalmente la sección 2.4 resume los trabajos mas relevantes relacionados a los conceptos descritos en el marco teórico.

2.1. Business Process Management

Business Process Management (BPM)[10] tiene sus orígenes en los 90s donde una nueva forma de organizar las empresas basada en los procesos de negocio fue propuesta. En el libro seminal "Reengineering the Corporation", Michel Hammer y James Champy defienden la idea de realizar diseños radicales de los procesos de negocio de la empresa. Ellos definen un proceso de negocio como una colección de actividades que tienen uno o más tipos de entrada y que genera una salida de valor para el cliente [10].

En la actualidad una de las definiciones más aceptadas describe a BPM como un conjunto de métodos y tecnologías utilizados para diseñar, representar y controlar procesos de negocio, en el que colaboran personas de negocio y tecnólogos para fomentar procesos de negocio efectivos y ágiles. BPM se centra principalmente en los procesos combinando tecnologías con metodologías de proceso y gobierno [12].

La tecnología BPM incluye herramientas para diseñar, representar y controlar los procesos de negocio operacionales, el diseño y modelado de procesos permite que se definan procesos que coordinan roles y comportamientos de personas, actividades y recursos. En estos modelos las personas se denominan actores, las actividades representan subprocesos y los recursos determinan datos que fluyen entre tareas [13].

2.1.1. Business Process Modeling Notation

Business Process Modeling Notation (BPMN) fue desarrollado bajo la coordinación del Object Management Group[9], el principal objetivo de BPMN es proveer una notación que sea realmente comprensible para todos los usuarios del negocio, como lo son: los analistas de negocio que crean los borradores iniciales de los procesos, los desarrolladores responsables de la implementación tecnológica y las personas de negocio que gestionan y monitorean esos procesos [12].

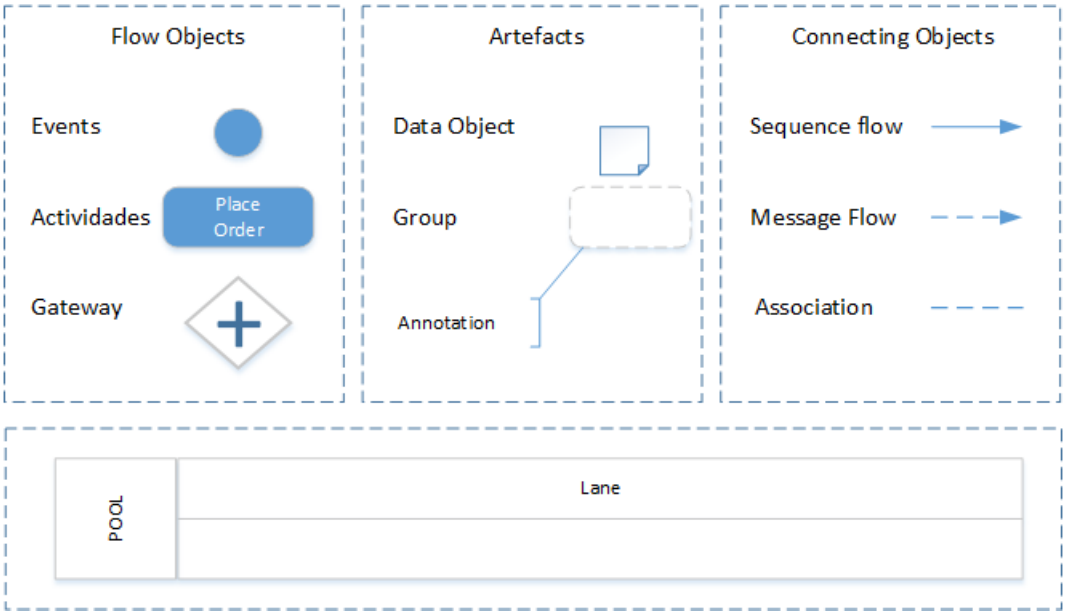


Figura 2-1.: Elementos BPMN (Adaptado de:[12])

Los elementos de la notación en los diagramas de procesos de negocio están divididos en cuatro categorías básicas como se ilustra en la figura **2-1**, cada una de las cuales tiene una serie de elementos:

- Los objetos de flujo (Flow Objects) construyen los bloques de los procesos de negocio, ellos incluyen eventos actividades y compuertas.
- Los aspectos organizacionales son representados por carriles de nado (swimlanes), que son elementos visuales utilizados para dividir responsabilidades de diferentes subprocesos. Estos tienen 2 niveles de jerarquía: piscinas (Pool) y carriles (Lanes).
- Los artefactos (artefacts) son utilizados para mostrar información adicional de los procesos aunque no son relevantes para la secuencia de flujo o los mensajes del proceso. Artefactos como los objetos de datos (Data Objects) son representados simplemente por un nombre y su propósito es documentar el proceso.
- Los objetos de conexión (Connecting Objects) asocian secuencialmente los elementos que componen un proceso.

2.2. Model Driven Architecture

La arquitectura dirigida por modelos es la respuesta de la ingeniería de software a la industrialización del desarrollo de software ya que mejora la productividad en el desarrollo de aplicaciones. Esta aproximación definida por Object Management Group (OMG) [14] en el 2001, propone realizar el desarrollo de sistemas dirigidos por modelos separando la especificacion de los sistemas de los detalles de su implementación en una plataforma específica [3].

Este estándar para el desarrollo de software utiliza los modelos de alto nivel de abstracción en las etapas de implementación, integración, mantenimiento y prueba del sistema; con la finalidad de que el esfuerzo de desarrollo y mantenimiento del software esté orientado hacia el trabajo a nivel de modelos y no de código fuente. MDA se basa en la construcción de tres modelos diferentes [15]:

A **Modelo Independiente de la computación - Computation Independent Model (CIM)**. Se enfoca en el ambiente del sistema en el cual los detalles de la estructura y procesamiento están ocultos e indeterminados. Normalmente es llamado un modelo de dominio y un vocabulario familiar a los participantes del dominio en cuestión es usado en esta especificación.

B **Modelado lógico independiente de la plataforma - Platform-Independent Model (PIM)**. Está enfocado en la operación de un sistema mientras oculta los detalles necesarios de una plataforma particular, por lo cual muestra qué parte de la especificación no cambia de una plataforma a otra.

C **Modelo específico de la plataforma - Platform Specific Model (PSM)**. Nivel de implementación que combina la especificación del PIM con los detalles específicos de cómo usar el sistema en una plataforma concreta.

Para pasar de un modelo a otro, MDA utiliza un proceso de transformación que genera un nuevo modelo del mismo sistema a través de una especificación de la transformación como se puede observar en la figura **2-2**.

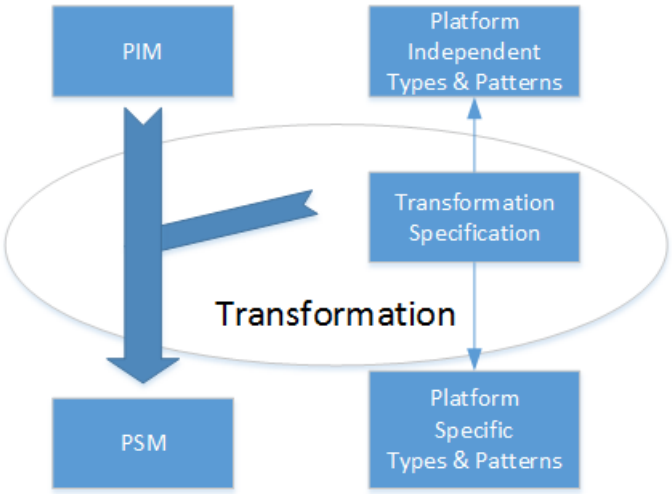


Figura 2-2.: Proceso de transformación (Adaptado de:[15])

2.2.1. Domain-Specific Modeling

Domain Specic Modeling (DSM) es una metodología para el diseño y desarrollo de software que envuelve el vocabulario del dominio de un tema particular, los pasos de esta metodología se ilustran en la figura **2-3**, el objetivo es que los desarrolladores solo creen modelos en el lenguaje resultado del DSM y las aplicaciones sean automáticamente generadas [4]. Para ello es necesario tener tres insumos:

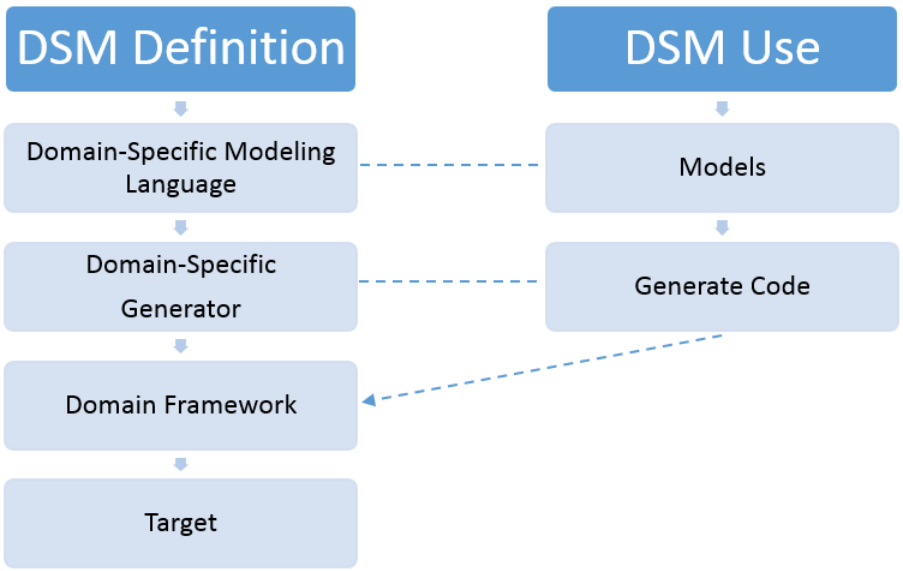


Figura 2-3.: Elementos de DSM y su uso (Adaptado de:[4])

- Un lenguaje de modelado de dominio específico. Contiene los elementos que representan los conceptos del dominio.

- Un generador de código de dominio específico. Programa mediante el cual se genera código ejecutable con base al modelo de dominio específico.

- Un framework de dominio. Soporta el código generado y provee la implementación común para las aplicaciones dentro del dominio.

Estos elementos son usados secuencialmente desde el modelo hasta generar el sistema propuesto.

2.2.2. Domain-Specific Language

Es un lenguaje estructurado dedicado a proveer una técnica para solucionar un problema en particular, mediante la representación de los conceptos de un tema específico. Al estar relacionado a un tema en concreto lo hace un lenguaje de limitada expresividad, el cual comprende cuatro puntos claves [11]:

- **Un lenguaje de programación informático**: un DSL es usado por humanos para decirle a un computador qué debe hacer. Como cualquier lenguaje de programación moderno, es estructurado y diseñado para hacer fácil de comprender por humanos.

- **Un lenguaje natural**: un DSL es un lenguaje estructurado que debería tener sentido de fluidez, donde la expresividad no sólo está compuesta por expresiones individuales sino por la forma en que están unidas.

- **Expresividad limitada**: Un lenguaje de propósito general provee la capacidad de soportar: variedad de datos, control y abstracción de estructuras. Todo esto es útil pero lo hacen difícil de aprender y usar. Un DSL soporta un mínimo de características necesarias para el dominio. No se puede construir el software completo de un sistema ya que el DSL cubre un aspecto particular del sistema.

- **Enfocado en el dominio**: un limitado lenguaje es útil sólo si este tiene un foco claro en un dominio específico. El foco en un dominio es lo que hace que un lenguaje limitado valga la pena.

