Lenguaje de dominio específico para generar facturas electrónicas de acuerdo a los requerimientos técnicos de la DIAN - invoiceQL

Edwar Alonso Rojas Blanco

Universidad Nacional de Colombia
Facultad de Ingeniería, Departamento de Ingeniería de Sistemas e Industrial
Bogotá, D. C., Colombia
2020

Lenguaje de dominio específico para generar facturas electrónicas de acuerdo a los requerimientos técnicos de la DIAN - invoiceQL

Edwar Alonso Rojas Blanco

Trabajo de grado presentado como requisito parcial para optar al título de:
Magíster en Ingeniería - Ingeniería de Sistemas y Computación

Director(a):
M.Sc., Henry Roberto Umaña Acosta

Línea de Investigación:
Ingeniería de Software Dirigida por Modelos
Grupo de Investigación:
Colectivo de Investigación en Ingeniería de Software (ColSWE)

Universidad Nacional de Colombia
Facultad de Ingeniería, Departamento de Ingeniería de Sistemas e Industrial
Bogotá, D. C., Colombia
2020

Este trabajo está dedicado a mis padres y hermano por su incondicional amor y apoyo. También a mi pequeño hijo Ed. M. y si lee esto alguna vez, recuerde que con pasión y disciplina todo es posible.

Agradecimientos

Agradezco a mis padres y hermano por el apoyo incondicional en el cumplimiento de mis proyectos.

Muchas gracias al profesor Henry Roberto Umaña Acosta por su invaluable guía y apoyo durante el desarrollo de este proyecto. También agradezco a los profesores y compañeros de la Universidad Nacional de Colombia con los que tuve la oportunidad de compartir experiencias y recibir valiosas enseñanzas para mi vida profesional y personal.

Resumen

La DIAN (Dirección de impuestos y aduanas nacionales) está implementando el proceso de facturación electrónica en Colombia, esto implica que los sistemas de información de las organizaciones (como los ERP) que tienen que ver con la facturación deban implementar nuevos requerimientos. En este trabajo se presenta el lenguaje de dominio específico llamado InvoiceQL que permite generar facturas electrónicas tan solo con escribir algunas instrucciones. Para desarrollar InvoiceQL se utilizó una metodología basada en desarrollo de software dirigido por modelos o MDSD con una variante llamada MDDF(Desarrollo de funcionalidades dirigido por modelos) y con la ayuda de las herramientas de modelado de Eclipse (más específicamente el framework EMF) se creó un programa interprete que genera facturas electrónicas directamente desde sentencias InvoiceQ. InvoiceQL tambien puede generar código fuente en lenguaje Python que al ejecutarse genera facturas electrónicas.

Palabras clave: Desarrollo de software dirigido por modelos MDSD, Desarrollo de funcionalidades dirigido por modelos MDF,facturación electrónica e-invoicing, lenguaje de dominio específico DSL, sistema de planificación de recursos empresariales ERP..

Abstract

The DIAN (Dirección de impuestos y aduanas nacionales) is implementing the electronic invoicing process in Colombia, this implies that the information systems in the organizations (such as ERPs) they have to implement new requirements. In this work the specific domain language InvoiceQL is presented, this language allows generating electronic invoices just by typing some instructions. To develop InvoiceQL was used a methodology based on model-driven software development or MDSD with a variant that we have called MDDF(Model-driven development of functionality). With the help of Eclipse modeling tools (more specifically the EMF framework) an interpreter program was developed, this program can generates electronic invoices directly and can also generate source code in Python language that can be integrated into the ERP ODOO to generate electronic invoices.

Keywords: MDSD Model-driven software development, e-invoicing electronic invoicing, DSL domain specific language, MDDF Model-driven development of functionality, ERP enterprise resource planning system.

Contenido

Lista de Figuras

Lista de Tablas

1. Introducción

La facturación electrónica o *e-invoicing* es un proceso mediante el cual se digitaliza la factura de venta y los documentos asociados para mejorar la productividad, agilizar los procesos administrativos, y lograr transparencia en las transacciones comerciales [18]. En Colombia la DIAN (Departamento de impuestos y aduanas nacionales) es la entidad gubernamental encargada de implementar el proceso de facturación electrónica. A la fecha de la elaboración de este trabajo (año 2020) recién se está implementando la facturación electrónica de manera obligatoria a los contribuyentes en Colombia. La factura electrónica debe seguir el estándar UBL 2.1 y además cumplir con las condiciones técnicas que exige la DIAN incluyendo el adecuado diligenciamiento de los 470 campos (de acuerdo al ultimo manual técnico expedido por la DIAN) que puede contener este documento.

Debido a lo anterior, los sistemas de información (como los ERP) encargados de la facturación en las organizaciones deben implementar los nuevos requerimientos y se deben actualizar cada vez que haya cambios en la reglamentación legal. En este trabajo se presenta el desarrollo de InvoiceQL, que es un lenguaje de dominio específico (DSL, por sus siglas en inglés) que sirve para generar facturas electrónicas e introducir de forma rápida y flexible los cambios que se generan a partir de los nuevos requerimientos derivados de la normativa legal.

InvoiceQL permite generar la factura electrónica por medio de una *transformación directa*, es decir, de instrucciones InvoiceQL a un documento XML(que es el formato de la factura electrónica) para ello, se desarrolló un programa interprete escrito con Xtend.

Para el desarrollo de InvoiceQL se utilizó una metodología cíclica basada en desarrollo de software dirigido por modelos (MDSD por sus siglas en inglés) y que cuenta con 8 etapas: 1) implementación de referencia, 2) análisis, 3) elaboración del metamodelo, 4) validación del metamodelo, 5) diseño del DSL basado en el metamodelo, 6) diseño de transacciones, 7) implementación del DSL y 8) caso de uso. En cada etapa siempre se tienen en cuenta tres aspectos: Dominio, arquitectura y tecnología de desarrollo. Se propone en este trabajo una variante de MDSD que llamamos MDDF(Model-driven development of functionality o Desarrollo de funcionalidades dirigido por modelos) y consiste en enfocarse en los requerimientos de la funcionalidad más que en los requerimientos de software. Por lo tanto, invoiceQL está diseñado para generar la factura electrónica directamente y no un software que la genere, sin embargo, se propone en este trabajo una opción de *transformación indirecta* que consiste en

aprovechar los elementos del metamodelo para generar código en python capaz de construir facturas electrónicas y que puede ser integrado en sistemas ya desarrollados como el ERP ODOO.

El proceso de facturación electrónica definido por la DIAN incluye funciones de envío y verificación de los documentos electrónicos, sin embargo, este trabajo se limita únicamente a la generación del documento factura electrónica. También cabe mencionar que además de la factura (Invoice) hay otros 4 documentos legales que sirven para diferentes operaciones y que no se desarrollan en este trabajo, estos son: nota crédito(credit note), nota débito(debit note), contenedor de documentos(attached document), registro de eventos(application response). El diseño de InvoiceQL esta pensado para seguir ampliando el lenguaje en trabajos futuros que incluyan los documentos mencionados anteriormente y así llegar a una completa solución de facturación electrónica.

1.1. Definición del problema

La labor de mantenimiento del software o 'software maintenance' se refiere al conjunto de actividades que deben realizarse para corregir errores y hacer modificaciones de acuerdo a nuevos requerimientos funcionales o no funcionales [1].

Mantener un software puede ser costoso y complicado especialmente cuando los desarrolladores no están familiarizados con el código fuente del proyecto pues deben pasar mucho tiempo entendiendo y aprendiendo acerca del código. Dependiendo de la estructura que tenga el proyecto (puede ser por componentes o módulos) y el acoplamiento de cada una de las partes del software las tareas de mantenimiento pueden tener mayor o menor complejidad y por lo tanto ser más o menos costosas [2].

Además cuando el software tiene alto acoplamiento (alta dependencia entre sus componentes) es difícil predecir como responderá el sistema ante los nuevos cambios y debido a esta incertidumbre es un gran problema hacer mantenimiento a un software con esta característica, este problema de diseño sucede a menudo porque las organizaciones se enfocan en empezar a utilizar el software lo más pronto posible y no tanto en su calidad [3]. Por lo tanto, al realizar modificaciones se pueden inducir errores que desestabilizan el software.

Debido a lo anterior, tener estrategias para hacer que los proyectos de software tengan bajo acoplamiento es muy importante. Una de estas estrategias es utilizar DSL (domain-specific language por sus siglas en inglés) que facilita las tareas de mantenimiento al aislar la lógica de dominio de la lógica del sistema. Los DSL facilitan el mantenimiento del software por que: Ayudan a identificar fácilmente código problemático asociado a un dominio, se pueden añadir funciones que hacen mas comprensible el código, y ayudan a adaptar el código existente cuando hay nuevos requerimientos [4].

Los software que ayudan a gestionar procesos empresariales que dependen de normas legales (como procesos contables, nóminas, entre otros) son un caso de sistemas cuyos requerimientos cambian frecuentemente por lo que necesitan mantenimiento constante y por lo tanto su diseño debe permitir realizar cambios sin que haya desestabilizaron, es decir, debe haber la mínima dependencia posible entre sus módulos o componentes (Bajo acoplamiento).

La facturación electrónica es un proceso empresarial que depende mucho de la reglamentación gubernamental. En Colombia desde el año 1995 se habla de facturación electrónica con la ley 223 (artículo 37) en el que "la factura electrónica se equipara como documento de venta de igual validez que la factura tradicional en papel", sin embargo, no es hasta el año 2016 (mediante la resolución 000019 de la DIAN) cuando se determinan criterios técnicos y se hace un plan piloto de facturación electrónica en Colombia. Desde el año 2017 de forma

escalonada se ha empezado a establecer la facturación electrónica de carácter obligatorio para diferentes tipos de empresas. Desde entonces los requerimientos técnicos para generar la expedición de facturas electrónicas no han hecho mas que cambiar. De hecho durante el desarrollo de este trabajo se emitió por parte de la DIAN la resolución 000042 del 05 de mayo de 2020 "Por la cual se desarrollan los sistemas de facturación"lo que implica modificaciones en las especificaciones técnicas para generar facturas electrónicas en Colombia.

Algunas de las últimas modificaciones técnicas en la normativa de la DIAN con respecto a la facturación electrónica son las siguientes [5]:

- Resolución 000019 de febrero 24 de 2016. Anexo Técnico 002, Política de Firma de los Documentos XML de Facturación Electrónica.

- Resolución No. 000001 de 2019. En el que se establece UBL (Universal Business Language) com estándar para la generación de cinco tipos de documento: Invoice (factura), CreditNote (Nota Crédito), DebitNote (Nota Débito), ApplicationResponse (Registro de Evento2) y AttachedDocument (Contenedor de Documentos).

- Resolución No. 000020 de 2019. "La Dirección de Impuestos y Aduanas Nacionales -DIAN podrá establecer las condiciones, los términos y los mecanismos técnicos y tecnológicos para la generación, numeración, validación, expedición, entrega al adquiriente y la transmisión de la factura o documento equivalente, así como la información a suministrar relacionada con las especificaciones técnicas y el acceso al software que se implemente."

- Resolución No. 000030 de 2019. En donde se establecen los requisitos de la factura electrónica de venta con facturación previa a su expedición, y las condiciones, términos y mecanismos técnicos para su implementación.

- Resolución No. 000064 de 2019. ": Indica el plazo dentro del cual el sujeto obligado a expedir factura electrónica de venta, debe registrarse como facturador electrónico y señalar el software de facturación con el cual hará pruebas de habilitación en el servicio informático electrónico de factura electrónica"(Deben realizarse pruebas técnicas de habilitación del software)

- Resolución No. 000083 de 2019: Precisiones sobre la factura electrónica.

- Resolución No. 000042 de 2020: Por la cual se desarrollan los sistemas de facturación, los proveedores tecnológicos, el registro de la factura electrónica de venta como título valor, se expide el anexo técnico de factura electrónica de venta y se dictan otras disposiciones en materia de sistemas de facturación.

Este cambio frecuente en las especificaciones técnicas que deben ser adoptadas por los sistemas de información que ayudan a gestionar el proceso de facturación electrónica genera problemas de mantenimiento, por lo que surge la siguiente pregunta:

¿Cómo lograr que un sistema de información que implementa la facturación electrónica en Colombia separe la lógica necesaria para cumplir con los requerimientos de la DIAN de la lógica de negocio del sistema?

Como se mencionaba al comienzo de esta sección utilizar un DSL es una buena estrategia para facilitar el mantenimiento del software. Por eso, en este trabajo se propone el desarrollo del DSL llamado **invoiceQL** (o lenguaje de consulta para facturas) que tiene como propósito aislar la lógica de negocio de los sistemas de información de la lógica necesaria para generar y manipular las facturas electrónicas.

La metodología utilizada para el diseño del lenguaje invoiceQL se basa en el desarrollo de software dirigido por modelos (MDD, model-diven development por sus siglas en inglés), ya que este enfoque facilita realizar modificaciones de las especificaciones técnicas a partir de un metamodelo en el que se representan las diferentes reglas para la generación de la factura electrónica.

1.2. Objetivos

1.2.1. Objetivo General

Diseñar una herramienta que ayude a integrar la facturación electrónica con sistemas información de acuerdo a los requerimientos técnicos de la DIAN mediante un lenguaje de dominio específico o DSL.

1.2.2. Objetivos Específicos

- Discriminar los elementos de UBL utilizados por la DIAN comparando el anexo técnico de la DIAN con la documentación de UBL.
- Determinar las reglas semánticas y sintácticas de invoiceQL mediante la definición de la gramática del lenguaje.
- Identificar los elementos necesarios para Integrar invoiceQL con un sistema ERP agregando un módulo de facturación electrónica al sistema de código abierto Odoo.