Debido a que un DSL es una especialización de MDA sigue los mismos principios relacionados a modelar y generar código a partir de un modelo; como se puede observar en la arquitectura ilustrada en la figura **2-4**, donde el elemento "DSL Script" se refiere al DSL (que puede ser textual o gráfico) mediante el cual se representa los elementos de un dominio particular y que solo tienen significado gracias al modelo semántico (Semantic Model), este modelo semántico indica que elementos componen el DSL y como se relacionan entre ellos. Consecuentemente el modelo generado por el DSL puede ser usado para generar otro modelo o código objetivo.

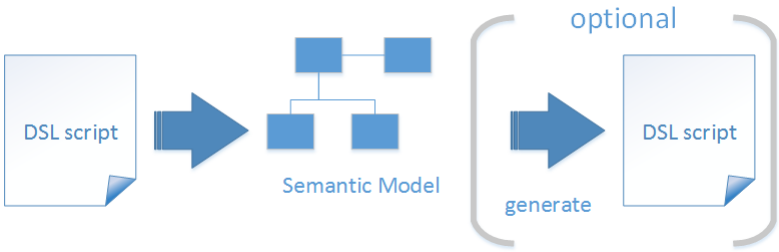


Figura 2-4.: Arquitectura de procesamiento de un DSL (Adaptado de: [11])

Para crear un DSL es necesario definir un metamodelo el cual permite describir los elementos del lenguaje. Consecuentemente para definir el metamodelo es necesario tener claridad de los siguientes aspectos del lenguaje:

- **Sintaxis abstracta**:Define los diferentes elementos del lenguaje y las reglas que establecen cómo pueden ser combinados.
- **Sintaxis concreta**: Define cómo los elementos del lenguaje aparecen en una notación visual o textual utilizable por las personas.
- **Semántica**: Define el significado del lenguaje.

2.3. Ontologías

La definición más difundida de ontología es la presentada por Gruber "una ontología es una especificación explícita de una conceptualización" [5]. Las ontologías son entonces las representación abstracta de la realidad de un dominio, para lo cual se basan en objetos, propiedades, las categorías o clases a las que pertenecen y las relaciones entre los objetos.

En cuanto al mundo de la informática; una ontología es una entidad computacional creada para el entendimiento común y compartido de un dominio que puede comunicarse entre expertos del dominio y sistemas computacionales [16]. Esto hace que una ontología se utilice como un mecanismo estándar para representar y compartir conocimiento [5].

Las ontologías posibilitan tanto representar conceptos como consultas sobre estos conceptos, un ejemplo de este uso es la web semántica que propone usar ontologías para representar el contenido de los sitios en internet y hacer búsquedas de contenido más exacto entre conceptos que no están explícitamente relacionados [17].

2.3.1. Lenguajes

La World Wide Web Consortium (W3C) ha propuesto un conjunto de lenguajes que permiten la representación de la información, las siguientes están relacionadas con este proyecto:

- **RDF (Resource Description Framework)**: es un lenguaje para representar información en la web de propósito general. Expresar información usando tripletas con la forma (sujeto, predicado, objeto). RDF permite definir modelos de datos en forma de grafo dirigido donde cada tripleta define un nodo origen (sujeto), una arista (predicado) y un nodo destino (objeto) [17].
- **SPARQL (es un acrónimo recursivo de SPARQL Protocol and RDF Query Language)**: es un lenguaje de consultas sobre grafos, el cual mediante patrones permite recuperar tripletas o subgrafos RDF [18].
- **OWL (Web Ontology Language)**: permite representar el significado de conceptos que componen un dominio particular, así como las relaciones entre dichos términos. Esta representación de términos y sus relaciones se denomina comúnmente ontología [19].

La especificación propuesta por la W3C describe tres variantes del lenguaje OWL:

- **OWL Lite**: Permite definir jerarquías y restricciones sencillas. Se limita la expresividad del lenguaje para mantener el desempeño, lo cual lo hace útil para representar tesauros y taxonomías.
- **OWL-DL**: Está pensado para los usuarios que requieren mayor expresividad, y necesitan asegurar completitud computacional en sus procesos de razonamiento. Todas las conclusiones pueden ser deducidas. Todos los cálculos se realizan en un tiempo finito.
- **OWL Full**: Es la variante con mayor expresividad aunque no garantiza la completitud computacional en los procesos de razonamiento. Permite definir entidades que sean a la vez clases, propiedades e instancias.

2.3.2. Lógica descriptiva

La lógica descriptiva (DL) es una familia de lenguajes que permiten representar el conocimiento de un dominio de forma estructurada con jerarquías conceptuales. Estos conceptos son usados para especificar propiedades de objetos e individuos que pertenecen al dominio. Otra característica de la lógica descriptiva es la capacidad de ofrecer servicios de razonamiento, mediante los cuales se infiere conocimiento a partir de la base de conocimiento representado en la lógica descriptiva [20].

Una base de conocimiento (KB) representada con lógica descriptiva está compuesta por dos componentes como se puede observar en la figura **2-5**. El primero comprende los Tbox, los cuales indican la terminología del dominio. La segunda parte son los Abox, que contienen las aserciones acerca de los individuos particulares del domino con base en el vocabulario expresado en la terminología. Es decir los ABox son afirmaciones realizadas sobre los individuos con base en los conceptos descritos por los TBox [20].

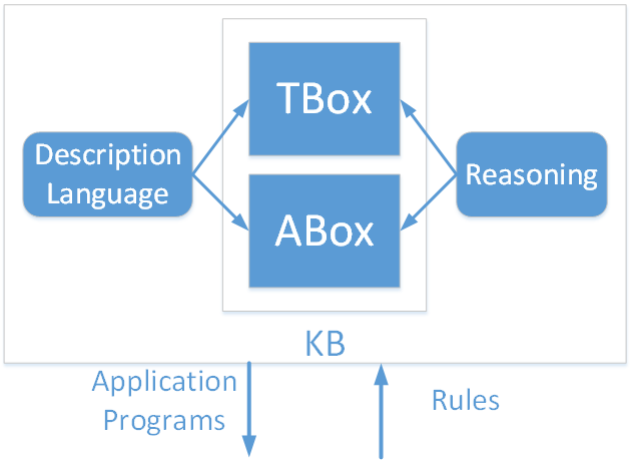


Figura 2-5.: Arquitectura una base de conocimiento (Adaptado de: [20])

Cuando se modela un dominio de conocimiento se construye una terminología T. Si se desea construir nuevos conceptos en base a T seguramente es necesario verificar si son consistentes o contradictorios respecto a T, propiedad conocida como **satisfacibilidad**. También, puede ser necesario saber si un concepto es más general que otro lo cual es llamado **subsunción** o si es **equivalente** o **disyunto** [20]. Formalmente estas propiedades se definen de la siguiente forma:

Si T es un Tbox, e I es una función de interpretación en un conjunto no vacío del domino, entonces podemos decir que:

- **Satisfactibilidad**: Un Concepto C es satisfactible respecto a T si existe un modelo I de T tal que C^I no es vacío. En este caso se dice también que I es un modelo de C.
- **Subsunción**: Un concepto C se subsume a otro concepto D con respecto a T si se cumple $C^I \subseteq D^I$ para todo modelo I de T. Este concepto se escribe con la siguiente notación $T \models C \subseteq D$.
- **Equivalencia**: Dos conceptos C y D son equivalentes respecto a T si $C^I = D^I$ para todo modelo I de T. Este concepto se escribe con la siguiente notación $T \models C \equiv D$.
- **Disyunción**: Dos conceptos C y D son disjuntos respecto a T si $C^I \cap D^I = 0$ para todo modelo I de T.

OWL-DL es una variante de la lógica descriptiva sobre la que se pueden realizar servicios de inferencia basados en las reglas descritas previamente. Estos servicios de razonamiento permiten determinar los siguiente:

- **Validación de Consistencia**: Validando que todos los conceptos expresados en la ontología sean validos con respecto a los TBox.

- **Clasificación de una base de conocimiento**: permite calcular todas las relaciones de subsunción entre conceptos para crear una jerarquía conceptual.
- **Reconocimiento de instancias**: un individuo a es una instancia del concepto C respecto a T si para todo modelo I de T se cumple $a^I \in C^I$.
- **Realización**: encontrar los conceptos atómicos más específicos de los que cada individuo es instancia.

2.3.3. SWRL

Semantic Web Rule Language (SWRL) se basa en OWL DL y OWL Lite y utiliza el subconjunto de reglas RuleML del Rule Markup Language. El propósito de SWRL es extender el conjunto de axiomas OWL al incluir cláusulas condicionales if-then o más formalmente llamadas implicaciones, que puedan ser combinadas con la base de conocimiento de OWL [21].

2.3.4. Razonamiento

Los razonadores permiten realizar operaciones de inferencia sobre las ontologías, como operaciones basadas en lógica descriptiva descritas en la sección 2.3.2, mediante las cuales se puede obtener nuevo conocimiento [20]. Existen diversos razonadores tanto comerciales, como de uso académico, de los cuales se realizó el siguiente análisis:

- **KAON2** [22]: Fue desarrollado por FZI Research Center y el AIFB Institute of the University of Karlsruhe, Alemania. Se puede usar para desarrollo de software excepto por estos dos módulos: OiModeler y KAON PORTAL, mediante los cuales se puede editar y mantener ontologías. soporta razonamiento con ontologías; RDF con relaciones simétricas, transitivas y relaciones inversas. [23].
- **HermiT** [24]: Este razonador maneja DL y principalmente soporta inferencia subsunción de jerarquica. HermiT implementa un novedoso algoritmo de razonamiento hypertableau, que es mucho menos determinista que los existentes algoritmos tableau. [25].
- **Pellet** [26]: este implementa un proceso de decision basado en tableau, emplea también optimizaciones para el razonador estándar DL [27].
- **Racer-Pro** [28]: Puede manejar varios TBoxes y varios ABoxes y trata a los individuos o instancias con el nombre único [16]. Además de tareas de razonamiento básicos, como satisfacibilidad y subsunción, ofrece consulta ABox basado en las optimizaciones nRQL. Se implementa en el lenguaje de programación Lisp Común [29].

La tabla **2-1** realiza un comparativo de los razonadores teniendo en cuenta los formatos de ontologías que soportan, las características de inferencia que poseen y adicionalmente los

mecanismos de interoperabilidad que posibilitan ser usados por otras aplicaciones. Características necesarias para ser usados en el proyecto propuesto. Dado este análisis se puede determinar que Pellet es el razonador más adecuado ya que permite interoperabilidad ejecutando instrucciones por linea de comando, realizar consultas SPARQL y chequeos de consistencia características que necesita el framewrok de validación propuesto.

Razonador	**Formato Soportado**	**Interoperabilidad**	**Chequeo de Consistencia**	**SPARQL**
KAON2	RDF, OWL-DL	API Java	Si	Si
Hermit	OWL-DL	API Java, Line Command	Si	No
Pellet	RDF, OWL-DL	API Java, Line Command	Si	Si
RacerPro	OWL-DL	API Java	No	No
Jena	OWL-DL	API Java	Si	Si

Tabla 2-1.: Comparativo de razonadores. (Adaptado de [23], [25], [27], [29])

2.4. Trabajos Relacionados

A continuación se describen los trabajos encontrados en el marco de este proyecto más relevantes a la evaluación de los procesos de negocio de un organización:

- **SUPER**: Uno de los casos más relevantes es SUPER (Semantics Utilised for Process management within and between Enterprises), para el cual el objetivo principal es elevar "la gestión de los procesos de negocio" en manos de los analistas de negocio donde debe estar, y sacarlo del nivel de TI donde reside ahora en la mayoría de los casos. Este proyecto fue desarrollado por la Comisión Europea, es coordinado por SAP Alemania [30] y es un importante ejemplo del uso de las ontologías para representar conocimiento. Sin embargo, está enfocado a arquitecturas empresariales y no se preocupa por validar los modelos con bajo nivel de abstracción.

- **Oracle BPM Ontology**: Oracle BPM Ontology está almacenada en "Oracle Database Semantic Technologies" y crea un modelo que permite representar diagramas BPMN, el cual está compuesto por una ontología que posee relaciones entre las Clases OWL esto con la finalidad de gestionar cómo se realizan los flujos de aprobación en un proceso de negocio [31].

- **"Toronto Virtual Enterprise Deductive Enterprise Model" (TOVE DEM)**: desarrollado en el laboratorio de integración de la universidad de Toronto [32], el cual realiza la definición de una terminología (dada en objetos, atributos y relaciones) y adicionalmente elabora un modelo DEM (Deductive Enterprise Model) que representa un conjunto de preguntas que el sistema debe ser capaz de responder, en el

cual los modelos se especifican en lógica de primer orden para ser implementados posteriormente en Prolog [32].

- **REA**: Se enfoca en la transferencia de valor económico. La ontología REA se basa en elementos del modelo REA (Recursos-Eventos-Actor) [33]. Los conceptos y definiciones de REA se aplican al espacio de colaboración entre empresas, en las que se produce intercambio de mercado entre socios comerciales. Aunque REA se considera una de las ontologías de dominio de negocios más prometedoras, es criticada por la falta de claridad e inconsistencia [34].

La tabla **2-2** ilustra cada uno de los trabajos relacionados, en la cual se puede observar como la representación de conocimiento y la capacidad de realizar razonamiento se ha utilizado en diferentes áreas y han sido campo de estudio por importantes empresas comerciales y entes educativos.

Proyecto	**Desarrollador**	**Objetivo**	**Lenguajes**
SUPER	Comisión Europea, SAP Alemania	Arquitectura Empresarial	OWL, WSML (Web Service Modeling Language)
Oracle BPM Ontology	ORACLE	Niveles de Aprobación	OWL, WSDL (Web Services Description Language)
REA	Propuesto por William E. McCarthy en 1982	Transferencia de Valor Económico	No usa un formato de ongología aceptado formalmente
TOVE	universidad de Toronto	Arquitectura Empresarial	FOL representation (First-Order Logic)

Tabla 2-2.: Comparativo de trabajos relacionados. (Adaptado de [30], [31], [32], [33])

3. DISEÑO DE LA SOLUCIÓN

Este capítulo explica cómo se realizó la implementación del framework de validación, razón por la cual está dividido en tres secciones: la sección 3.1 describe el alcance del proyecto, la sección 3.2 describe la metodología utilizada para implementar el framework y la sección 3.3 especifica la arquitectura y sus componentes.

3.1. Alcance

La finalidad del framework es validar los diagramas BPMN de un proceso de negocio con respecto a las restricciones que lo rigen, para lo cual representa y almacena estas restricciones y con base en estas realiza procesos de verificación. De acuerdo a este alcance se definen los siguientes objetivos para el framework de validación propuesto:

3.1.1. Objetivo General

Diseñar y construir un framework que permita verificar modelos de procesos de negocio y proponer correcciones mediante el uso de técnicas de MDE y razonamiento apoyado en ontologías.

3.1.2. Objetivos Específicos

- Construir un lenguaje de dominio específico que extienda de la notación de modelos de procesos de negocio (BPMN) para modelar restricciones y conceptos de negocio.
- Construir un componente que tome los modelos realizados mediante el lenguaje de dominio específico y los transforme en notación del lenguaje OWL.
- Integrar al framework un componente de razonamiento existente, que ofrezca la capacidad de verificar un modelo basado en ontologías.
- Construir un componente que permita traducir las respuestas entregadas por el componente de razonamiento a sus equivalentes del modelo de procesos de negocio original.
- Validar el funcionamiento del framework a través de un caso de estudio, en el cual se usen las funcionalidades de verificación para un modelo de procesos de negocio.

3.2. Metodología

El marco metodológico que se empleará en el desarrollo del presente trabajo está basado **parcialmente** en "La investigación científica basada en el diseño" [35], este marco está compuesto por 3 ciclos: rigor, relevancia y diseño, tal como lo ilustra la figura **3-1**. Estos ciclos utilizan el **entorno** que está compuesto por el dominio o contexto donde reside el problema y la **base de conocimiento** constituida por las teorías, modelos y métodos existentes con la finalidad de construir y evaluar una solución destinada a resolver un problema relevante [35].

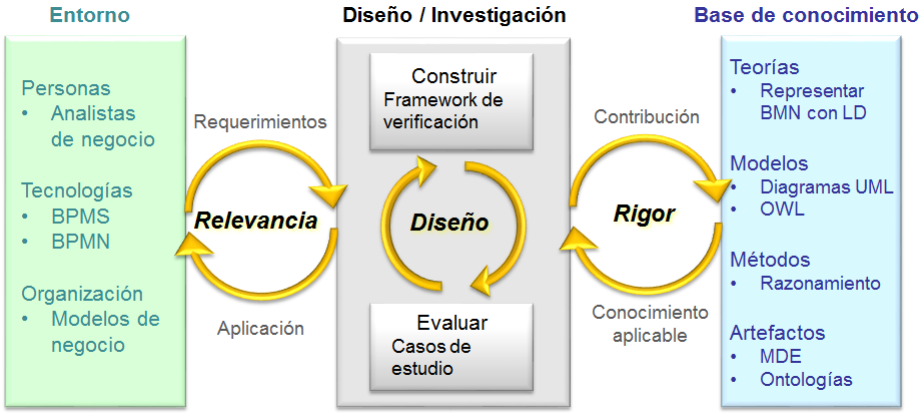


Figura 3-1.: Metodología investigación científica basada en el diseño (Adaptado de:[36])

Los ciclos de la ciencia basada en el diseño se ejecutan de forma paralela, es decir que mientras se determina el entorno en el ciclo de **relevancia** conjuntamente se define la base de conocimiento en el ciclo de **rigor**; con base en estos elementos se construye el diseño de la solución, el cual es refinado con los métodos usados para diseñar almacenados en la base de conocimiento [36].
Los ciclos y elementos del método aplicados al proyecto se detallan a continuación:

A **Relevancia**: Este ciclo se apoya en el entorno donde se desarrolla el problema, que para este contexto es donde se encuentran los analistas de negocio usando sistemas BPM que apoyan los procesos de negocio de una compañía, donde dichos procesos son representados mediante diagramas BPMN. A través de este ciclo se obtienen los requerimientos que resuelven la problemática encontrada en el entorno.

B **Rigor**: En este ciclo se investigan las teorías, métodos y modelos relevantes que ayudan a solucionar los requerimientos identificados. Para este proyecto se analizan las técnicas de modelado de conocimiento y procesos de negocio, adicionalmente se determina qué técnicas pueden ser usadas para validar procesos de negocio. Todo esto se realiza apoyándose en los trabajos de institutos y comunidades científicas [30] [31] [32] [33]. Estas teorías son insumos clave en el desarrollo de cada una de las fases del proyecto.

C Diseño: En este ciclo se construye y evalúa la solución de forma iterativa. Dicha solución comprende el framework de validación propuesto, basándose en las metodologías y artefactos de la base de conocimientos. La validación del diseño se realizará de forma teórica, a través de pruebas de aceptación definidas en base a los requerimientos. Este aspecto del proceso es el único que no se ciñe de manera estricta a la ciencia del diseño.

No obstante es relevante aclarar que en el ciclo del diseño para la construcción del framework se utilizó una metodología especial para construir lenguajes de dominio específico llamada Domain Specific Modeling DSM [4] que se explicó en detalle en la sección 2.2.1 del capítulo con el estado del arte. Esta metodología especifica que se deben ejecutar los siguientes pasos secuencialmente:

A Construir un lenguaje de modelado de dominio específico (DSL). Este lenguaje contendrá los elementos necesarios para representar restricciones de un proceso de negocio.

B Construir un generador de código de dominio específico. La finalidad del mismo es generar una base de conocimiento que contenga la información modelada con el DSL y el diagrama BPMN que se desea validar.

C Identificar el framework de dominio. Como el código generado en el paso anterior es una base de conocimiento, no existe un framework que compile el resultado del generador, esta base de conocimiento será utilizada directamente por un razonador para realizar validaciones.

D Generar el software objetivo. En este paso se utilizan los elementos construidos o identificados de los pasos anteriores para generar la aplicación destino, que para este proyecto será la ejecución de la validación de consistencia de la base de conocimiento y la entrega de los resultados.

Los resultados de la ejecución de estos pasos son los componentes del framework, los cuales se describen en detalle en la sección 3.3 de este capítulo.

3.3. Arquitectura

La arquitectura del framework es ilustrada en la figura **3-2**. Debido a que este framework es basado en técnicas de MDE la arquitectura describe como cada modelo proviene o se realiza **conforme a** un metamodelo, donde dicho metamodelo describe los elementos y las relaciones entre elementos que posee cada modelo específico, por ejemplo el metamodelo (1) al que pertenecen los diagramas BPMN especifica que existen elementos como: actividades, actores y compuertas, de igual forma el DSL y la ontología poseen un metamodelo (2)(3) que describe el contenido de los mismos.

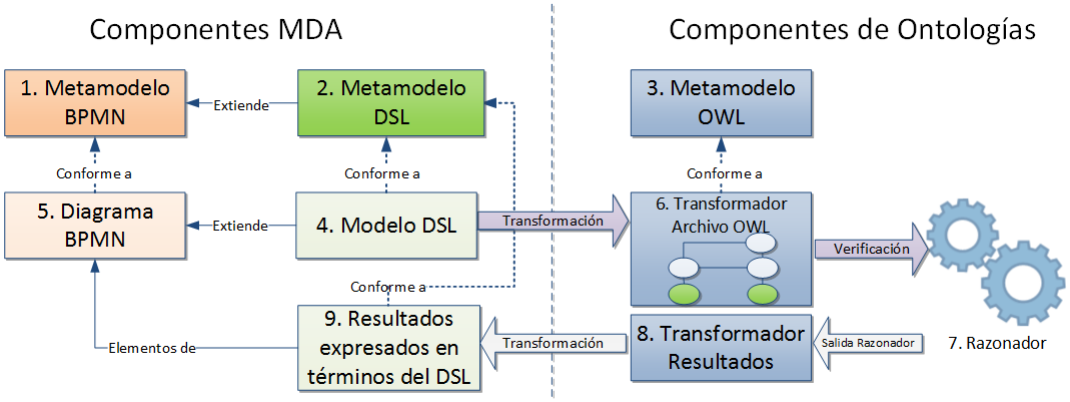


Figura 3-2.: Arquitectura del Framework

La forma en que se relacionan cada uno de los componentes de la arquitectura se describe a continuación:

- Mediante el lenguaje de dominio específico se modelan los elementos y restricciones de un proceso de negocio que necesite ser validado, este **modelo hecho con el DSL (4)** contiene entidades que posteriormente son relacionadas a los elementos del diagrama BPMN.

- Adicionalmente existe el diagrama o **modelo BPMN (5)** de un proceso de negocio al que pertenecen las restricciones descritas en el paso anterior.

- Posteriormente mediante un componente de generación de código tanto el diagrama BPMN como el diagrama modelado con el DSL es transformado en una ontología que es almacenada en un archivo con formato OWL (6).

- Mediante un **razonador (7)** son ejecutados procesos de validación sobre la ontología generada en el paso anterior, con la finalidad de determinar la validez del diagrama BPMN con base en las restricciones de negocio.

- Finalmente un componente de transformación entrega la salida del razonador (8) en un formato comprensible por el usuario (9), es de aclarar que las posibles salidas de la validación son consistencia del diagrama BPMN o inconsistencia del mismo y en el caso de encontrar una inconsistencia se informa el punto del diagrama y la razón por la que presenta la inconsistencia.