1.3. Delimitación del trabajo

Como se muestra en el anexo técnico de la resolución de la DIAN No. 000042 del 05 de mayo de 2020, en Colombia se definen 5 documentos electrónicos para el sistema de facturación electrónica [25] :

- Factura (Invoice)
- Nota Crédito (CreditNote)
- Nota Débito (DebitNote)
- Application Response
- Attached Document

Sin embargo, este trabajo se limitará únicamente a la generación de la Factura (Invoice). El prototipo desarrollado con este trabajo no realizará las operaciones de validación ni de envío del documento hacia la DIAN.

1.4. Esquema del documento

En el *capítulo 1(Introducción)* se muestra la definición del problema y los objetivos que este proyecto se propone cumplir. En el *capítulo 2(Contexto y trabajo relacionado)* se presentan

los conceptos clave para entender este trabajo, tales como: facturación electrónica en general y en Colombia, lenguaje de comercio Universal UBL(por sus siglas en inglés Universal Business Language), lenguajes de dominio específico DSL(por sus siglas en inglés domain-specific language) y desarrollo de software dirigido por modelos MDD (por sus siglas en inglés model-driven development), además se hace referencia a trabajos previos relacionados con este proyecto. En el *capítulo 3:(Implementación del lenguaje InvoiceQL con MDD)* se describe el desarrollo de invoiceQL siguiendo el paradigma MDD. En el *capítulo 4:(Evaluación)* se hace una evaluación conceptual y cuantitativa de los resultados obtenidos con este trabajo. Finalmente en el *capítulo 5 (Conclusiones y recomendaciones)* se habla principalmente de trabajos futuros y de la evolución que podría tener InvoiceQL.

2. Contexto y trabajo relacionado

La factura electrónica es un documento en formato XML (para el caso de Colombia con las especificaciones de UBL 2.1) que es generado por un software de acuerdo a un marco legal que fija las directrices técnicas. En este trabajo se propone desarrollar un DSL llamado InvoiceQL que permite generar facturas electrónicas de acuerdo a las especificaciones técnicas de la DIAN (Dirección de impuestos y aduanas nacionales, de Colombia) para lo cual se sigue el paradigma de desarrollo de software dirigido por modelos (o MDD). Como ejemplo de la utilidad de InvoiceQL se desarrolla un módulo de facturación electrónica para el ERP de código abierto ODOO.

Por lo anterior es importante tener claridad sobre algunos conceptos. A continuación se muestran aspectos importantes de cada uno de estos conceptos.

2.1. Facturación electrónica

La facturación electrónica o *e-invoicing* es un proceso mediante el cual se digitaliza la factura de venta y los documentos asociados para mejorar la productividad, agilizar los procesos administrativos, y lograr transparencia en las transacciones comerciales [18].

La facturación electrónica tiene como principal objetivo proporcionar interoperabilidad entre los diferentes procesos de negocio, puesto que la factura es un documento que sirve de insumo para diferentes procesos (por ejemplo el proceso contable, comercial, legal, entre otros). Debido a la importancia de la facturación electrónica diferentes organizaciones internacionales como OASIS, UN/CEFACT, GS1, o CEN se han preocupado por estandarizar el proceso (más adelante trataremos UBL el estándar propuesto por OASIS) [7]. La estandarización es muy importante para la interoperabilidad, además de que la facturación electrónica por lo general es impuesta por el gobierno.

E-invoicing presenta la ventaja de tener una trazabilidad muy detallada de la factura desde el momento en que se emite hasta cuando es declarada al ente gubernamental encargado, por este motivo muchos gobiernos en el mundo implementan este mecanismo en sus sistemas tributarios pues es una gran herramienta para prevenir y perseguir la evasión fiscal.

De acuerdo al reporte "The e-invoicing journey 2019-2025" [8] publicado por la empresa Suiza Billentis de amplia trayectoria en consultoría acerca de facturación electrónica la adopción de la e-invoice por parte de los gobiernos es muy amplia, se estima que para el 2035 la factura de papel haya sido totalmente reemplazada por la electrónica. La figura **2-1** se muestra el grado de madurez de la implementación de la facturación electrónica en el mundo.

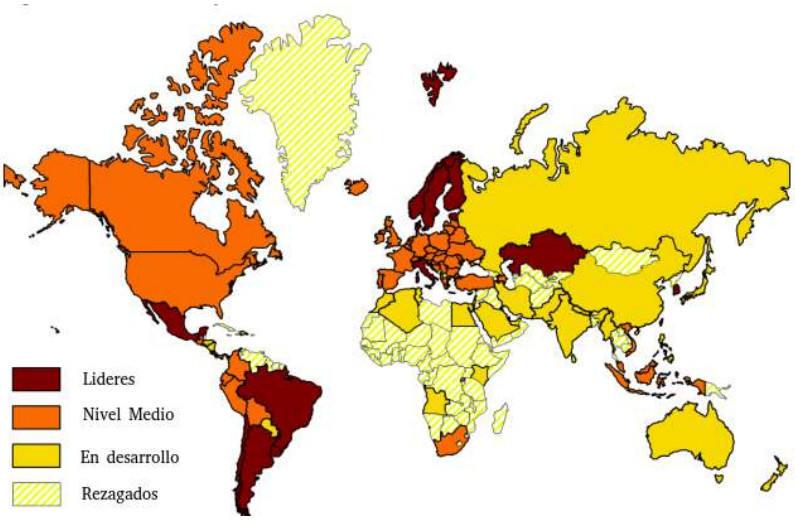

Figura 2-1.: Maduréz de la implementación de la factura electrónica en el mundo. Fuente: Billentis 2019.

Como podemos observar en el mapa, con respecto a latinoamérica países como México, Argentina, y Brasil son líderes en la implementación de la factura electrónica, y otros como Perú, Ecuador y Colombia esta en fase de desarrollo.

En el caso particular de Colombia desde hace varios años se habla de facturación electrónica (ley 223 de 1995, ya establece la validez de la factura electrónica), sin embargo, hasta el año 2016 con la resolución No. 000019 de la DIAN se determinan las condiciones técnicas para la implementación de la factura electrónica en Colombia. La DIAN escojió UBL 2.1 (Universal Business Language) como estándar para la generación de documentos electrónicos, y establece tres alternativas válidas para expedir la factura electrónica [9]:

- Facturación Gratuita DIAN: Es el software gratuito ofrecido por la DIAN, sin embargo, tiene limitaciones de integración a otros software y de almacenamiento.
- Desarrollo propio: Los contribuyentes pueden desarrollar un software propio y homologarlo ante la DIAN.
- Proveedores tecnológicos: Son empresas avaladas por la DIAN para prestar el servicio de facturación electrónica, estos proveedores ofrecen una diversa cantidad de funcionalidades y costos.

El lenguaje InvoiceQL puede ser una herramienta muy útil para los proveedores tecnológicos y para el desarrollo de software propio.

El esquema de facturación en términos generales se muestra en la figura **2-2**.

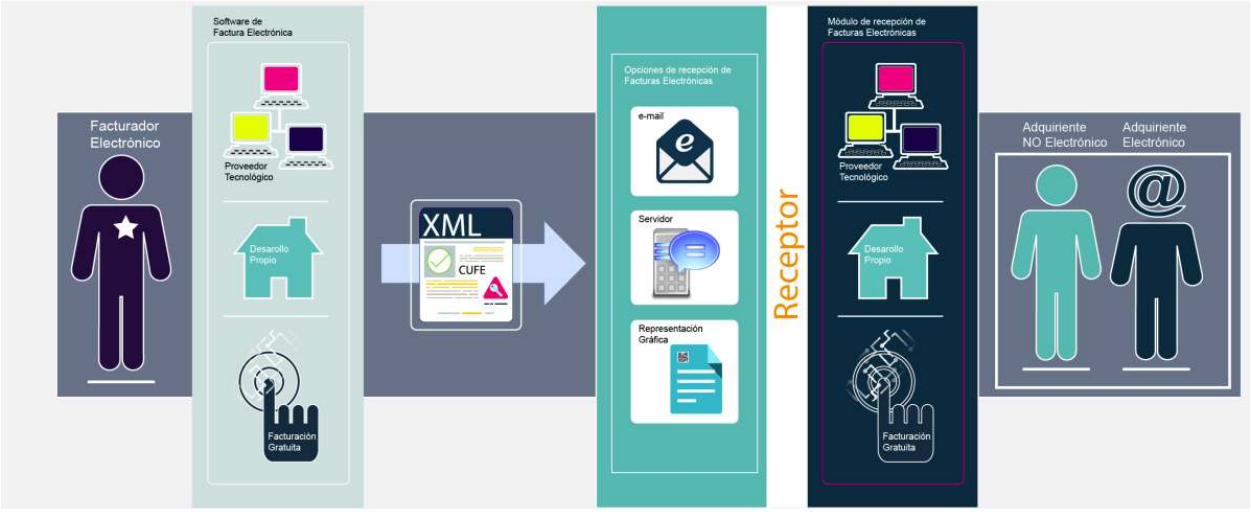

Figura 2-2.: Esquema de facturación electrónica en Colombia. Fuente: DIAN. [26]

Existen cuatro actores en este esquema:

- El facturador electrónico: Persona jurídica o natural que tiene la obligación de expedir la factura electrónica (es decir, el comercio).
- El medio de generación (son las tres alternativas mencionadas anteriormente)
- La DIAN: Que valida y acepta o rechaza el documento generado.
- El Adquiriente: Persona natural o jurídica que compra el producto o servicio.

Cuando un adquiriente realiza una transacción comercial el facturador electrónico debe expedir una factura en formato XML con especificaciones UBL 2.1 utilizando un software (de algún proveedor tecnológico, propio, o el gratuito de la DIAN) autorizado por la DIAN, luego el documento se envia a la DIAN y si es aceptado debe enviarse un correo electrónico al adquiriente con una copia del documento XML y una representación gráfica de la factura en formato PDF.

2.2. Lenguaje comercial universal UBL

UBL es una especificación de XML creada por la organización de estándares internacionales OASIS y que permite el intercambio de documentos comerciales en diferentes industrias [10]. UBL ofrece un lenguaje de negocio con componentes reutilizables para la gestión de diferentes documentos comerciales comúnmente utilizados (facturas, ordenes de pedido, ordenes de despacho, entre otros). UBL tiene una librería de esquemas XML para direcciones, productos, pagos y otros elementos comúnmente utilizados en transacciones comerciales.

Algunas ventajas que tiene el utilizar UBL son [10]:

- Bajo costo de integración debido a la reutilización de elementos comunes.
- Una curva de aprendizaje más fácil, porque los usuarios necesitan dominar solo una biblioteca.
- Capacitación estandarizada, lo que resulta en muchos trabajadores calificados.
- Herramientas de entrada y salida de datos estandarizadas y económicas.

En la ducumentación de UBL se aclar que los procesos comerciales y reglas asociadas que se proponen en el estándar son una semántica propuesta para lograr un efectivo intercambio de documentos y no una limitante para las aplicaciones que utilicen UBL.

En la figura **2-3** se muestra un diagrama de 'caso de uso' que ilustra los procesos comerciales que cubre UBL 2.1. Se observan los tres elementos principales: La compra, el envío, y el pago. 'La compra' es el elemento primordial del que se desprenden los demás componentes.

La factura electrónica reglamentada por la DIAN utiliza muchos de los elementos de UBL, así que la entidad en la resolución 000042 del 05 de mayo de 2020 establece como estándar para la generación, envío y recepción de documentos a UBL 2.1. Los documentos que se especifican para el proceso de facturación electrónica en Colombia son:

- Factura (Invoice)
- Nota Crédito (CreditNote)
- Nota Débito (DebitNote)
- Contenedor de documentos (AttachedDocument)
- Registro de eventos (ApplicationResponse)

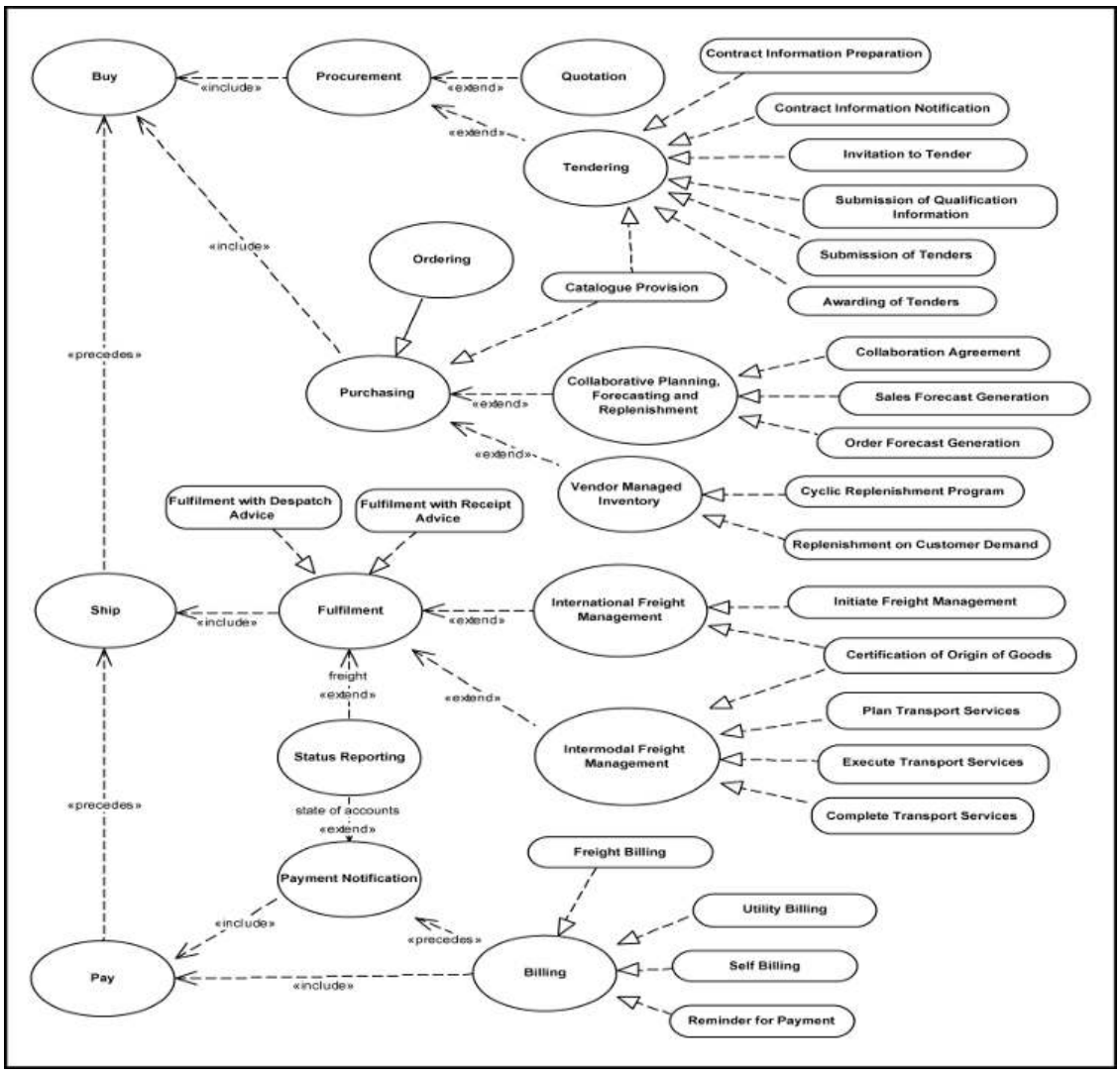

Figura 2-3.: Caso de uso de UBL 2.1. Fuente: OASIS. http://docs.oasis-open.org/ubl/os-UBL-2.1/UBL-2.1.html#S-INTRODUCTION

En UBL 2.1 se utilizan XSD (XML Schema Definition) en donde se definen los diferentes objetos que se van a utilizar. Puntualmente en los documentos XML los XSD se definen a través de los espacios de nombres (xmlns).