El objetivo de este comportamiento es que el usuario modifique el diagrama BPMN cuantas veces sea necesario hasta lograr un diagrama consistente con las restricciones de negocio. Adicionalmente un modelo realizado con el DSL puede ser guardado a manera de plantilla para validar múltiples diagramas BPMN relacionados a un proceso de negocio. A continuación se detalla el diseño de cada uno de los componentes de la arquitectura:

3.3.1. Lenguaje de dominio específico - DSL

El DSL fue desarrollado con el IDE de Microsoft llamado Visual Studio 2010, ya que cuenta con las siguientes herramientas que facilitan la creación de un DSL:

- Kit de desarrollo enfocado a lenguajes de dominio específico "Microsoft Visual Studio Visualization & Modeling SDK" [37].
- "Plantillas de texto T4" Estas plantillas son una combinación de bloques de texto y lógica de control que permiten generar nuevos archivos de texto, mediante las cuales se puede interpretar el contenido de un DSL [38], lo cual es ideal para el marco de este proyecto donde se necesita generar un archivo con formato OWL.

Cada uno de los elementos que pueden representarse en el DSL están contenidos en el cuadro de herramientas que se ilustra en la figura **3-3**, la forma en que estos elementos son usados para diseñar un modelo se ilustra en la figura **3-4**, a continuación se explica el significado de cada uno de los componentes del lenguaje de dominio específico y la forma en la que son utilizados:

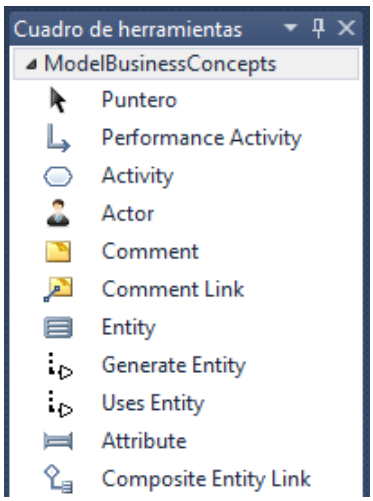


Figura 3-3.: Elementos DSL

- **Activity**: Representa una tarea que es ejecutada en un proceso de negocio, por ejemplo la tarea "Realizar Pedido".
- **Entity**: Representa los datos relacionados a un proceso de negocio como lo son: los clientes, los productos y los pedidos, entre otras posibles entidades. Una entidad puede ser generada como resultado de ejecutar una actividad, como se ilustra en la figura **3-4** en la cual la actividad "Realizar Pedido" genera la entidad "Pedido".
- **Attribute**: Son las propiedades que poseen las entidades, por ejemplo Fecha y Número de pedido son propiedades de la entidad "Pedido", las cuales se representan dentro de la entidad como se ilustra en la figura **3-4**.

- **Composite Entity Link**: La relación de composición es usada cuando una entidad puede contener otra entidad como es el caso de la entidad pedido que contiene la entidad "Item" como se ilustra en la figura **3-4**.
- **Generate Entity**: Cuando una actividad es ejecutada es posible que genere como resultado una entidad, para lo cual se usa este conector que relaciona una actividad con la entidad que genera, como el caso de la actividad "Realizar Pedido" la cual cuando es ejecutada **genera** la entidad "Pedido".
- **Uses Entity**: Cuando una actividad necesitan una entidad para su ejecución se relaciona con la entidad mediante este conector, como la actividad "Validar Pedido" que **usa** la entidad "Pedido".
- **Actor**: Representa un rol en un proceso de negocio el cual puede ejecutar actividades como es el caso de un cliente.
- **Performance Activity**: la relación de ejecución es el conector que relaciona una actor con la actividad que ejecuta, como es el caso del cliente que ejecuta la actividad "Realizar Pedido".
- **Comment**: Sirve para guardar observaciones sobre el modelo, enriquece el modelo con información que puede ser útil para el analista de negocio aunque no se utiliza en los procesos de validación.
- **Comment Link**: Relaciona cualquier elemento del modelo con un comentario.

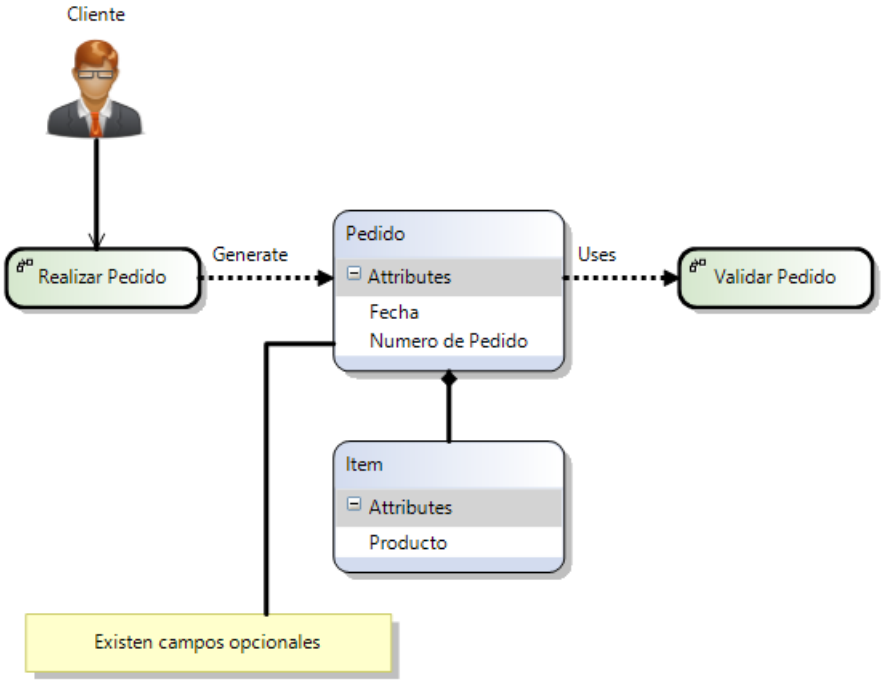


Figura 3-4.: Modelo DSL

El metamodelo del lenguaje de dominio específico describe los elementos y las relaciones válidas entre elementos del modelo. De acuerdo a la herramienta de desarrollo seleccionada debe existir un nodo padre del cual se desprenden todos los tipos de elementos del lenguaje, la figura **3-5** ilustra el elemento padre del metamodelo llamado "ModelBusinessConcept" el cual se relaciona con el elemento "Comment" a través de la relación llamada "ModelBusinessConceptHasComment". Esta relación indica que el modelo puede contener elementos de tipo "Comment", adicionalmente la relación posee propiedades de cardinalidad las cuales indican que puede existir una cantidad infinita de elementos de tipo "Comment" en un modelo.

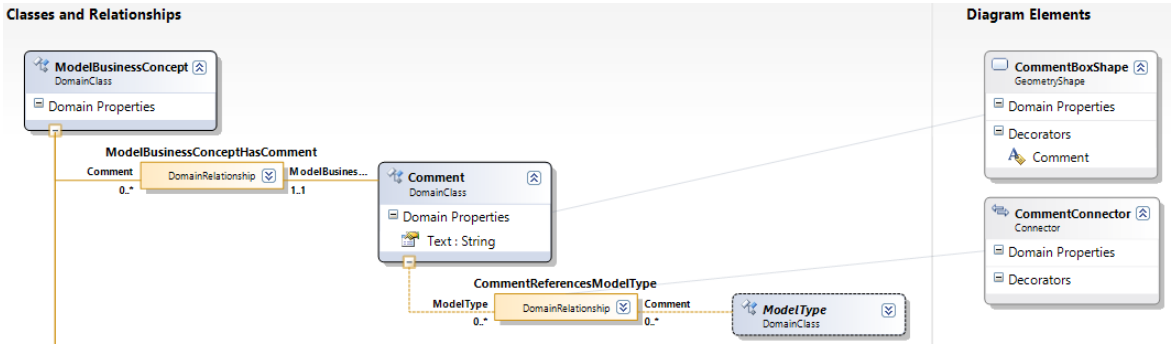


Figura 3-5.: Definición metamodelo del lenguaje

El metamodelo del lenguaje está dividido en dos secciones como se ilustra en la figura **3-5**; la sección de la izquierda llamada **Classes and Relationships** permite definir los elementos y sus relaciones y la sección de la derecha llamada **Diagram Elements** define la forma en que los elementos del lenguaje van a ser visualizados, por ejemplo el elemento "Comment" de la izquierda está relacionado al elemento "CommentBoxShape" de la derecha. El elemento "CommentBoxShape" mediante sus propiedades permite definir que los comentarios sean visualizados mediante cajas cuadradas de color amarillo, como se observó en el figura **3-4**.

Este metamodelo también describe cómo se visualizan los conectores del lenguaje, como se observa en la figura **3-6**, donde el conector "ActorReferencesActivity" está relacionado al componente de diseño llamado "PerformanceConnector", el cual mediante sus propiedades indica qué tipo de línea usará y la imagen que mostrará en la terminación e inicio de la línea que representa la relación entre los elementos. Como se puede ver en la figura **3-3**, donde un "Cliente" se relaciona con la actividad "Realizar Pedido" mediante este conector que es representado por una línea continua, la cual termina en una fecha.

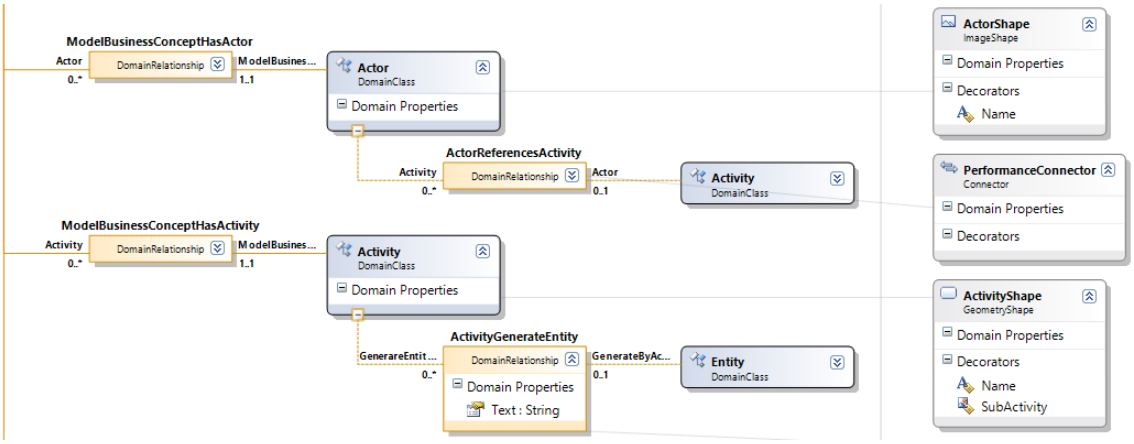


Figura 3-6.: Relaciones entre elementos

Otra característica del metamodelo utilizada es la capacidad de incluir gráficamente un elemento dentro del otro, como es el caso de los atributos los cuales están contenidos dentro de una entidad, para lo cual se incluye una relación de contención como se ilustra en la figura **3-7**, la cual a través de la relación "EntityHasAttributes" indica que uno o varios atributos pueden estar contenidos en una entidad.

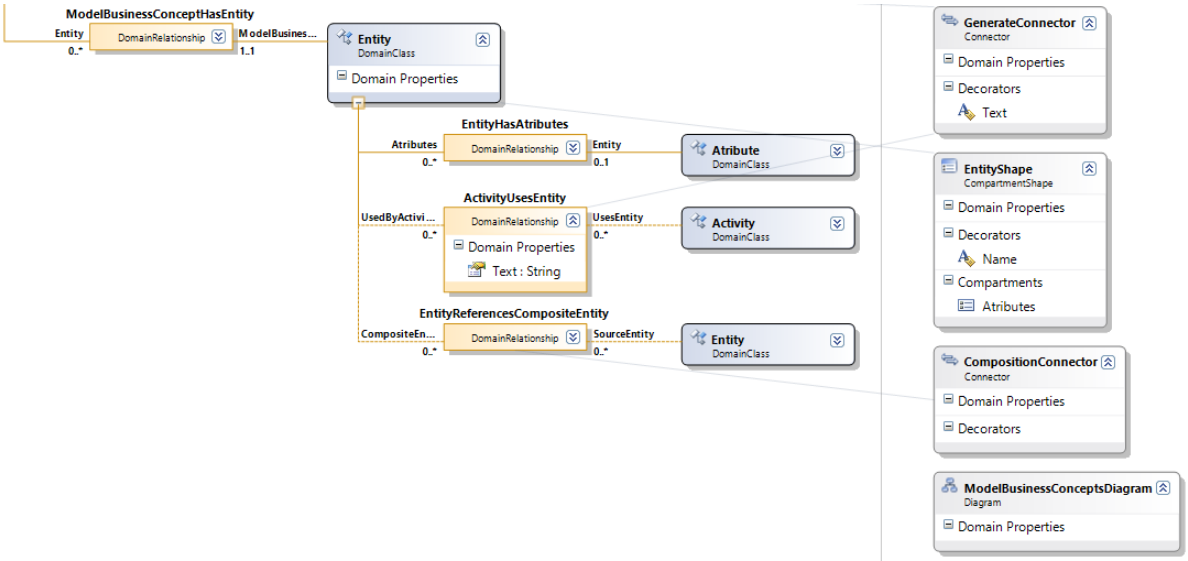


Figura 3-7.: Relaciones de composición

3.3.2. Generador OWL

La herramienta utilizada para desarrollar el generador fue "Plantillas de texto T4" [1] mediante las cuales se puede interpretar el contenido de un DSL y generar archivos de texto [38]. El generador desarrollado lee el modelo DSL y por cada elemento del DSL

[1] es una combinación de bloques de texto y lógica de control que puede generar un archivo de texto

genera su elemento equivalente en lenguaje OWL. La figura **3-8** muestra gráficamente la ontología resultante de transformar cada uno de los elementos del DSL de la figura **3-4** en elementos con notación OWL.

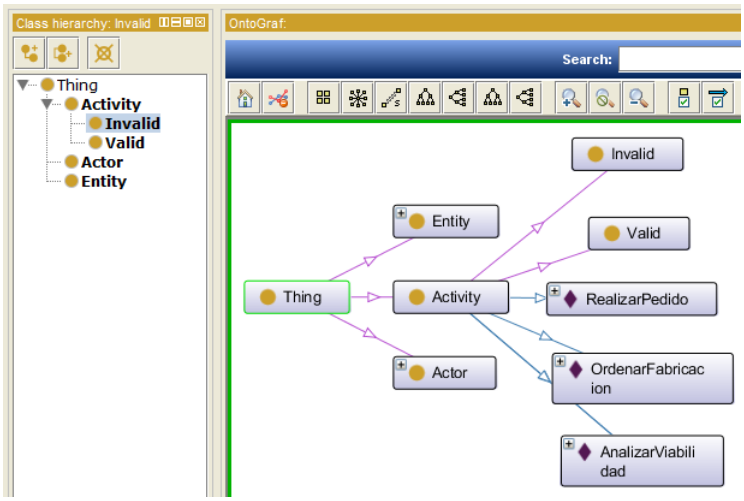


Figura 3-8.: Ontología del framework

La tabla **3-1** ilustra la asociación que existe entre los elementos del DSL y su correspondiente representación en notación del lenguaje OWL. De esta forma cada uno de los elementos es asignado a su respectiva clase o clasificación dentro de la ontología.

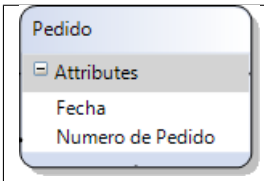
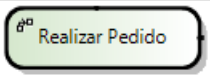
DSL	OWL
	**ClassAssertion**(:Entity :Pedido)
	**ClassAssertion**(:Actor :Cliente)
	**ClassAssertion**(:Activity :RealizarPedido)

Tabla 3-1.: Relación entre elementos del DSL y su notación OWL

Adicionalmente es necesario que las relaciones entre elementos también se transformen en notación OWL. En la tabla **3-2** se ilustra como cada uno de los conectores del DSL puede ser convertido a notación del lenguaje OWL, en la primera fila se dice que la actividad "Realizar Pedido" puede usar la entidad "Pedido" tanto en notación del DSL como en notación OWL, en la segunda fila se expresa que la actividad "Validar Pedido" necesita

la entidad pedido para su ejecución y finalmente en la tercera fila dice que la actividad **Realizar Pedido** es predecesora de la actividad **AnalizarViabilidad**. Esta tercera fila es parte del modelo BPMN que también es transformado y generado en el archivo OWL.

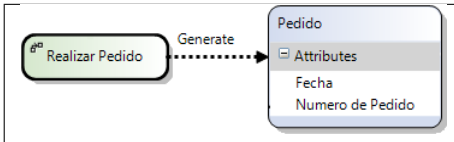
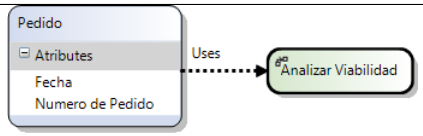
Modelo	OWL
	**ObjectPropertyAssertion**(:canUse :Pedido :RealizarPedido)
	**ObjectPropertyAssertion**(:Uses :AnalizarViabilidad :Pedido)
	**ObjectPropertyAssertion**(:Precedence :RealizarPedido :AnalizarViabilidad)

Tabla 3-2.: Conectores entre elementos del DSL y su notación OWL

Las siguientes sentencias también están definidas dentro de la ontología, ya que son necesarias para el proceso de razonamiento:

- **Declaration**(ObjectProperty(:Precedence))
- **TransitiveObjectProperty**(:Precedence)
- **SubObjectPropertyOf**(ObjectPropertyChain(:canUse :Precedence) :canUse)

La primera sentencia define una propiedad llamada "Precedence" que relacionara dos actividades, la segunda sentencia estipula que esta relación es de tipo **trasnsitiva**, lo que significa que esta propiedad se heredará transitivamente entre actividades, Por ejemplo si se define que la actividad "Ordenar Fabricación" es precedida por la actividad "Analizar Viabilidad" y adicionalmente que la actividad "Analizar Viabilidad" es precedida por "Realizar Pedido", por la característica de transitividad un razonador asumirá que "Ordenar Fabricación" también es precedida por la actividad "Realizar Pedido". Finalmente la tercera sentencia encadena la relación **canUse** con la propiedad de **Precedence**. Ya que la relación **Precedence** es transitiva y está encadenada a la propiedad **canUse** automáticamente las relaciones que permiten que una actividad use una entidad se heredarán a las actividades que se ejecuten posteriormente según se determine en el flujo del diagrama BPMN.

Una vez que el archivo OWL se ha generado en su totalidad, se pueden realizar razonamientos y consultas SPARQL sobre la ontología. La consulta SPARQL de la figura **3-9**

devolverá todas las actividades que estén relacionadas al uso de una entidad pero que no tengan la posibilidad de usar la entidad, es decir no tengan asignada la propiedad "canUse" de esa entidad.

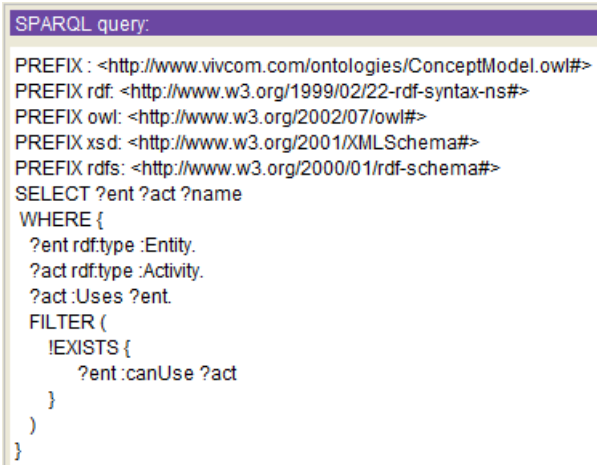


Figura 3-9.: Consulta SPARQL

En el anexo A se ilustra en contenido de un archivo OWL generado a partir de un modelo DSL. De esta forma se pueden identificar las inconsistencias del diagrama BPMN con respecto al modelo DSL.

3.3.3. Razonador

El razonador utilizado para validar la consistencia del archivo OWL es "Pellet" [27], ya que no solo realiza validaciones de consistencia y consultas SPARQL sino que adicionalmente estos procesos de razonamiento se pueden realizar por línea de comando, lo cual es ideal para que este razonador use el archivo OWL generado por el DSL.

Para ejecutar una consulta por línea de comando la distribución de Pellet incluye el archivo **pellet.bat** el cual recibe instrucciones para realizar inferencias. La siguiente línea valida la consistencia de la ontología contenida en el archivo modelo.owl y genera la respuesta del razonamiento en el archivo llamado salida.txt:

pellet.bat consistency modelo.owl >salida.txt

El resultado de ejecutar este comando directamente por consola lo muestra la figura **3-10**

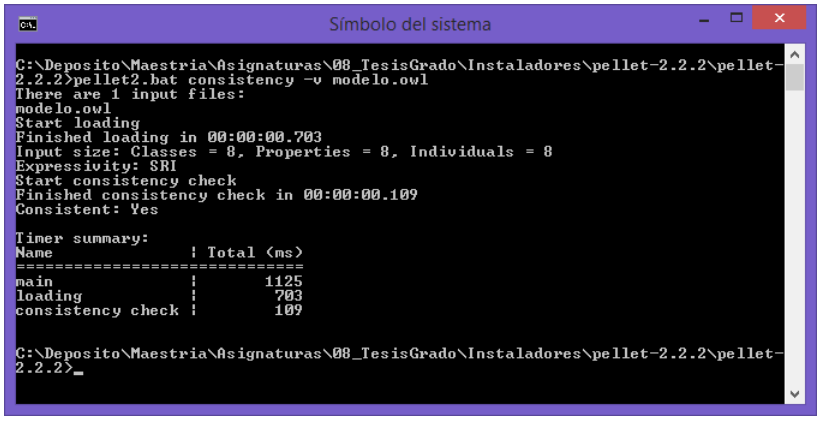


Figura 3-10.: Verificación de consistencia

En caso de encontrar alguna inconsistencia en la ontología el comando **explain** explicaría por qué es inconsistente, indicando la clase que genera dicha inconsistencia.

pellet.bat explain –inconsistent modelo.owl >salida.txt

Una vez se ha determinado que la ontología es consistente, se pueden realizar consultas SPARQL sobre la misma mediante el comando **query**, dicho comando recibe el parámetro $-q$ para indicar el nombre del archivo que contiene la consulta SPARQL y el parámetro $-o$ para indicarle cual es el formato de salida de la consulta; por ejemplo con el siguiente comando se ejecutará la consulta contenida en el archivo "consulta.sparql" y la salida de la consulta se entregará en formato XML:

pellet.bat query -q consulta.sparql -o XML modelo.owl >salida.xml

De esta forma se puede utilizar el razonador "pellet" para implementar los procesos de validación en el framework propuesto.

3.3.4. Transformador de resultados

Una vez se ha obtenido la salida del razonador un formulario es utilizado para visualizar las actividades que presentaron inconsistencias, estas actividades son visualizadas en una tabla, para que al hacer clic sobre cada una de ellas se pueda obtener la razón por la cual la actividad presenta inconsistencia, la figura **3-11** ilustra un ejemplo donde la actividad "Despachar Pedido", no puede utilizar la entidad "Pago".