En la figura **2-4** tenemos un segmento de ejemplo de una factura en formato UBL en donde podemos ver la utilización de los XSD, por ejemplo el espacio de nombres (xmlns) con el prefijo 'cbc' contiene elementos como: <cbc:UUID/> para identificar el documento, <cbc:IssueDate/> con la fecha de expedición del documento, <cbc:InvoiceTypeCode/> para indicar el tipo de documento. La DIAN agrega y define sus propios XSD con elementos muy particulares necesarios para la facturación en Colombia.

Figura 2-4.: Ejemplo Factura UBL 2.1. Fuente: OASIS. http://docs.oasis-open.org/ubl/os-UBL-2.1/xml/UBL-Invoice-2.0-Example.xml

2.3. Lenguaje de dominio específico DSL

Los DSL (Domain Specific Language, por sus siglas en inglés) son lenguajes diseñados para satisfacer las necesidades de un dominio de aplicación específico [11], se centran en temas particulares y por lo tanto ayudan a los usuarios a concentrarse en dar solución a los problemas específicos dentro de un dominio en lugar de gastar tiempo en aprender conceptos (en ocasiones muy abstractos) de un lenguaje de propósito general (GPL, General proposit Language) que no son importantes para el trabajo que intentan realizar. Por ejemplo, el conocido lenguaje de dominio específico SQL (Structured Query Language) ayuda a un experto en base de datos a centrarse en gestionar la información de una base de datos sin que deba preocuparse por manipular archivos o gestionar directamente los recursos computacionales (memoria, procesador, almacenamiento) o diseñar algoritmos complejos para realizar consultas sobre los datos.

Los DSL proporcionan un alto nivel de abstracción para modelar problemas y soluciones en un dominio y por lo tanto su principal objetivo es ser una herramienta para que los expertos diseñen e implementen sistemas de una forma más sencilla que si lo hicieran con un GPL [12]. Algunas diferencias entre los DSL y los GPL se muestran en la siguiente tabla :

Tabla 2-1.: Diferencias entre DSLs y GPLs. Tomado de [13].

	GPLs	DSLs
Dominio	Grande y complejo	Pequeño y bien definido
Tamaño del lenguaje	Grande	Pequeño
Turing Completo	Siempre	No a menudo
Abstracciones definidas por el usuario	Sofisticadas	Limitadas
Ejecución	Nativa	Mediante GPL
Esperanza de vida	Años o Decadas	Meses a años
Diseñado por	Gurus y comités	Algunos ingenieros y expertos en el dominio
Comunidad de Usuarios	Grande, anónima y generalizada	Pequeña, accesible y local
Evolución	Lenta y menudo estandarizada	A un rápido ritmo
Depresación, Cambios para compatibilidad particular	Casi imposible	Factible

[1] Turing Completo, hace referencia a la máquina universal de Turing.

Dentro de los beneficios de utilizar un DSL están [13]:

- *Productividad*: Cuando se requiere automatizar una tarea de dominio utilizar un DSL implica menos líneas de código que utilizar un GPL.
- *Calidad*: Un DSL puede llevar a que se cometan menos errores pues pueden ser revisados facilmente por los expertos del dominio, son más fáciles de mantener, y además al quitar libertad (innecesaria) a los programadores se puede evitar el código duplicado.
- *Validación y verificación*: Debido a que la gramática de los DSL omite detalles técnicos de implementación hace que el código escrito sea fácilmente validado por los expertos en el dominio sin que estos deban ser expertos programadores.
- *Longevidad de los modelos*: El desarrollo de un DSL implica la construcción de modelos que tienen niveles de abstracción que pueden perdurar incluso si se cambia la tecnología del DSL, es decir, que el trabajo realizado para el desarrollo de los modelos puede ser reutilizado.
- *Herramientas para el pensamiento y la comunicación*: Cuando se tiene un lenguaje alineado con un dominio este permite que el pensamiento sea más claro, ya que ayuda a separar los detalles de la implementación de la complejidad esencial del problema que se quiere resolver. Los DSL ayudan también a alinear las diferencias que surgen en las personas que tratan de resolver un mismo problema de formas diferentes mejorando

así la comunicación de los equipos de trabajo (Dado que todos comprenden el mismo lenguaje).

- *Participación de los expertos del dominio*: Los DSL permiten una muy buena integración entre los desarrolladores con los expertos puesto que los DSL tienen abstracciones totalmente alineadas con los conocimientos del experto lo que permite incluso que estos escriban parte del código de los sistemas a desarrollar.
- *Herramientas productivas*:A los DSL se pueden asociar herramientas que ayudan a mejorar la experiencia del usuario con el lenguaje y permitirle realizar por ejemplo: análisis estáticos, vizualizaciones, estadísticas y simulaciones entre otras funcionalidades.
- *Independencia y aislamiento de las plataformas*: En ciertos casos se puede aislar la lógica de negocio escribiéndola con un DSL, esta lógica puede reutilizarse haciendo uso de herramientas independientementes de cualquier plataforma.

Pero ¿qué elementos necesitamos para crear y trabajar con un DSL?

- *Dominio*: El dominio brinda el contexto en el que será util el DSL, del dominio dependen las reglas, restricciones e incluso la sintaxis.
- *Gramática*: La gramática del lenguaje sigue las reglas del modelo del dominio, y además contiene los elementos léxicos y sintácticos del lenguaje.
- *Interprete*: El intérprete es un programa que entiende el DSL y genera acciones y salidas a partir de programas escritos en el lenguaje.
- *Herramientas de desarrollo*: Para poder trabajar adecuadamente con un DSL hace falta tener herramientas como por ejemplo un IDE(Integrated Development Environment) que facilite la escritura de los programas con funcionalidades de autocompletado, coloreado de sintaxis(highlighting), marcado de errores, hipervínculos, impresión de salidas, entre otras [14].

2.4. Desarrollo de software dirigido por modelos MDSD

El desarrollo de software dirigido por modelos MDD (Model-Driven Development por sus siglas en inglés) o MDSD (Model-Driven Software Development) es un paradigma que consiste en poner el análisis y los modelos del desarrollo de software al mismo nivel de implementación que el código fuente, con el objetivo de aumentar la velocidad del desarrollo y la realización de cambios, esto se logra a partir de la automatización, las transformaciones, y los modelos formales[15]. La idea es que los modelos hacen parte del código fuente no solo como herramientas utiles para el análisis, sino también como artefactos desde los que se genera y modifica código, por lo tanto MDSD es una estrategia de desarrollo que requiere un alto nivel de abstracción (Una de las formas más utilizadas para representar este alto nivel de abstracción son los metamodelos).

Otro término asociado a MDSD es MDA (Model-Driven Architecture, por sus siglas en inglés) o arquitectura dirigida por modelos que en términos generales es la visión de desarrollo de software dirigido por modelos de la importante organización OMG (The Object Management Group)[11]. Para la ORG MDA no es como tal un estandar, si no más bien un enfoque de desarrollo que utiliza especificaciones como UML (Unified Modeling Language), SysML, SoaML o CORBA.

Algunos de los objetivos de MDSD son[15]:

- Incrementar la velocidad del desarrollo de software.
- El uso de metamodelos, transformaciones, y automatización que ayudan a mejorar la calidad del software.
- Rápido cambio de características transversales y fácil corrección de errores.
- Alta reutilización del código.
- Mejorar la capacidad de gestión disminuyendo la complejidad por medio de la abstracción.
- Ayudar a la adopción de buenas prácticas de desarrollo de software.
- Y de acuerdo a la OMG (con respecto a MDA) el desarrollo dirigido por modelos mejora la interoperabilidad(independencia del fabricante por medio de la estandarización) y la portabilidad de los sistemas.

2.5. Desarrollo de funcionalidades dirigido por modelos dentro del contexto de MDSD

Los DSL en el contexto de MDSD son usados para construir modelos que permiten generar código fuente de software[30], sin embargo, como se mostró en la sección 2.3 este no es el único propósito que tienen los DSL. Se puede utilizar el enfoque MDSD para construir un DSL que resuelva un requerimiento puntual como por ejemplo: la generación de facturas electrónicas. Este enfoque de MDSD para generar artefactos finales de requerimientos y no sistemas de software lo llamamos **desarrollo de funcionalidades dirigido por modelos**.

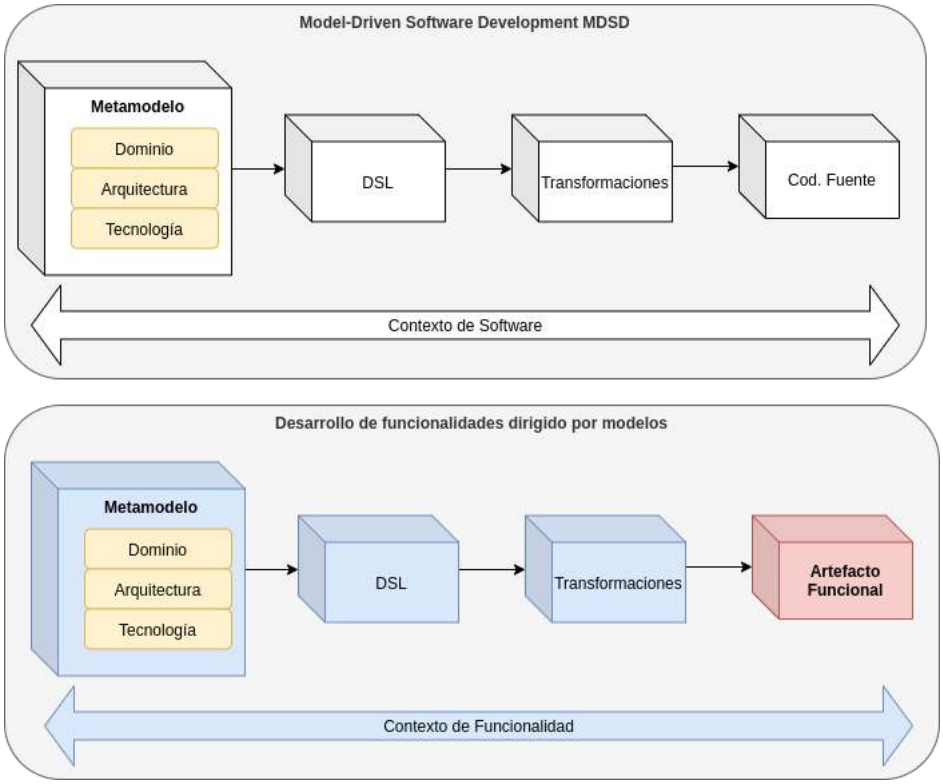

Figura 2-5.: Desarrollo de funcionalidades dirigido por modelos. Fuente: Imagen propia

En la parte superior de la figura **2-5** se muestran las fases de un proyecto con enfoque MDSD, estas fases se desarrollan teniendo en cuenta los elementos (dominio, arquitectura y tecnología) necesarios para que al final del proceso se genere código fuente que incluye archivos de configuración, código en algún lenguaje de propósito general (java, python, javascript, etc.), e incluso documentación dependiendo de lo definido en el metamodelo.

En la parte inferior de la figura **2-5** se muestran las mismas fases típicas de un proyecto con paradigma MDSD, sin embargo, el punto de vista que se tiene en cuenta es *la funcionalidad* por lo que al final del proceso se obtendrán los artefactos puntuales definidos por los reque-

rimientos. Por ejemplo, si el requerimiento es 'diseñar una solución informática que permita generar boletos de autobus' el DSL no se enfocará en generar el código fuente de un software que genere boletos de autobus, sino que el DSL por medio de las transformaciones generará directamente el boleto.

El DSL InvoiceQL se construirá teniendo en cuenta el contexto de funcionalidad como se muestra en la parte inferior de la figura **2-5**.