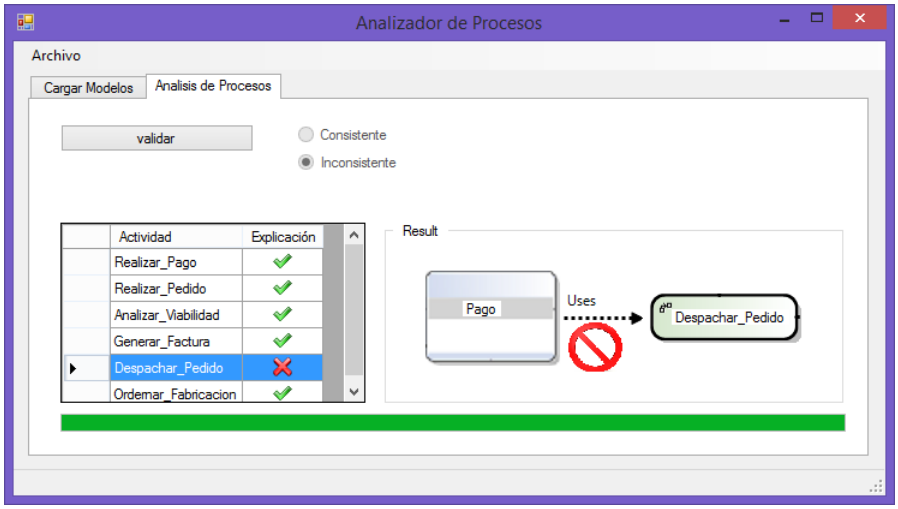


Figura 3-11.: Transformador de resultados

En el anexo A se ilustra el diagrama de clases de este transformador de resultados. De esta forma el transformador muestra resultados comprensibles por el analista que realiza los diagramas BPMN y el modelos del DSL.

4. VALIDACION

Este capítulo está enfocado en proponer algunos escenarios mediante los cuales se valida el funcionamiento del framework de verificación. Estos escenarios permiten observar el comportamiento de las validaciones realizadas sobre los diagramas BPMN. Este capítulo está dividido en tres secciones. En la primera sección 4.1 se describe el caso de estudio, en la segunda sección 4.2 se explica el proceso de verificación y finalmente la tercera sección 4.3 describe los posibles procesos de corrección que se pueden realizar.

4.1. Caso de estudio

El caso de estudio analizado corresponde al proceso de negocio de una compañía dedicada a vender productos que fabrica bajo demanda, es decir que una vez un cliente ha realizado un pedido, la compañía analiza la solicitud para fabricar el producto, cobrar y despachar este producto al cliente.

El primer paso que debe seguir un analista de negocio para utilizar el framework es modelar con el DSL los elementos del proceso de negocio que desea validar, no obstante una de las ventajas de usar este framework es la posibilidad de utilizar plantillas o procesos de negocio estándar, para ahorrar tiempo a la hora de utilizar el DSL.

Las plantillas con las que actualmente cuenta el framework están relacionadas a los procesos claves o de generación de valor, aunque también se puede almacenar procesos apoyo y estratégicos sin ninguna restricción, las plantillas que actualmente están implementadas se listan a continuación y sus respectivos modelos y elementos se pueden observar en el anexo B:

- Proceso de venta
- Pago de facturas
- Proceso de compra
- Selección y Reclutamiento

En la medida que se desarrollen modelos con el DSL estos constituirán una base reutilizable de restricciones de negocio que apoyen diferentes áreas de la organización, razón por la cual antes de modelar con el DSL es necesario identificar modelos diseñados previamente que puedan ser utilizados o adaptados. Para el caso de estudio se seleccionó la plantilla del

"Proceso de venta", la cual se adaptó al proceso objetivo de este análisis. Para el proceso de edición del modelo en el DSL el presente trabajo propone los siguientes lineamientos:

- Identificar los actores o responsables que intervienen en el proceso y representarlos en el modelo.
- Identificar todas las actividades que se desarrollan en el proceso de negocio que se esté analizando y representarlas en el modelo.
- Identificar todas las entidades que son generadas a partir de la ejecución de alguna tarea y asociarlas con su respectiva tarea generadora.
- Finalmente identificar qué insumo necesita cada actividad para su ejecución y relacionar las entidades con las actividades que las usan.

Como resultado de realizar estos pasos se identificaron los actores, actividades y entidades relacionadas en el modelo de la figura **4-1**. En la cual se establecen los siguientes actores:

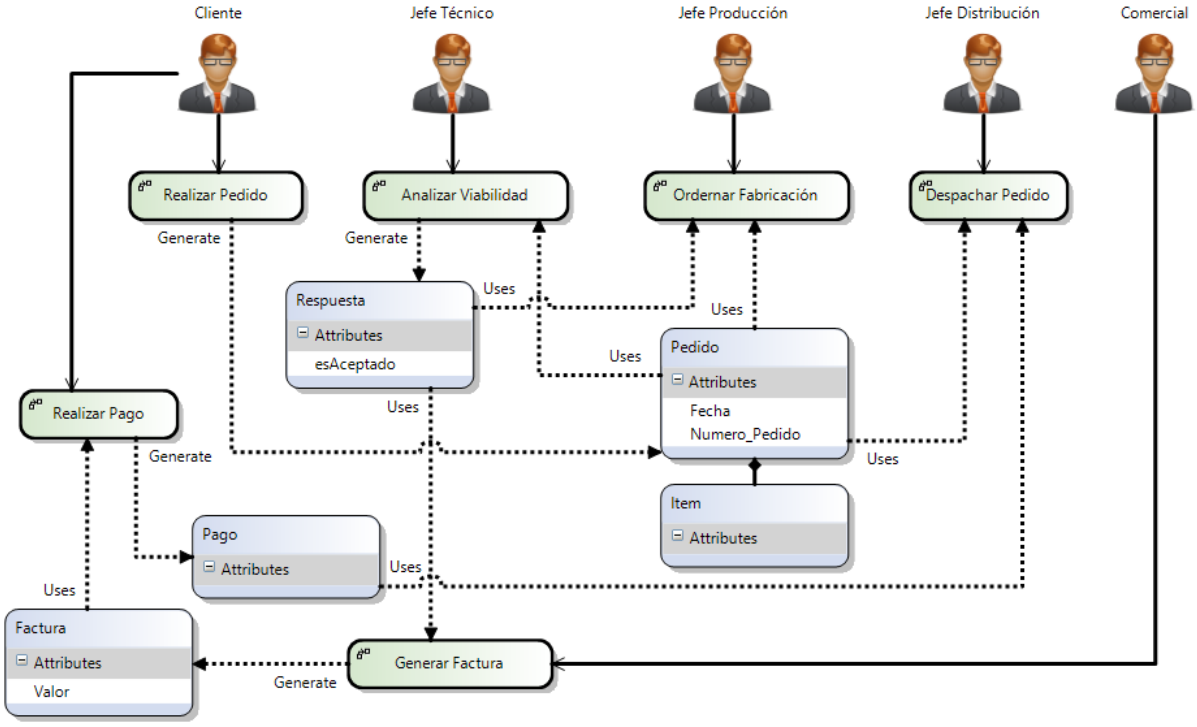


Figura 4-1.: DSL Caso de estudio

- Cliente: rol que ejecuta la actividad "Realizar Pedido", de la cual se genera la entidad "Pedido", adicionalmente este rol ejecuta el proceso "Realizar Pago".
- Jefe Técnico: rol que "Analiza la Viabilidad", es decir aprueba o rechaza la fabricación de un producto solicitado en un pedido y para generar su respuesta usa la información del pedido.

- Jefe Producción: es el rol que autoriza la producción del producto solicitado.
- Jefe de Distribución: rol que se encarga de realizar las tareas de despacho o envío del pedido.
- Comercial: rol encargado de la comunicación con el cliente.

Teniendo como base la información del anterior modelo es decir los roles, actividades y entidades, el analista de negocio realiza el diagrama BPMN de la figura **4-2** del proceso de venta.

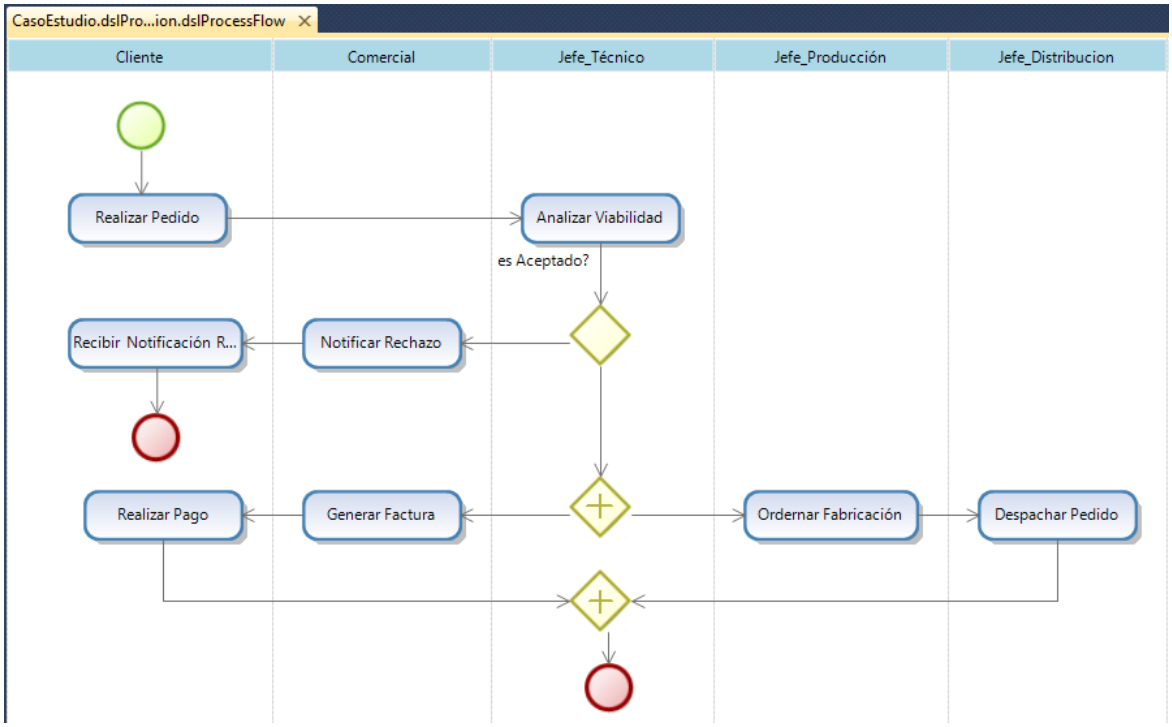


Figura 4-2.: BPMN Caso de estudio

En estos pasos iniciales se definieron los modelos que serán usados en la verificación, las siguientes secciones describen el proceso de validación y corrección.

4.2. Proceso de Verificación

Con base en los modelos DSL y BPMN se puede proceder a realizar la verificación del diagrama BPMN mediante el formulario de validación del framework. Este formulario indica si encontró alguna inconsistencia en el diagrama BPMN, lista todas las actividades del diagrama e identifica cuáles actividades presentan alguna inconsistencia, como se puede observar en la figura **4-3**.

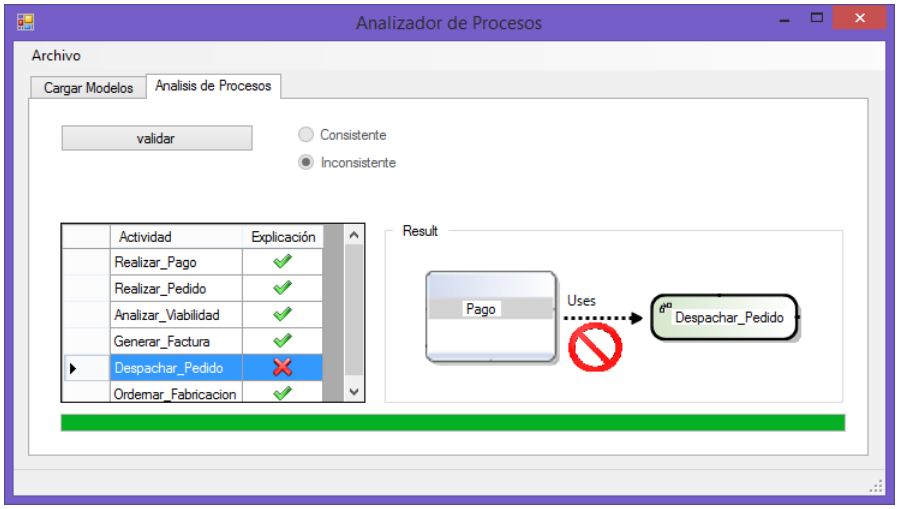


Figura 4-3.: Formulario de validación

El diagrama BPMN es consistente con la notación BPM aunque al realizar la validación de diagrama se encuentra que existe una inconsistencia en la actividad "Despachar Pedido" como se ilustra en la figura **4-3**. Para conocer la razón que genera la inconsistencia el formulario de validación despliega un cuadro de resultados cuando se hace clic sobre la actividad inconsistente. En el caso de la actividad "Despachar Pedido" nos indica que esta actividad necesita la entidad "Pago" para poderse ejecutar pero la entidad pago no existe en el momento que se ejecuta la actividad, esta restricción se puede observar en el modelo del DSL **4-1**, donde explícitamente se estipula que "Despachar Pedido" usa la entidad "Pago", lo cual indica en términos de negocio que no se puede despachar ningún producto a menos que se haya realizado el respectivo pago del mismo. Por otra parte en el diagrama BPMN de la figura **4-2**, la secuencia de pasos no exige que se haya ejecutado la actividad "Realizar Pago" antes de la actividad "Despachar Pedido", razón por la cual se presenta la inconsistencia entre los modelos. De esta forma se puede confirmar como las restricciones representadas en el DSL son complemento del diagrama BPMN.

4.3. Corrección de los modelos

Una vez identificada la razón que la generó la inconsistencia, el analista puede entonces modificar el modelo BPMN según crea conveniente, existen múltiples soluciones posibles y es criterio del analista determinar la mas adecuada según el negocio. La figura **4-4** ilustra una posible solución en la cual se ubican de forma secuencial las actividades del diagrama BPMN.

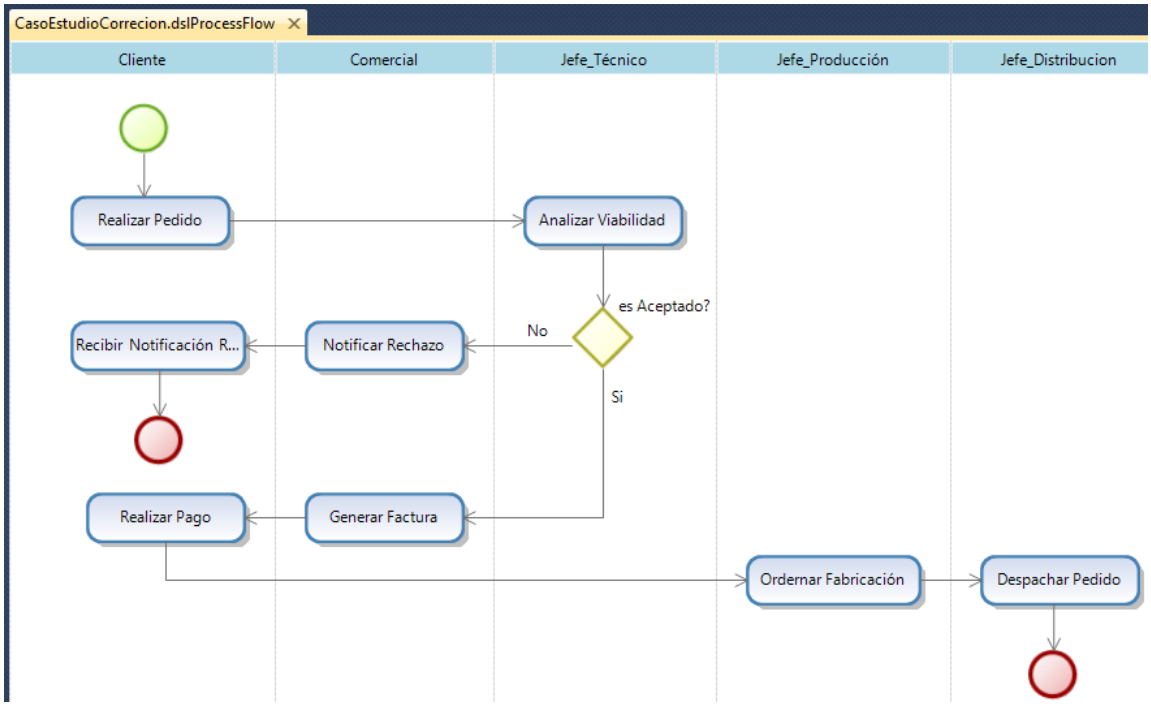


Figura 4-4.: BPMN Corrección

Al realizar la validación del diagrama generado se obtiene un modelo consistente como se puede visualizar en la figura **4-5**, donde no existe ninguna actividad que tenga algún tipo de inconsistencia.

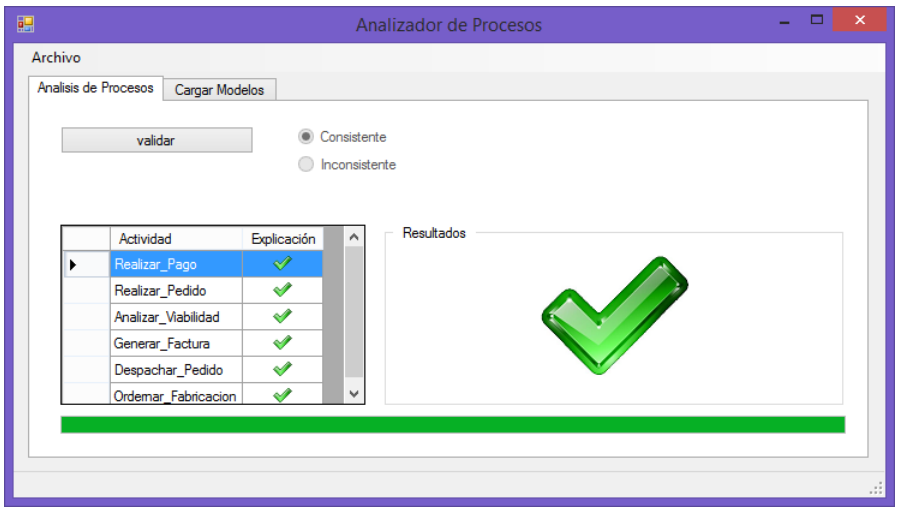


Figura 4-5.: Validación nuevo modelo

Dados los procesos de validación realizados, se encuentra que es posible identificar inconsistencias en la fase de diseño de un diagrama BPMN a través de las restricciones modeladas con el DSL. No obstante, también es factible que las restricciones deban ser modificadas

conforme lo indique el negocio, ya que el objetivo del DSL es apoyar los procesos de diseño de los diagramas BPMN, representando algunas restricciones de los procesos de negocio. Por consiguiente, para el caso analizado es posible que las reglas del negocio dictaminen que a todos los clientes se les puede vender a crédito y realizar los procesos de despacho independientemente del pago, para lo cual el modelo DSL debería ser modificado como ilustra la figura **4-6**, eliminando la relación de uso entre la entidad "Pago" y la actividad "Despachar Pedido" que aparece de color rojo en el diagrama.

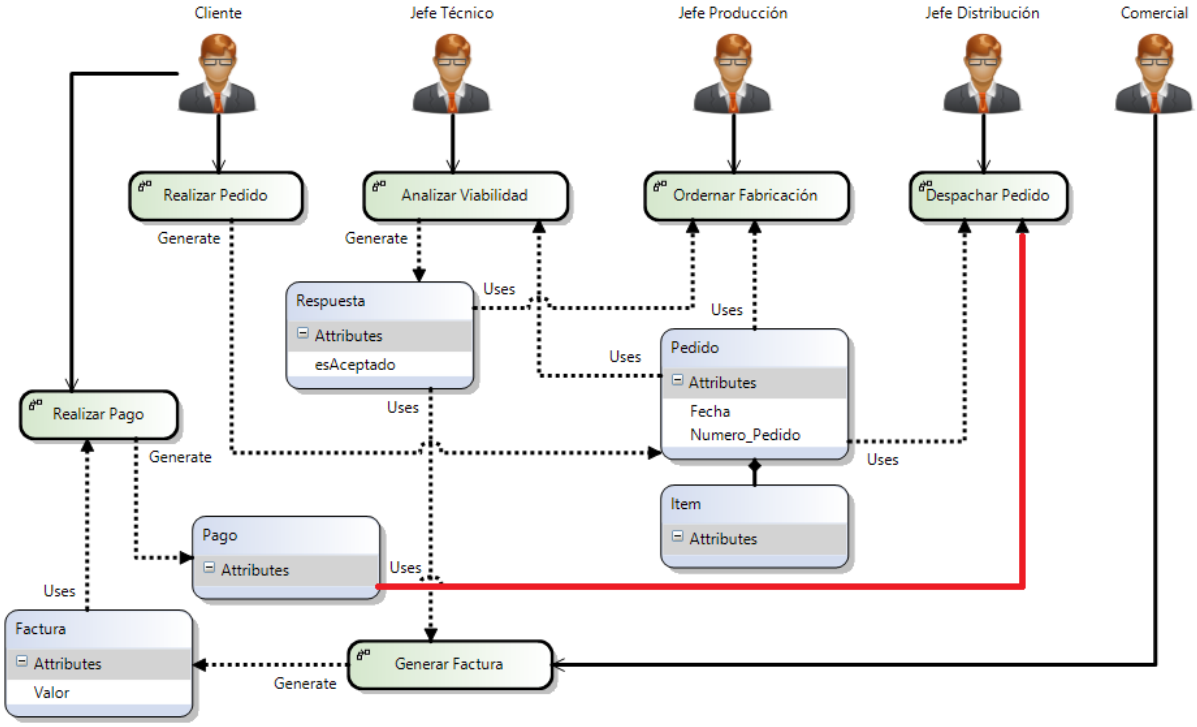


Figura 4-6.: DSL Modificado

Para relacionar cada elemento del diagrama BPMN con su homólogo del lenguaje de dominio específico o DSL, es necesario que los elementos del diagrama BPMN sean enlazados a través de una propiedad. Esta propiedad se denomina "Tipo DSL" y debe especificarse cuando se está modelando el diagrama BPMN cómo se muestra en la figura **4-7**, donde se puede observar que el actor "Cliente" está relacionado al elemento "Cliente" modelado en el DSL. La finalidad de realizar esta asociación es permitir realizar la validación de las actividades que puede ejecutar un determinado tipo de actor. Por ejemplo; según el modelo realizado con el DSL los clientes son los únicos actores que pueden realizar pedidos, esto permitiría validar que esta actividad no se asigne a otro actor.