2.6. Sistema de planificación de recursos empresariales ERP / ODOO

Parte de este trabajo consiste en mostrar como puede usarse invoiceQL en un sistema de información utilizado en entornos productivos. ODOO es un conocido ERP (Enterprise Resource Planning, por sus siglas en inglés) de código abierto que ayuda a las organizaciones a gestionar muchos de sus procesos empresariales. ODOO tiene una versión de pago y otra que es libre que es soportada por una comunidad. Este ERP tiene gran cantidad de módulos algunos de los cuales son de uso gratuito, algunos de estos son [16]:

- Gestión de inventario.
- Gestión de recursos humanos.
- Gestión de proyectos.
- CRM. (Customer Relationship Management)
- Puntos de Venta PoS. (Point of Sale)
- Módulo contable.
- Gestión de activos.
- Gestión de almacenes.

Odoo está desarrollado con Python y tiene una base de datos en PostgreSQL, la última versión estable es la 13.0 lanzada el 2 de octubre de 2019. Cuenta con amplia documentación para la creación de nuevos módulos y con un API (Application programming interface) para integrarse con sistemas externos.

La arquitectura de Odoo es cliente-servidor y cuenta con una interfaz web desde la que se gestiona todo el sistema. Odoo funciona en plataformas windows y linux con diferentes opciones para su instalación:

- Instalación desde código fuente de repositorio.
- Imagen docker.
- Archivo binario de instalación.

2.7. Tabajos Relacionados

Como parte de la revisión bibliográfica se identificaron diferentes trabajos relacionados los cuales se muestran a continuación y se agrupan por tema.

Facturación electrónica.

Como se mostró en la sección 2.1 son muchos los países que ya implementaron la facturación electrónica o están en el proceso por lo tanto hay diversas opciones para la generación, envío, y validación de estos documentos. En el muy completo panorama de la facturación electrónica mostrado por [8] se mencionan algunas empresas multinacionales que ofrecen diferentes soluciones de facturación electrónica. Por ejemplo:

- *Business to business e-Solutions (B2BE):*Con presencia en Australia y Europa ofrece una plataforma que ha procesado más de 75 millones de facturas, brinda personalización y herramientas de integración bajo un modelo de negocio B2B (Business to business).
- *Bizbox*: Con presencia en la Unión Europea presenta un interesante modelo de facturación electrónica y certificación de facturas (mediante cifrado). La plataforma procesa más de 80 millones de documentos por año.
- *CisBox*: Con presencia en Suiza, Alemania y Austria y mas de 125.000 transacciones por año ofrece un modelo BPaaS (Business-Process-as-a-Service) o proceso de negocio como servicio que se trata de un servicio en la nube que además de gestionar la factura electrónica también gestiona el pago del producto o servicio.
- *Crossinx*: Empresa que opera en Suiza, Austria y Alemania tiene la particularidad que utiliza la tecnología Blockchain para garantizar la autenticidad de los documentos procesados.
- *Datamolino*: Empresa Europea que ofrece una plataforma en un modelo SaaS(sofware como servicio) y cuenta además con un API para integración con sistemas externos.
- *Digital Planet*: Esta empresa Turca que procesa más de 1 millón de facturas por año tiene una solución para la gestión de 'factoring'(Que es el término como se le conoce al proceso de compra y venta legal de facturación o cartera)

- *Digital Technologies Srl*: Empresa que opera principalmente en Italia y España como valor agregado presenta una solución que incluye el uso de robótica y tecnología IoT (Internet of things) para agilizar la generación de la factura electrónica y también para digitalizar las facturas que están en papel.
- *GoSocket*:Con presencia en latinoamérica (Argentina, Bolivia, Brasil, Chile, Colombia, Costa Rica, Ecuador, Guatemala, México, Paraguay, Perú y Uruguay) procesa mas de 400 millones de facturas electrónicas por año con un modelo B2B ofrece entre otras cosas intercambio de facturas (factoring) entre negocios compatibles.
- *Indicium Solutions, S.A. de C.V.*: Con operaciones en México, Chile, Colombia y Argentina ofrece inegración con cualquier ERP o aplicación Ad-Hoc que requiera facturación electrónica.
- *Netsend Ltd* : Opera en países de la Unión Europea y Estados Unidos de América ofrece una solución de e-invoicing que se integra directamente con franquicias de tarjeta crédito.

Cómo se puede observar en las empresas mencionadas anteriormente hay diferencias entre las soluciones de los países donde el proceso de facturación electrónica es más maduro (Norte de Europa principalmente) con aquellos que apenas empiezan. En los países con 'Facturación electrónica madura' las soluciones son más sofisticadas y hacen uso de tecnologías como Blockchain, Inteligencia artificial, o IoT. En laltinoamérica donde en algunos países como Colombia recién se empieza a implementar la Facturación electrónica las soluciones son más básicas y cuentan con modelos de negocio B2B, SaaS, y con herramientas para Factoring además de APIs para la integración de diferentes sistemas de información.

Las soluciones de facturación electrónica para PyMES (Pequeñas y medianas empresas) por lo general tienen la estructura que se muestra en la figura **2-6** [17].
Los actores del proceso son: un sistema e-invoicing en la nube, la agencia de impuestos (en Colombia la DIAN), el cliente (para la DIAN el adquiriente) y la empresa que factura (para la DIAN el facturador electrónico).

La agencia de impuestos se comunica con el sistema e-invocing por medio de protocolo de servicios SOAP (por medio de este se envía y recibe la factura electrónica en formato UBL, en el caso de Colombia), el cliente es notificado de la operación por correo electrónico (protocolo SMTP) y la empresa que factura interactúa con el sistema e-invocing por lo general por HTTP a través de un navegador (o Browser). El sistema e-invocing tambien puede contar con una API para comunicarse con sistemas propios del facturador electrónico.

En cuanto al sistema e-invocing suele funcionar como un servicio en la nube (SaaS). Con respecto a la arquitectura de software un ejemplo típico se muestra en la figura **2-6** que

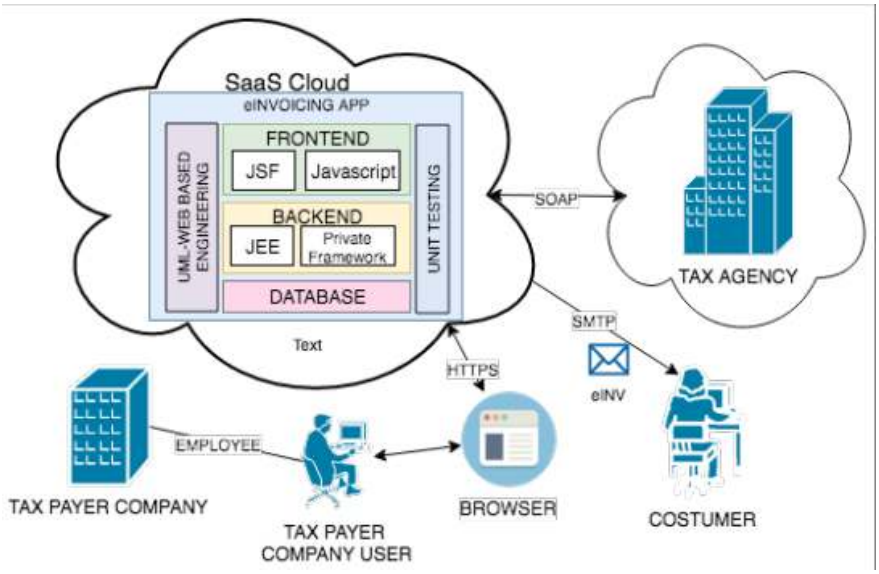

Figura 2-6.: Ejemplo de una solución típica de facturación electrónica para PyMES. Fuente: Matus(2017)[17]

consiste en una aplicación web con arquitectura Java JEE y tres capas(Frontend, Backend, y Datos).

El esquema descrito anteriormente ejemplifica la facturación electrónica que será implementada en Colombia, sin embargo, existen otros modelos, protocolos, y formatos de factura electrónica.

En cuanto a protocolos de comunicación con el cliente además de SMTP también se utilizan: FTP, sFTP, x.400, o AS2; Además de XML otra alternativa es EDIFACT; Y además de UBL otros estándares utilizados son: ISO20022, GS1-XML, SAP-iDoc, Finvoice y OIOXML[18].

ERPL - Lenguaje de dominio específico para ERPs (Enterprise Resource Planning)

Los ERP (ya sean de proveedores como: SAP, Microsoft Dynamics, ODOO, entre otros; o de desarrollo propio) son sistemas que ayudan mucho las organizaciones a gestionar sus procesos de negocio, sin embargo, estos presentan también algunas desventajas [19]:

- Son sistemas muy complejos.
- Pueden resultar costosos para organizaciones de pequeño y mediano tamaño.
- Para desarrollar un ERP se necesita la asistencia de expertos en diferentes dominios.

- Para que los trabajadores de una organización utilicen el ERP se requiere de una costosa capacitación.
- A manudo son costos de mantener, en términos de desarrollo.

Así que en la *conferencia internacional de tendencias emergentes en las tecnologías de la información (ic-ETITE)* [19] se propone el desarrollo de un DSL llamado ERPL que ayude a solventar muchos de los problemas mencionados anteriormente (la argumentación de porque el lenguaje ERPL puede ayudar se muestran en [19]). En la figura **2-7** se muestra la arquitectura de un sistema interprete de ERPL que contiene 4 componentes: Una base de datos, una aplicación de servidor, unos roles de acceso sobre la aplicación que gestiona la base de datos y unos 'script' de entrada (escritos en lenguaje ERPL) con instrucciones de negocio que son interpretadas por la aplicación en el servidor.

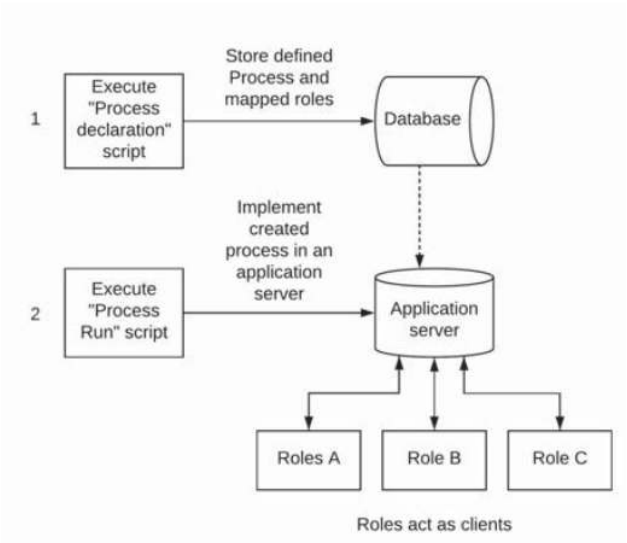

Figura 2-7.: Arquitectura de ERPL. Fuente: Sharma(2020)[19]

MDSD y Ontologías para los requerimientos

Una interesante propuesta para el desarrollo de sistemas de información (como los ERP) es aplicar el paradigma de desarrollo de software dirigido por modelos (MDSD) pues tiene múltiples ventajas como las mostradas en la sección 2.4 y utilizar modelos basados en ontologías que representen los requerimientos [20]. Uno de estos requerimientos puede ser por ejemplo la necesidad de generar facturas electrónicas basadas en un estándar como UBL.

En la figura **2-8** se muestra un esquema conceptual con la idea. Cuando el cliente define los requeriminientos estos se representan como una ontología (mediante diagrama OWL), después este modelo se abstrae en el metamaodelo que se utiliza bajo el marco de MDSD de

esta forma el requerimiento formará parte del desarrollo de forma directa. La salida de este sistema será el un código fuente que se genera por medio de transformaciones.

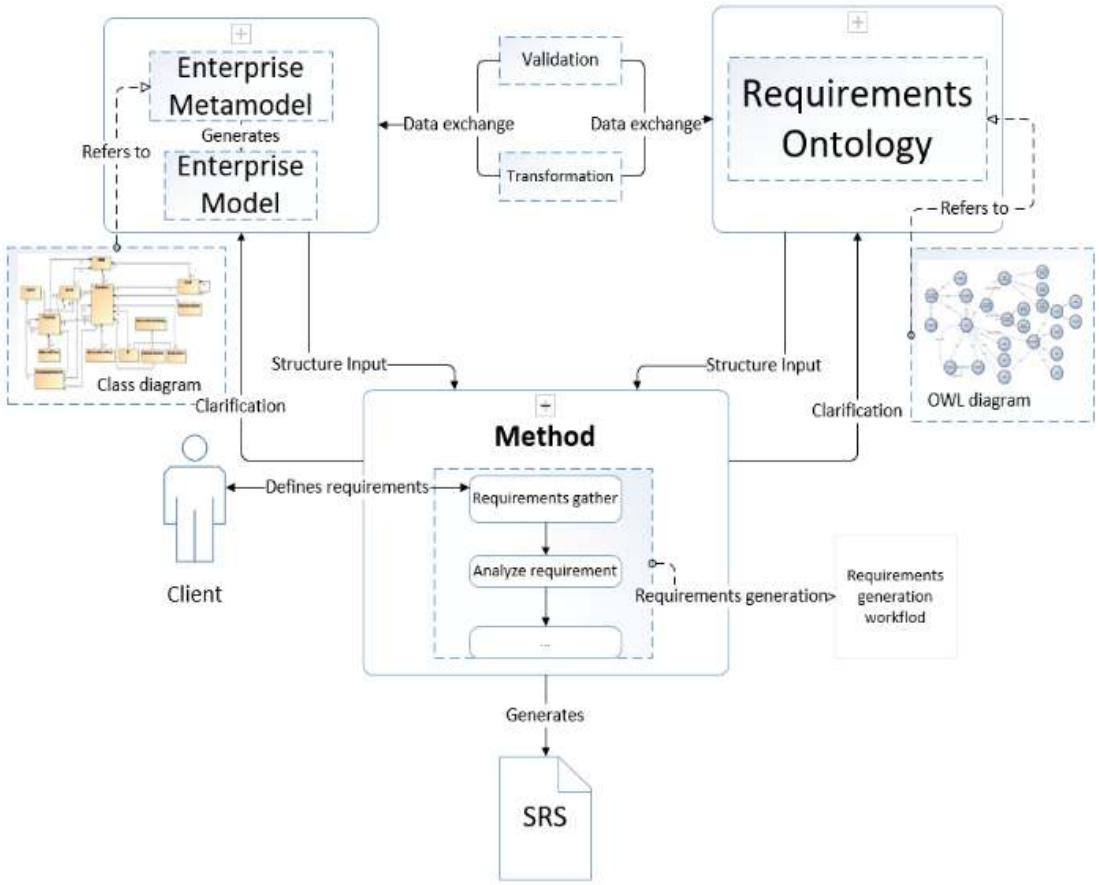

Figura 2-8.: Esquema conceptual de MDSD con ontologías para los requerimientos. Fuente: Makrickiene(2019)[20]

Celonis PQL, un DSL para la minería de datos

Una interesante propuesta para aprovechar la información de documentos como las facturas electrónicas es Celonis PQL que es un lenguaje de dominio específico que se especializa en descubrir, mejorar y monitorear procesos de negocio mediante la minería de datos[21]. Las herramientas asociadas a este DSL permiten que se integre con sistemas ERP y CRM de los cuales extrae la información requerida para hacer minería de datos. En la figura **2-9** se muestra la arquitectura del sistema que implementa este DSL. Como entrada de datos tiene integrados 'sistemas fuentes de datos', cuenta con un núcleo en el que se analiza la información se procesan las consultas en lenguaje PQL que las aplicaciones cliente solicitan.