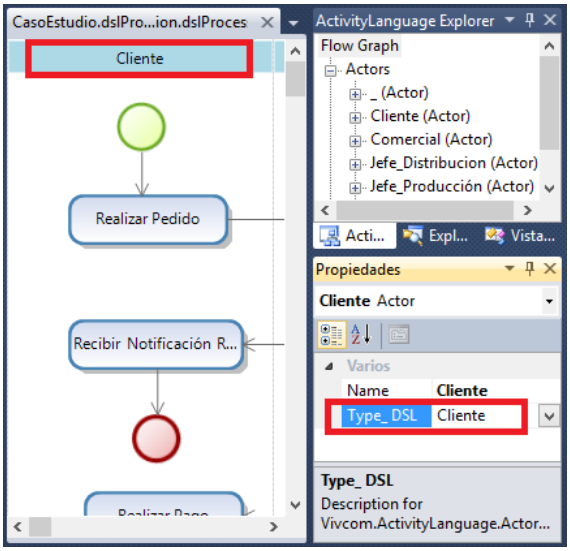


Figura 4-7.: Relación BPMN con el modelo DSL

En el caso de que la actividad “Despachar Pedido” sea asignada al actor “Cliente”, el cual es diferente al tipo de actor “Jefe Distribución” para la que fue diseñada según se modeló con el DSL; el formulario de verificación evidenciará la inconsistencia encontrada cómo se puede observar en la figura **4-8**.

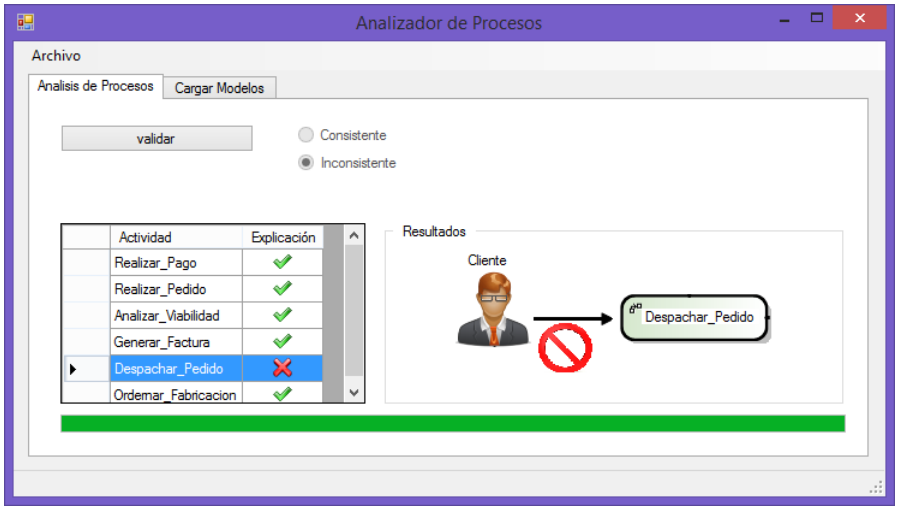


Figura 4-8.: Verificación de la actividad que puede ejecutar un actor

La edición de los modelos se puede realizar de forma iterativa hasta representar adecuadamente el proceso de negocio objetivo, adicionalmente almacenar estos modelos posibilita conservar las restricciones de negocio de la organización.

5. CONCLUSIONES Y TRABAJOS FUTUROS

En este capítulo se concluyen los beneficios encontrados en la validación del framework y se describe posibles mejoras con la visión de incluir el enfoque propuesto en los sistemas de gestión de procesos ya existentes en el mercado.

5.1. Conclusiones

Esta memoria presentó un framework para la verificación de modelos BPMN con la finalidad de apoyar a los proyectos de implementación y mejoramiento de procesos de negocio en las organizaciones. Para lograr este objetivo el framework se basó en técnicas y teorías aplicables en el modelado de restricciones de negocio gestionadas a través de ontologías y mediante las cuales se realizó la verificación de los diagramas BPMN. Para validar el funcionamiento del framework se planteó un caso de estudio que permitiera demostrar el funcionamiento de las verificaciones y su ciclo de corrección. Con base en los resultados obtenidos de este proceso de verificación, se realiza este apartado en el documento con las conclusiones del proyecto.

Al contrastar los requerimientos u objetivos planteados en la definición del framework con los resultados obtenidos en el capítulo de validación, se pueden realizar las siguientes conclusiones:

- Es posible identificar los inconvenientes en el modelo BPMN antes de que se implementen en un BPMS. Esto es, potencialmente, una ventaja al momento de realizar proyectos que implementen o mejoren procesos en una compañía, debido a que es mucho más económico y rápido cambiar un diagrama BPMN mientras se diseña, que modificar el funcionamiento de un sistema mientras se implementa. Esta reducción de costos y tiempo son el factor diferencial que fomentaría el uso de este enfoque.

- Adicionalmente, este enfoque asegura que no se pierda el conocimiento del negocio, manteniendo una base consistente de las restricciones que maneja una compañía en sus procesos y permitiendo que se comparta y utilice por diferentes analistas de la organización.

5.2. Trabajos futuros

Es posible evolucionar este framework adicionado la capacidad de cargar archivos XPDL (XML Process Definition Language). Esto le permitiría cargar modelos BPMN generados con otras herramientas y BPMS y realizar validaciones de sistemas ya implementados.

Este trabajo permitió validar que es factible verificar restricciones de negocio en cuanto la precedencia de actividades sobre diagramas BPMN, lo cual se convierte en la base para crear nuevos elementos del lenguaje que posibiliten representar nuevas restricciones de negocio, como; niveles de autorización, obligatoriedad de actividades en tipos de procesos, verificación de resultados en compuertas lógicas, entre otros.

Por otra parte, al adicionar elementos al DSL para representar tiempos de ejecución de cada una de las actividades y alimentar estos valores con los historiales de ejecución de los BPMS, permitiría realizar análisis que determinen los cuellos de botella en los procesos ya implementados, extendiendo así el uso de este framework.

Si un BPMS obtuviera estos beneficios podría realizar minería de procesos de negocio, lo cual sería un factor diferencial en este tipo de sistemas. Un ejemplo de ello es el caso de "Oracle BPM" [31] que hoy en día representa sus procesos de negocio mediante ontologías y las utiliza para realizar validaciones en cuanto a niveles de autorización, este BPMS se describe en la sección de trabajos relacionados 2.4.

Bibliografía

[1] Gartner. Gartner survey shows more than half of respondents plan to increase spending on BPM by more than 5 percent in next 12 months, August 2009.

[2] Jakob Freund and Bernd Rücker. *Real-Life BPMN: Using BPMN 2.0 to Analyze, Improve, and Automate Processes in Your Company*. CreateSpace Independent Publishing Platform, 1 edition edition, October 2012.

[3] David S. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. Wiley, New York, 1 edition edition, January 2003.

[4] Steven Kelly and Juha-Pekka Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley-IEEE Computer Society Pr, Hoboken, N.J, 1 edition edition, March 2008.

[5] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, June 1993.

[6] W3C, Michael K. Smith, Chris Welty, and Deborah L. McGuinness. OWL web ontology language guide, February 2004.

[7] Gartner. Business process improvement, May 2014.

[8] Van Der Aalst and Wil M. P. Business process management: A comprehensive survey. *International Scholarly Research Notices*, 2013:e507984, February 2013.

[9] OMG. Business process model & notation (BPMN), January 2011. http://www.omg.org/bpmn/.

[10] Michael Hammer and James Champy. *Reengineering the Corporation: A Manifesto for Business Revolution*. Harper Business, New York, edición: rev upd edition, October 2006.

[11] Martin Fowler. *Domain-Specific Languages*. Addison-Wesley Professional, Upper Saddle River, NJ, 1 edition edition, October 2010.

[12] Mathias Weske. *Business Process Management - Concepts, Languages, Architectures*. Springer Berlin Heidelberg, 1 edition edition, September 2007.

[13] Tristan Boutros and Tim Purdie. *The Process Improvement Handbook: A Blueprint for Managing Change and Increasing Organizational Performance*. McGraw-Hill Professional, New York, 1 edition edition, October 2013.

[14] OMG. Object management group, 1997. http://www.omg.org/.

[15] Jishnu Mukerji and Joaquin Miller. OMG document – omg/03-06-01 (MDA guide v1.0.1), June 2003.

[16] Kavi Mahesh. *Ontology Development for Machine Translation: Ideology and Methodology*. 1996.

[17] W3C, Dan Brickley, and R.V. Guha. RDF vocabulary description language 1.0: RDF schema, February 2004.

[18] W3C, Eric Prud'hommeaux, and Andy Seaborne. SPARQL query language for RDF, January 2008.

[19] W3C, Deborah L. McGuinness, and Frank van Harmelen. OWL web ontology language overview, February 2004.

[20] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, Cambridge, 2 edition edition, June 2010.

[21] W3C. SWRL a semantic web rule language combining OWL and RuleML, May 2004. http://www.w3.org/Submission/SWRL/#2.1.

[22] Information Process Engineering (IPE), Institute of Applied Informatics and Formal Description Methods (AIFB), and Information Management Group (IMG). KAON2 – ontology management for the semantic web, 2004. http://kaon2.semanticweb.org/.

[23] Daniel Oberle, Steffen Staab, and Raphael Volz. An application server for the semantic web. page 220. ACM Press, 2004.

[24] Information System Group. HermiT reasoner: Home, August 2008. http://www.hermit-reasoner.com/.

[25] Boris Motik, Rob Shearer, and Ian Horrocks. Optimized reasoning in description logics using hypertableaux. In Frank Pfenning, editor, *Automated Deduction – CADE-21*, number 4603 in Lecture Notes in Computer Science, pages 67–83. Springer Berlin Heidelberg, January 2007.

[26] C&P internship. Pellet OWL 2 reasoner for java, 2005. http://clarkparsia.com/pellet/.

[27] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet a practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, June 2007.

[28] Institute für Informatik der Universität zu Lübeck. Racer: IFIS uni lübeck, 2001. www.ifis.uni-luebeck.de/index.php?id=385.

[29] Volker Haarslev, Ralf Möller, and Michael Wessel. Querying the semantic web with racer + nRQL. In *In Proceedings of the KI-2004 International Workshop on Applications of Description Logics (ADL'04*, 2004.

[30] SUPER. Semantics utilised for process management within and between enterprises, October 2009.

[31] Hong Fan, Wu Du, and Yao Zhang. Oracle semantic technology enable smart discovery of spatial services. In *Geoscience and Remote Sensing Symposium (IGARSS), 2010 IEEE International*, pages 3995–3998, July 2010.

[32] M.S. Fox, M. Barbuceanu, and M. Gruninger. An organisation ontology for enterprise modelling: preliminary concepts for linking structure and behaviour. In *, Proceedings of the Fourth Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, 1995*, pages 71–81, April 1995.

[33] William E. McCarthy. The REA accounting model: A generalized framework for accounting systems in a shared data environment. *The Accounting Review*, 57(3):554–578, July 1982. ArticleType: research-article / Full publication date: Jul., 1982 / Copyright © 1982 American Accounting Association.

[34] James C. Lampe. Discussion of an ontological analysis of the economic primitives of the extended-REA enterprise information architecture. *International Journal of Accounting Information Systems*, 3(1):17–34, March 2002.

[35] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Q.*, 28(1):75–105, March 2004.

[36] Alan Hevner and Samir Chatterjee. *Design Research in Information Systems - Theory and Practice*, volume 22. Springer, 2010 edition, 2010.

[37] Microsoft. Modeling SDK for visual studio - domain-specific languages, 2013. http://msdn.microsoft.com/en-us/library/bb126259.aspx.

[38] Microsoft. Generación de código y plantillas de texto t4, 2013. http://msdn.microsoft.com/es-co/library/bb126445.aspx.

[39] Microsoft. System.xml (espacio de nombres), 2014. http://msdn.microsoft.com/es-es/library/system.xml%28v=vs.110%29.aspx.

[40] Microsoft. Windows forms, 2014. http://msdn.microsoft.com/en-us/library/dd30h2yb%28v=vs.110%29.aspx.

[41] Bizagi. Bizagi process modeler user's guide, 2014. http://help.bizagi.com/processmodeler/es/index.html?process_templates.htm.