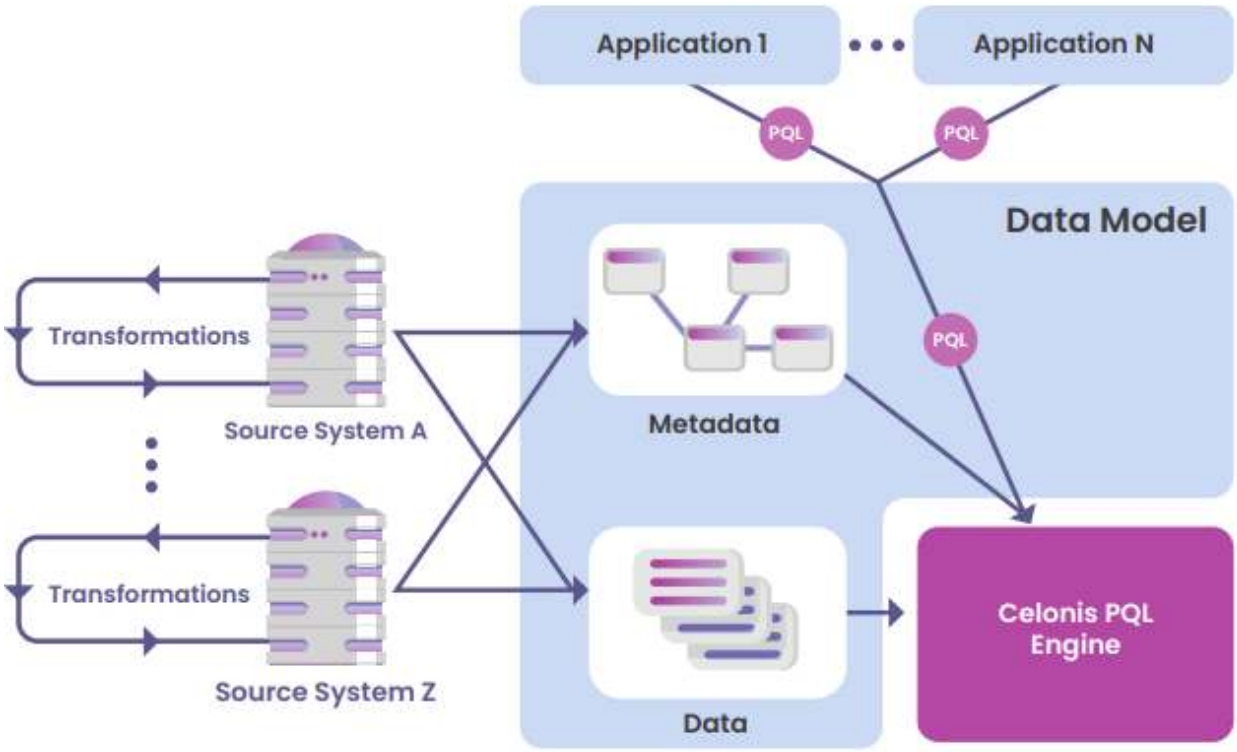

Figura 2-9.: Arquitectura de Celonis PQL. Fuente: Vogelgesang(2019)[21]

2.8. Metodología

La metodología para el desarrollo de *invoiceQL* se basa en los trabajos de Vergara(2018)[22], Castelblanco(2014)[23], y Neeraj(2017)[24]. Esta metodología tiene un enfoque de MDSD (Model-Driven software development) y consta de 8 fases como se muestra en la figura **2-10**. En cada una de estas fases siempre se tienen en cuenta tres aspectos: *El dominio* del problema, *la arquitectura* de la solución y *la tecnología* que se utilizará.

El enfoque de esta metodología estará en la funcionalidad, es decir, en generar la factura electrónica y no en generar un software que la genere por lo que todos los elementos de arquitectura, tecnología y dominio a tener en cuenta estarán centrados en la factura electrónica, lo que anteriormente nombrábamos como *desarrollo de funcionalidades dirigido por modelos* MDFD.

1. Implementación de referencia.

La implementación de referencia es un prototipo 'hecho a mano' del documento que se quiere obtener al final del proceso. En este caso la implementación de referencia es una factura

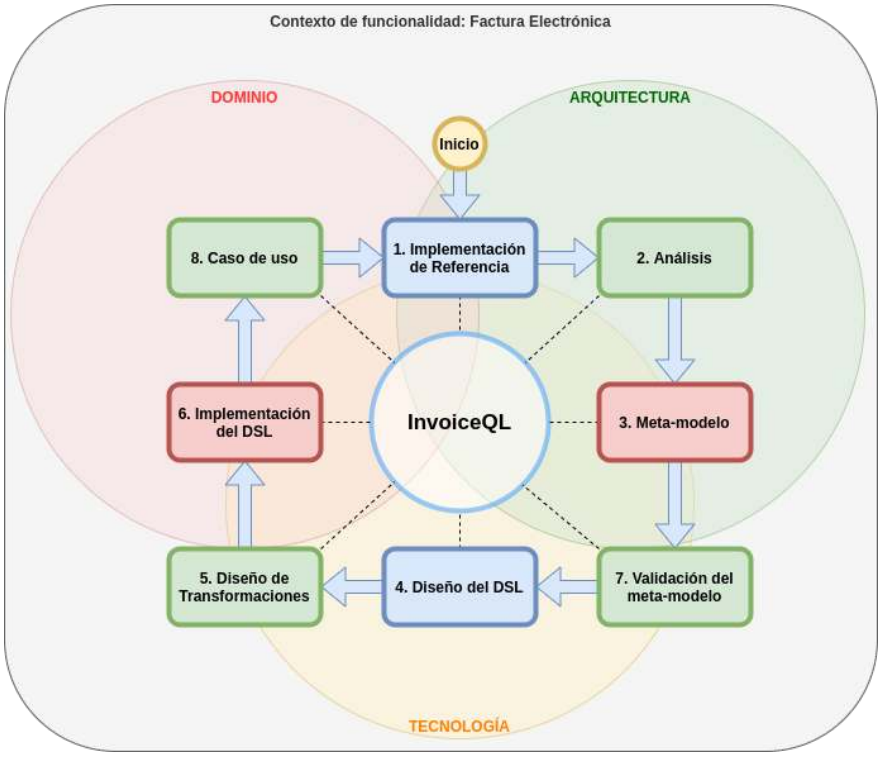

Figura 2-10.: Metodología para el desarrollo de InvoiceQL basada en MDSD. Fuente: Imagen propia

electrónica en formato XML que sigue el estándar UBL 2.1. Esta factura electrónica de referencia es un ejemplo de documento electrónico válido que hace parte de la documentación técnica que la DIAN suministra.

2. Análisis.

Se analizan cada uno de los componentes de la implementación de referencia desde los puntos de vista de: el dominio, la arquitectura y la tecnología. Para este trabajo el dominio son todas las restricciones y normas técnicas que impone la DIAN, la arquitectura es la descrita por el estándar UBL 2.1 y la tecnología que se utiliza es XML. Esta fase es muy importante pues de este análisis depende el siguiente nivel de abstracción, el meta-modelo.

3. El metamodelo.

El metamodelo es un diagrama de clases en UML que describe los componentes y relaciones necesarias para construir los modelos que resultaron de la fase de implementación y análisis. En el metamodelo hay tres ramas de forma explícita: Dominio, arquitectura y tecnología.

4. Validación del meta-modelo.

En esta etapa se realizan pruebas con la herramienta desarrolla en la fase anterior generando código y comparando los resultados con el metamodelo.

5. Diseño del DSL.
En esta fase se define la gramática del lenguaje de dominio específico (DSL) con base en los componentes y relaciones descritas en el metamodelo. Con la ayuda de árboles de sintaxis abstracta (o AST por sus siglas en inglés) y de la notación Bacus-Naur Form (BNF) se diseñan las diferentes reglas gramaticales que tendrá el lenguaje.

6. Diseño de las transformaciones.

En esta etapa se diseña la estrategia para la generación de código a partir de las reglas gramaticales definidas en la fase anterior. Para el caso de invoiceQL la estrategia se basa en una doble transformación. Del DLS a JSON y de JSON a XML. Las razones por las cuales se realiza esta doble transformación se explican en la siguiente sección.

7. Implementación del DSL.

Con la ayuda de el paquete de herramientas de Eplipse EMF(Eclipse model Framework) se implementa el DSL. Por una parte se escribe la gramática del DSL con el lenguaje XText y por otro lado se implementan las transformaciones previamente dieñadas con XTend. Como valor agregado EMF permite generar un IDE con base en eclipse para el nuevo DSL con funcionalidades de reconocimiento de sintaxis y autocompletado.

8. Caso de uso.

El DSL se prueba en un caso práctico de donde se puede obtener realimentación para mejorar el modelo y corregir posibles errores. Para el caso de invoiceQL se hace una prueba integrando el DSL al ERP de código abierto ODOO.

Como vemos en la figura **2-10** esta es una metodología es cíclica por lo que el modelo se puede mejorar con cada iteración.

3. Desarrollo del lenguaje InvoiceQL

3.1. Implementación de referencia

La factura electrónica es un documento XML que sigue el estándar UBL 2.1, la DIAN suministra ejemplos de este documento junto con la documentación técnica, así que se toma como *implementación de referencia* uno de estos ejemplos para comenzar con el desarrollo de este trabajo. La resolución de la DIAN No. 000042 del 05 de mayo de 2020 cuenta con un anexo técnico en el que se describen cada uno de los campos y componentes de la factura electrónica.

De acuerdo con la resolución 000042 (Artículo 23) "La generación de la factura electrónica de venta, las notas débito, notas crédito e instrumentos electrónicos que se derivan de la factura electrónica de venta, se deben elaborar cumpliendo con las condiciones, términos y mecanismos técnicos y tecnológicos, de conformidad con el «Anexo técnico de factura electrónica de venta»"[25], por este motivo es muy importante para la implementación de referencia seguir las indicaciones del citado anexo técnico.

3.1.1. Estructura de la factura electrónica reglamentada por la DIAN.

Haciendo una revisión detallada de la documentación técnica que suministra la DIAN se observa que el documento 'factura electrónica(invoice)' tiene 470 campos y se agrupan como se muestra en la tabla **3-1**(33 grupos de campos).

Grupo	**No.C.**	**Descripción**
FAA	2	Campos de definición de espacios de nombres.
FAB	36	Información relacionada con la generación del documento, resolución de facturación, datos del software, proveedor, etc.
FAC	3	Campos relacionados con la firma digital del documento.
FAD	15	Información general de la factura, divisa, fecha de expedición.

FAE	5	Grupo de campos relativos al Periodo de Facturación: Intervalo de fechas la las que referencia la factura por ejemplo en servicios públicos.
FAF	3	Grupo de campos para información que describen una exclusiva orden para esta factura.
FAG	3	Fecha de emisión: Fecha de emisión del documento de despacho.
FAH	3	Grupo de campos para información que describen uno o más documentos de despacho para esta factura.
FAI	4	Grupo de campos para información que describen un documento referenciado por esta factura,
FAJ	74	Información tributaria de la transacción.
FAK	63	Información relacionada con el adquiriente.
FAL	7	Grupo de información de la Persona autorizada para descargar documentos.
FAM	70	Información relacionada con el transporte de bienes y mercancías.
FAN	6	Grupo de campos para información relacionadas con el pago de la factura.
FAQ	10	Grupo de campos para información relacionadas con un cargo o un descuento.
FAR	7	Grupo de campos para información relacionadas con la tasa de cambio de moneda extranjera a peso colombiano (COP).
FAS	17	Grupo de campos para información totales relacionadas con un tributo.
FAT	13	Grupo de campos para información totales relacionas con los tributos retenidos.
FAU	15	Grupo de campos para información relacionadas con los valores totales aplicables a la factura.
FAV	7	Grupo de campos para información relacionadas con una línea de factura (Items de venta).
FAW	5	Grupo de información que indica el precio de referencia para línea que no contienen valor comercial.
FAX	17	Grupo de información específicas sobre cada tributo.
FAY	13	Grupo de información para tributos retenidos a nivel de línea de factura.
FAZ	14	Grupo de datos de identificación del artículo o servicio.

FBA	9	Grupo de información que describen el Mandante de la operación de venta. Aplica solo para mandatos, y se debe informar a nivel de ítem.
FBB	5	Grupo de información que describen los precios del artículo o servicio.
FBC	5	Grupo para información relacionadas con la entrega.
FBD	8	Grupo de campos para información relacionadas con un anticipo.
FBE	9	Grupo de campos para información relacionadas con un cargo o un descuento.
FBF	3	Grupo de información para adicionar información específica del ítem que puede ser solicitada por autoridades o entidades diferentes a la DIAN.
FBH	6	Grupo de información exclusivo para referenciar la Nota Crédito que dio origen a la presente Factura Electrónica.
FBI	6	Grupo de información exclusivo para referenciar la Nota Débito que dio origen a la presente Factura Electrónica.
FGB	7	Grupo de campos utilizado como método alternativo para infomar conversiones a otras divisas.

Tabla 3-1.: Grupos de campos de la factura electrónica reglamentada por la DIAN. Fuente: Propia.

Nota: No. C. : Numero de campos.

3.1.2. Elementos de UBL 2.1 que son utilizados en la factura electrónica de la DIAN.

Como se ve en la figura **3-1** los diferentes documentos electrónicos definidos por UBL 2.1 (Invoice, Order, etc.) utilizan diversas librerías de componentes XML los cuales son instanciados mediante prefijos que hacen referencia a espacios de nombres (o namespaces) que definen los esquemas XSD((XML Schema Definition) a utilizar. Tanto OASIS como la DIAN definen XSD que se utilizan en la factura electrónica.

En la figura **3-1** también se pueden observar las relaciones entre las librerías (referencias, generalidades, inclusiones e importaciones).

La factura electrónica definida por la DIAN además de las librerías de UBL utiliza librerías propias y de la W3C. En la tabla **3-2** se muestran las librerías utilizada en la factura electrónica.

Prefijo	Libreria	Descripción	Fuente
cac	Common Aggregate Components	Contiene componentes para la información básica del documento (fecha, números de referencia, direcciones teléfonos)	UBL 2.1
cbc	Common Basic Components	Define componentes básicos que son utilizados por CAC	UBL 2.1
ds	Xmldsig	Componentes requeridos para la firma digital	W3C
ext	Common Extension Components	Cualquier elemento en cualquier espacio de nombres que no sea la extensión UBL	UBL 2.1
sts	Componentes DIAN	Son los componentes propios definidos por la DIAN, por ejemplo para la información de las resoluciones de facturación.	DIAN

Tabla 3-2.: Librerias utilizadas para generar la factura electrónica. Fuente: Propia.

3.1.3. Restricciones de la DIAN.

Además de las restricciones de tipos de dato, obligatoriedad y cardinalidad, algunos de los campos de la factura electrónica definida por la DIAN tiene restricciones en sus posibles valores. Algunos son:

- Código del ambiente (Campo FAD04) : Producción/Pruebas
- CUFE (Código único de facturación electrónica - FAD06): Según fórmula explicada en el anexo técnico de la resolución 000043.
- Tipo de factura (FAD12): Factura electrónica de Venta, factura de exportación, o documento electrónico de transmisión.
- Divisa de la Factura (FAD15): Corresponde al estándar ISO de tres letras para las divisas. Ejemplo: EUR(Euro), MXV(Peso mexicano), COP (Peso Colombiano), etc.
- CUDE (Código de nota crédito relacionada / FBH05): Según fórmula del anexo técnico.

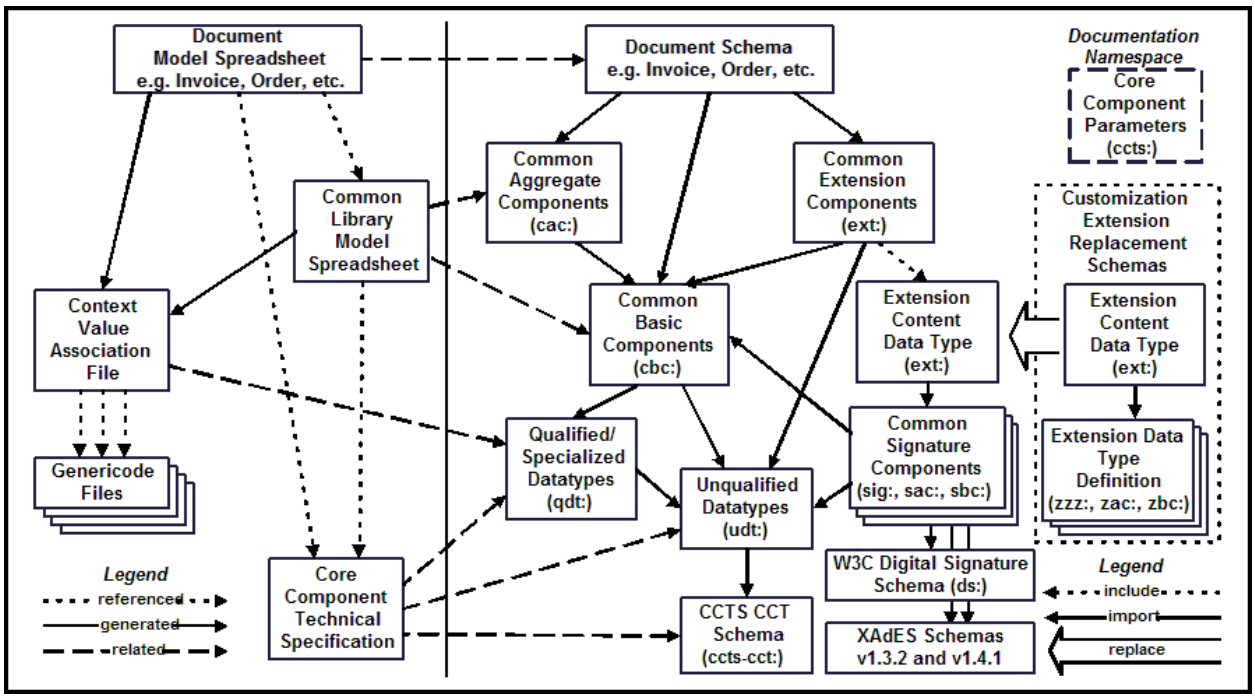

Figura 3-1.: Modelo de librerias de UBL 2.1. Fuente: OASIS.

- Tipo de Documento (FBI05): Factura, nota crédito o nota débito.
- Tipo de documento interno (FAI06): Remisión o referencias internas.
- Tipo de organización (FAJ02): Persona natural o jurídica.
- Código de municipio (FAJ09): Código de municipios nacionales según el DANE.
- Nombres de ciudades (Varios campos): De acuerdo a los datos del DANE (Para territorios nacionales).
- Código postal (FAJ10): Códigos suministrados por el DANE.
- Código de identificación del país (FAJ16): Código de país según normalización ISO 3166.
- Tipo de identificador fiscal (FAJ25): Cédula, registro civil, NIT, NUIP, pasaporte, etc.
- Obligaciones o responsabilidades del contribuyente (FAJ26): Retención en la fuente a título de renta, Retención timbre nacional, Gran contribuyente, Autorretenedor, etc.
- Identificador del lenguaje utilizado en el nombre del país (FAJ38) : Según la norma ISO 639.
- Identificador del tributo (FAJ40): IVA, IC, ICA, ReteIVA, ReteFuente, entre otros.

- Obligaciones del Participante del Consorcio (FAJ62): Retención en la fuente a título de renta, Retención timbre nacional, Informante de exógena, entre otros.
- Régimen al que pertenece el emisor del consorcio (FAJ63): Responsable del impuesto sobre las ventas –IVA o No responsable de IVA.
- Identificador de tipo de persona (FAK02): Persona Jurídica o Persona Natural
- Identificador del tributo del adquirente (FAK40): IVA, Timbre, ReteICA, ICA, entre otros.
- Obligaciones del transportardor (FAM37): Retención timbre nacional, Impuesto Nacional a la Gasolina y al ACPM, Impuesto Nacional al consumo, Agente marítimo, Agente de carga internacional entre otros.
- Condiciones de Entrega (FBC04): Costo y flete, Transporte Pagado Hasta, Franco transportista, Franco a bordo, Entregado en Terminal, Entregado en un Lugar, entre otros.
- Método de pago (FAN02): Instrumento no definido, Crédito ACH, Débito ACH, Efectivo, Crédito Ahorro, Cheque, Transferencia Crédito, Transferencia Débito, etc.
- Código para categorizar el descuento (FAQ04): Descuento por impuesto asumido, Envío gratis, Descuentos contractuales, Pague uno lleve otro, Descuento por volumen, entre otros.
- Código de moneda (FAQ08) : Según el estándar internacional ISO 4217.
- Tarifa del tributo (FAS14) : Exento(0 %), Bienes / Servicios al 5 (5 %), Contratos firmados con el estado antes de ley 1819 (16 %), Tarifa general (19 %), Tarifa especial(2 %), entre otros.
- Código del tipo de precio informado (FAW05) : Valor comercial, Valor en inventarios, entre otros.
- Código del estándar (FAZ12) : UNSPSC, GTIN, Partida Arancelarias, Estándar de adopción del contribuyente, entre otros.

En muchos de los campos se repiten las restricciones, por ejemplo: En los campos FAY06(Código de moneda de la transacción del tributo) y FAX08 (Código de moneda de la transacción) lo único que cambia es la entidad.

Otra restricción se presenta en las cifras decimales de las transacciones. En el anexo técnico suministrado por la DIAN se especifica la forma de redondear la cifras decimales.

3.2. Análisis

En esta sección con base en la implementación de referencia identificaremos cada uno de los componentes del documento XML que representa la factura electrónica.

3.2.1. Dominio.

El dominio se basa en el anexo técnico de la resolución No.000042 del 5 de mayo de 2020 de la DIAN, en donde se encuentran las diferentes especificaciones de la factura electrónica, como:

- **Identificador del campo**: Código único con el que se identifica cada campo. Ej. FAD12 - Tipo de factura.
- **Tipos de elemento**: Los campos de la factura pueden ser: Grupos de elementos (G), un elemento (E), o un atributo de un elemento (A).
- **Tipos de dato**: Los tipos de dato de los campos pueden ser: Alfanumérico(A), booleano(B), fecha(F), hora(H), intervalo de tiempo(I), y documento XML (X).
- **Funciones**: La DIAN establece unas instrucciones para redondear las cifras, además para calcular el CUFE(Código único de facturación electrónica), función para generar un código QR y la representación gráfica que debe llegar al adquiriente.
- **Cardinalidad**: Identifica la cantidad de posibles ocurrencias del elemento o grupo. El anexo técnico dice que:
  - 1..1 – Identifica que el elemento o grupo es obligatorio, con máximo de una ocurrencia.
  - 0..1 – Identifica que el elemento o grupo es facultativo (posible de no ser informado), con máximo de una ocurrencia.
  - 1..N – Identifica que el elemento o grupo es obligatorio, con máximo de N ocurrencias.
  - 0..N – Identifica que el elemento o grupo es facultativo (posible de no ser informado), con máximo de N ocurrencias.
- **Restricciones de valor**: Algunos campos de la factura tienen restricciones en los valores que pueden tener como se muestra en la sección 3.1.3.

3.2.2. Arquitectura.

La arquitectura del documento electrónico XML se define en UBL 2.1 y en el caso particular de la factura tiene los siguientes elementos:

Tipo de documento

Dependiendo del tipo de documento electrónico se utilizan las diferentes librerías de UBL o propias. Recordemos que la DIAN define 5 documentos electrónicos (sin embargo, en este trabajo solo se abordará la factura):

- Factura (Invoice)
- Nota Crédito (CreditNote)
- Nota Débito (DebitNote)
- Contenedor de documentos (AttachedDocument)
- Registro de eventos (ApplicationResponse)

Librerías

Dentro del documento XML que representa la factura electrónica (invoice) se utilizan espacios de nombre (o en inglés NameSpace ns) en los que se especifican las librerías de componentes que se van a utilizar. Estos 'namespaces' de componentes se describen mediante documentos XSD (XML Schema Definition) suministrados por UBL 2.1 y por la DIAN. Para la factura electrónica se utilizan las librerías que se muestran en la tabla **3-3**.

NS	Libreria
cbc	urn:oasis:names:specification:ubl:schema:xsd: CommonBasicComponents-2
cac	urn:oasis:names:specification:ubl:schema:xsd: CommonAggregateComponents-2
ext	urn:oasis:names:specification:ubl:schema:xsd: CommonExtensionComponents-2
sts	dian:gov:co:facturaelectronica:Structures-2-1
xades	http://uri.etsi.org/01903/v1.3.2#
xmlns	xades141="http://uri.etsi.org/01903/v1.4.1#

ds	http://www.w3.org/2000/09/xmldsig#

Tabla 3-3.: 'NameSpace' de la factura electrónica. Fuente: Propia.

3.2.3. Tecnología.

La tecnología a analizar para el diseño del metamodelo es XML. La W3C define a XML (Extensible Markup Language) como un lenguaje para describir y almacenar estructuras lógicas [27]. La estructura de los documentos XML se muestran en la figura **3-3**. El documento parte de un nodo principal (en HTML es el objeto DOM) que puede contener: otros elementos, atributos, comentarios y contenido. En la figura **3-2** se muestra un ejemplo identificando cada uno de los elementos de un documento XML[28].

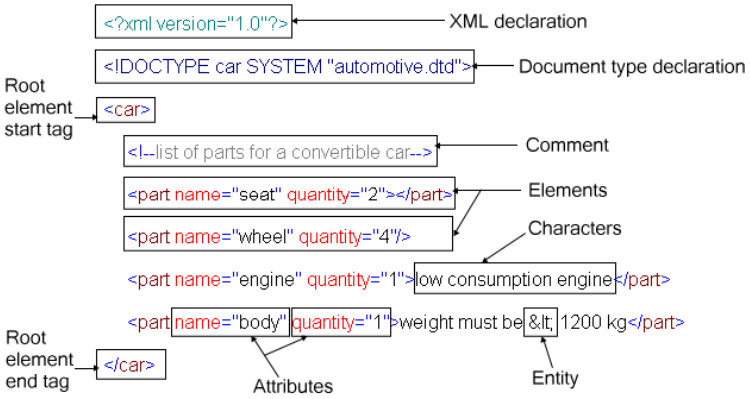

Figura 3-2.: Ejemplo de un documento XML identificando sus componentes. Fuente: https: //www.maruf.ca/files/caadoc/ CAAXmlTechArticles/CAAXmlIntroduction.htm

3.3. Metamodelo

El metamodelo es un diagrama de clases que representa cada uno de los elementos de la factura electrónica y sus relaciones. Hay tres ramas como se muestra en la figura **3-4**. Dominio, arquitectura y tecnología.

3.3.1. Rama de Dominio

En la rama de dominio del metamodelo hay 8 clases que representan componentes de la factura electrónica (Como se muestra en la figura **3-5**).

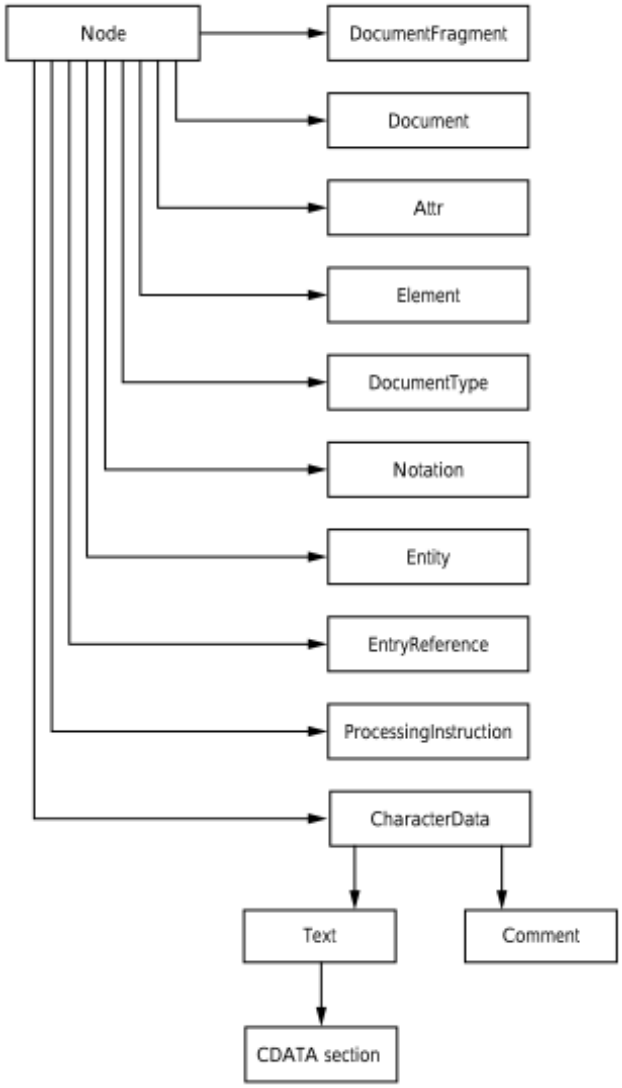

Figura 3-3.: Estructura de un documento XML. Fuente: Wood 1999 [28].

- **Document**: Representa los diferentes tipos de documento que pueden existir en el proceso de facturación electrónica (Invoice, CreditNote, DebitNote, AttachedDocument, ApplicationResponse).

- **DocumentAttribute** : Representa los diferentes campos que puede tener un documento (De acuerdo al manual técnico suministrado por la DIAN). *DocumentAttribute* tiene una relación fuerte de composición con *Document*. Esta clase tiene los atributos:

  - *Name*: Nombre que le da la DIAN al campo. Ej. Prefix.
  - *Id*: Identificador que le da la DIAN al campo. Ej. FAB10.
  - *Nullable*: Idica si el campo es o no obligatorio.

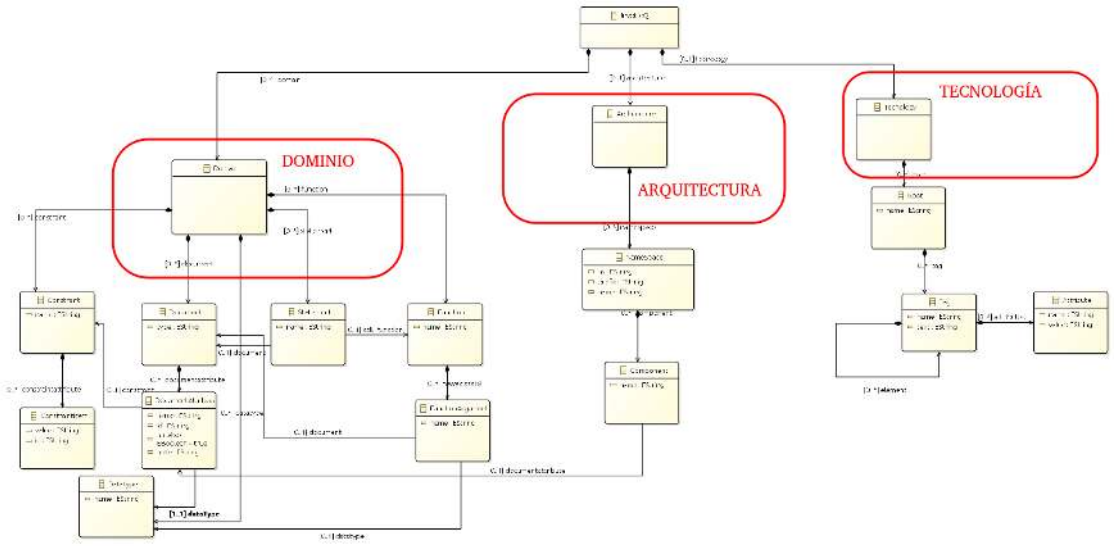

Figura 3-4.: Ramas del metamodelo. Fuente: Propia.

- *Note*: Descripción del campo.

- **Statement**: Representa las diferentes sentencias posibles para manipular los componenetes del dominio. Ej. Crear, Seleccionar, Modificar, entre otros.
- **DataType**: Representa los diferentes tipos de dato definidos por la DIAN para los campos de la factura electrónica. Ej. Alfanumérico, Numérico, Fechas, etc.
- **Constraint**: Esta clase representa las diferentes restricciones que existen sobre los diferentes campos de acuerdo al documento técnico de la DIAN.
- **ConstraintItem**: Tiene una relación de composición fuerte con la clase *Constraint* y representa cada uno de los elementos o posibles valores que pueden tener los campos restringidos.
- **Function**: Representa las diferentes funciones que pueden realizarse con la factura electrónica. Ej. Una función para generar la representación gráfica de la factura electrónica (generatePdf()).
- **FunctionArgument**: Esta clase tiene una relación de composición fuerte con la clase *Function* y representa los argumentos que puede tener una función.

3.3.2. Rama de Arquitectura

En la figura **3-6** se muestra la rama de arquitectura del metamodelo y se basa en UBL como se mostró en la sección 3.2.2.
Las clases en esta rama del modelo son:

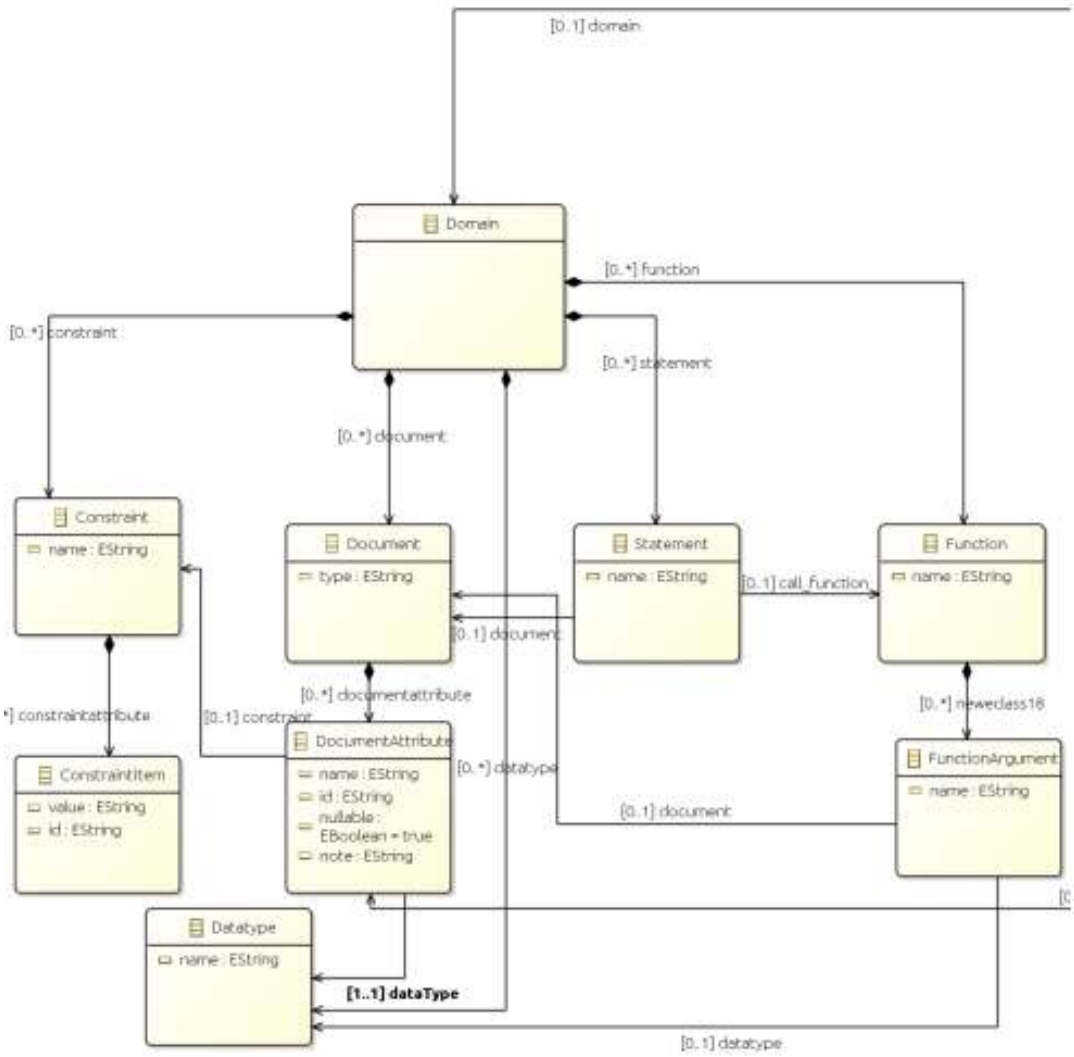

Figura 3-5.: Ramas de dominio en el metamodelo. Fuente: Propia.

- **NameSpace**: Representa los 'espacios de nombre' o librerías (de UBL o propias) que son requeridas para la generación de la factura electrónica. Esta clase tiene tres atributos. Ns, que indica el nombre del 'namespace' Ej: xmlns=ürn:oasis:names: specification:ubl :schema:xsd: Invoice-2". Prefix, que indica el prefijo con el que se marcan los componentes pertenecientes a la librería en el documento XML, ej: ext,cac. Name, es el nombre de la librería, Ej. CommonBasicComponents-2.

- **Component**: Esta clase representa cada uno de los componentes que pertenecen a una libreria y que son utilizados en el documento XML. Esta clase tiene una relación de composición fuerte con *NameSpace*.

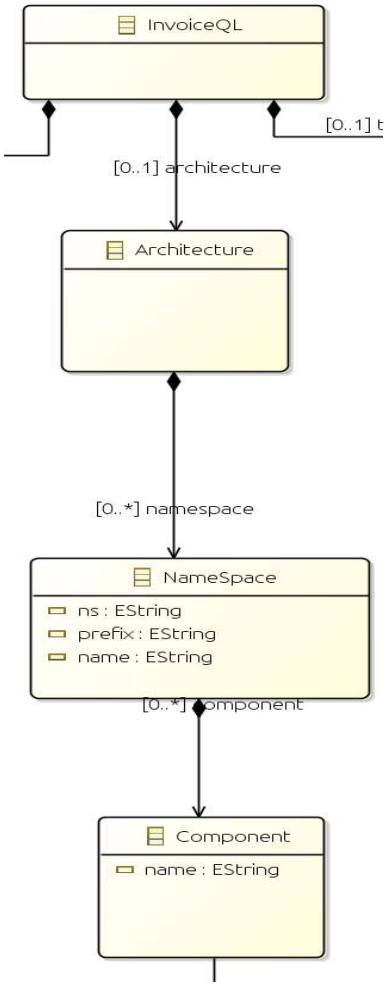

Figura 3-6.: Ramas de arquitectura en el metamodelo. Fuente: Propia.

3.3.3. Rama de Tecnología

Las clases de la rama de la tecnología representan los elementos necesarios para generar un documento XML, como se muestra en la figura **3-7**.

- **Root**: Esta clase representa la raíz de un documento XML.
- **Tag**: Representa las diferentes etiquetas o elementos que puede tener un documento XML. Tiene una relación de composición fuerte con la clase *Root*, además, tiene una relación de composición con sigo misma lo que indica que puede haber anidamiento.
- **Attribute**: Esta clase tiene una relación de composición fuerte con la clase *Tag* y una cardinalidad de 0 a n lo que indica que un tag puede tener más de un atributo o ninguno.

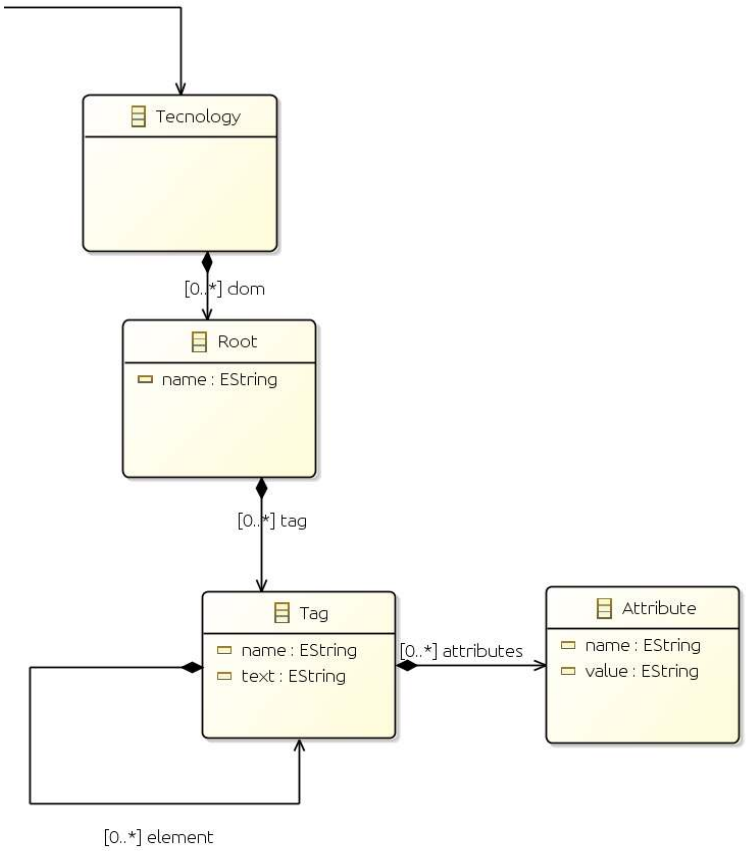

Figura 3-7.: Ramas de tecnología en el metamodelo. Fuente: Propia.

3.4. Validación del meta-modelo.

En esta sección se presenta la validación del meta-modelo. Con la ayuda del IDE Eclipse se crea una instancia del meta-modelo, esta instancia es un modelo que utiliza todos los componentes definidos en el metamodelo.

En la figura **3-8** se puede ver el modelo creado con base en el metamodelo de InvoiceQL. Este modelo tiene 3 componentes (o ramas) principales: Dominio, Arquitectura y tecnología.

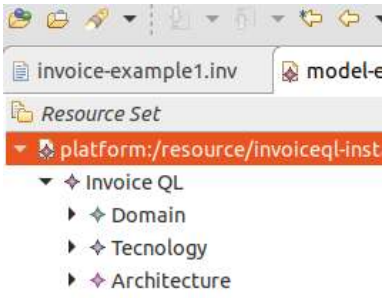

Figura 3-8.: Instancia del meta-modelo con las ramas de dominio, arquitectura y tecnología. Fuente: Propia.

3.4.1. Dominio.

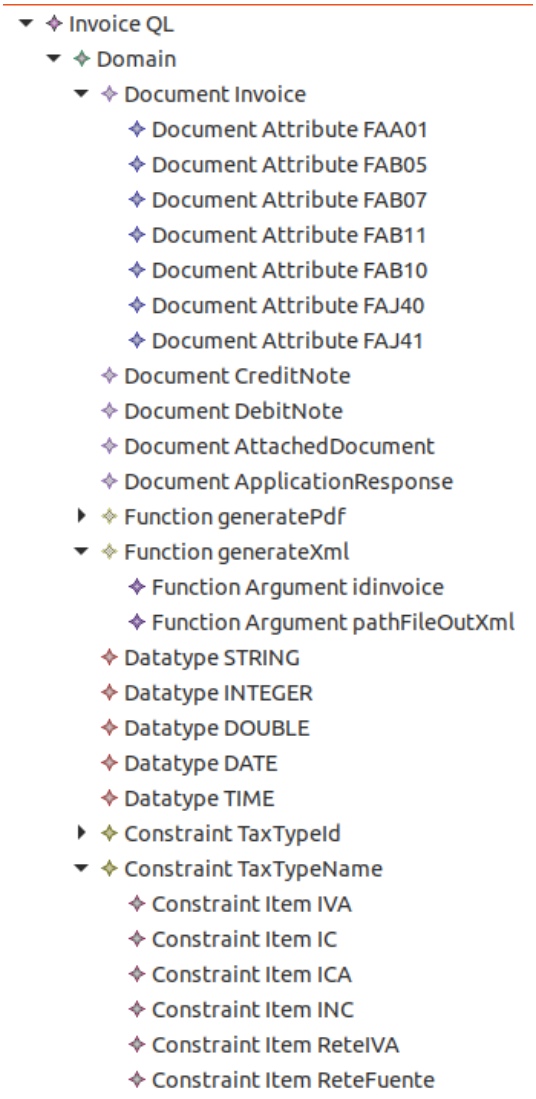

Figura 3-9.: Instancia del metamodelo con la rama de dominio extendida. Fuente: Propia.

En la figura **3-9** se observa la rama del dominio extendida. Se pueden crear Documentos(Invoice, CreditNote, DebitNote, entre otros), tipos de dato (String, Integer,Date, etc.), funciones con sus argumentos y restricciones(TaxTypeName, TaxTypeId, etc).

3.4.2. Arquitectura.

En la figura **3-10** se pueden ver diferentes librerías (NameSpace) y componentes(Component) que pertenecen a estas librerías. Esta arquitectura de librerías y componentes es la propuesta por UBL 2.1.

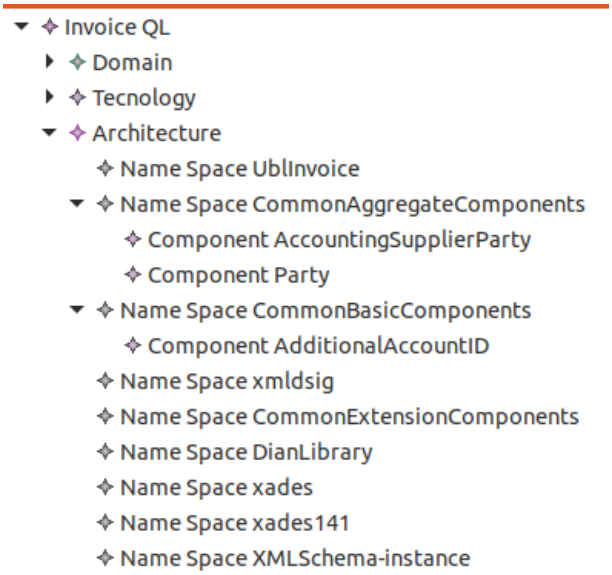

Figura 3-10.: Instancia del metamodelo con la rama de arquitectura extendida. Fuente: Propia.

3.4.3. Tecnología.

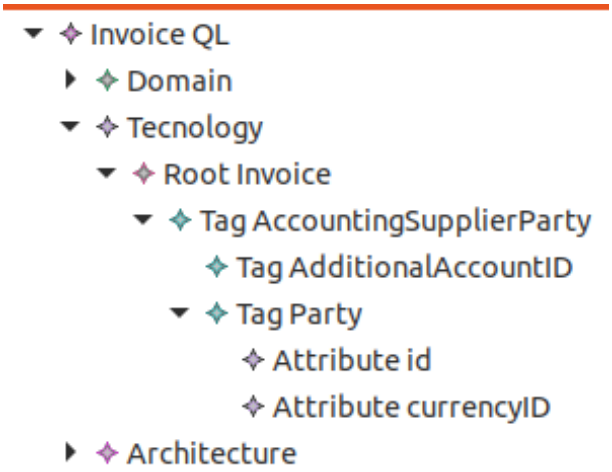

Figura 3-11.: Instancia del metamodelo con la rama de tecnología extendida. Fuente: Propia.

La figura **3-11** muestra la rama de tecnología con los elementos necesarios para crear un documento XML. Los componentes principales son las etiquetas (Tag) a las cuales pueden o no tener atributos. Las etiquetas pueden estar anidadas, pero siempre se debe partir de una raíz.

3.5. Diseño del DSL

Para el diseño de InvoiceQL se escriben las reglas gramaticales en formato Bacus-Naur Form (BNF) con base en el metamodelo presentado en la sección anterior. Posteriormente se realiza la verificación de las reglas gramaticales con un árbol de sintaxis abstracta (o AST por sus siglas en ingles).

3.5.1. Reglas gramaticales

1. LETTER ::= 'A'| 'B'| 'C'| 'D'| 'E'| 'F'| 'G'| 'H'| 'I'| 'J'| 'K'| 'L'| 'M'| 'N'| 'O'| 'P'| 'Q'| 'R'| 'S'| 'T'| 'U'| 'V'| 'W'| 'X'| 'Y'| 'Z'| 'a'| 'b'| 'c'| 'd'| 'e'| 'f'| 'g'| 'h'| 'i'| 'j'| 'k'| 'l'| 'm'| 'n'| 'o'| 'p'| 'q'| 'r'| 's'| 't'| 'u'| 'v'| 'w'| 'x'| 'y'| 'z ';

2. DIGIT ::= '0'| '1'| '2'| '3'| '4'| '5'| '6'| '7'| '8'| '9';

3. SYMBOL ::= '['| ']'| ''| ''| '('| ')'| '¡'| '¿'| '='| '|'| '.'| ','| ';'| '_ ';

4. CHARACTER ::= LETTER |DIGIT |SYMBOL;

5. STRING ::= '''(CHARACTER)* ''';

6. BOOLEAN ::= 'TRUE'|'FALSE'

7. INTEGER ::= '-'? (DIGIT)+

8. DOUBLE ::= '-'? (DIGIT)+ '.'(DIGIT)+

9. ID ::= (LETTER)+(LETTER |DIGIT+)*;

10. Datatype ::= INTEGER |STRING |DOUBLE |BOOLEAN |'NULL';

11. DatatypeName ::= 'INTEGER'|'STRING'|'DOUBLE'|'BOOLEAN'|'DATE'|'TIME';

12. Function ::= ID '(' Datatype? (',' Datatype)* ')';

13. DeclareConstraintStatement ::= 'CREATE' 'CONSTRAINT' ID 'IN' '('Datatype (','Datatype)* ')';

14. DeclareFieldStatement ::= 'CREATE' 'FIELD' ID DatatypeName ('NOT' 'NULL')? ('CONSTRAINT'ID)? ('NAME'STRING)? ('NOTE' STRING)?;

15. CreateDocumentInvoiceStatement ::= 'CREATE' 'DOCUMENT' 'INVOICE' ID '('ID '='Datatype (',' ID '=' Datatype)* ')';

16. SelectInvoiceStatement: 'SELECT' 'INVOICE ('WHERE' 'ID' '=' ID |'WHERE' ID '=' Datatype | Function);

17. AddUblLibraryStatement ::= 'ADD' 'LIBRARY' ID 'PREFIX' STRING 'NS' STRING;
18. DefineTagStatement ::='DEFINE' 'TAG' ID ('LIBRARY' ID)* ('PARENT' ID)* ('FIELD' ID)* ('(' ID '=' Datatype (',' ID '=' Datatype)* ')')*;
19. Statement ::= DeclareConstraintStatement |DeclareFieldStatement |CreateDocument InvoiceStatement |SelectInvoiceStatement |AddUblLibraryStatement |DefineTagStatement;

De la regla 1 a la regla 11 se definen los terminales, la regla 11 define los diferentes tipos de dato, la regla 12 define las funciones y sus argumentos, la regla 13 define la forma de crear restricciones, la regla 14 permite crear los campos de la factura, la regla 15 permite crear el documento factura, la regla 16 permite realizar consultas de selección, la regla 17 permite agregar las librerías de un documento XML(las de UBL y externas), la regla 18 permite representar etiquetas XML(junto con sus atributos) y la regla 19 agrupa las diferentes sentencias.

La regla 17 refleja la rama de arquitectura del metamodelo y la 18 la rama de tecnología.

3.5.2. AST con sentencias de ejemplo

A continuación se mostrarán algunas sentencias de ejemplo escritas en invoiceQL, se utilizan árboles de sintaxis abstracta (AST) para ver el recorrido de cada elemento de la sentencia.
Ejemplo 1:
En esta sentencia se crea una restricción para un campo que solo puede tener los valores: IVA, IC e ICA.

```
1    CREATE CONSTRAINT TaxTypeName IN ('IVA', 'IC', 'ICA')
```

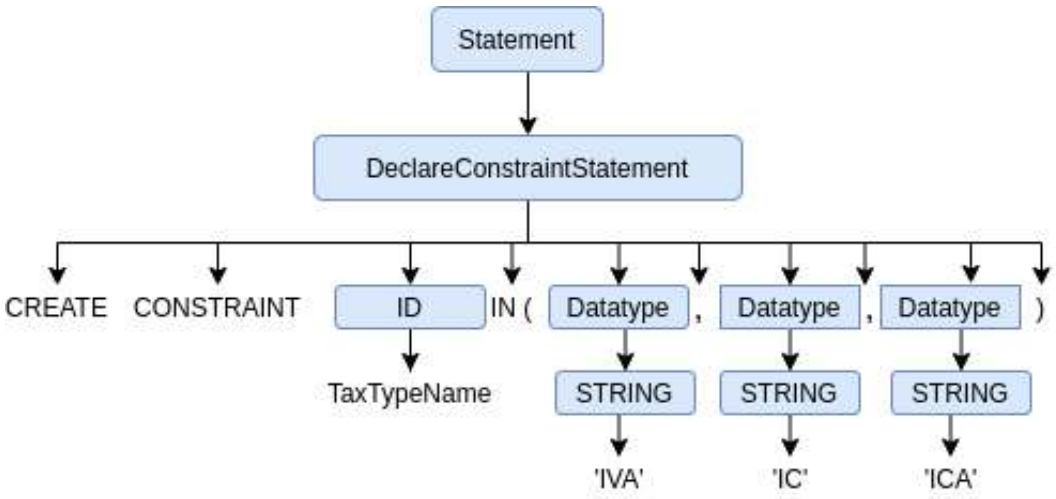

Figura 3-12.: AST Ejemplo 1. Declaración de restricción. Fuente: Propia.

Ejemplo 2:

Con esta sentencia se declara un campo: Un identificador, un tipo de dato, nombre y descripción del campo.

```
1 CREATE FIELD FAB05 INTEGER NAME 'InvoiceAuthorization' NOTE 'NUMERO_DE_
    AUTORIZACION_DE_RESOLUCION_DE_FACTURACION';
```

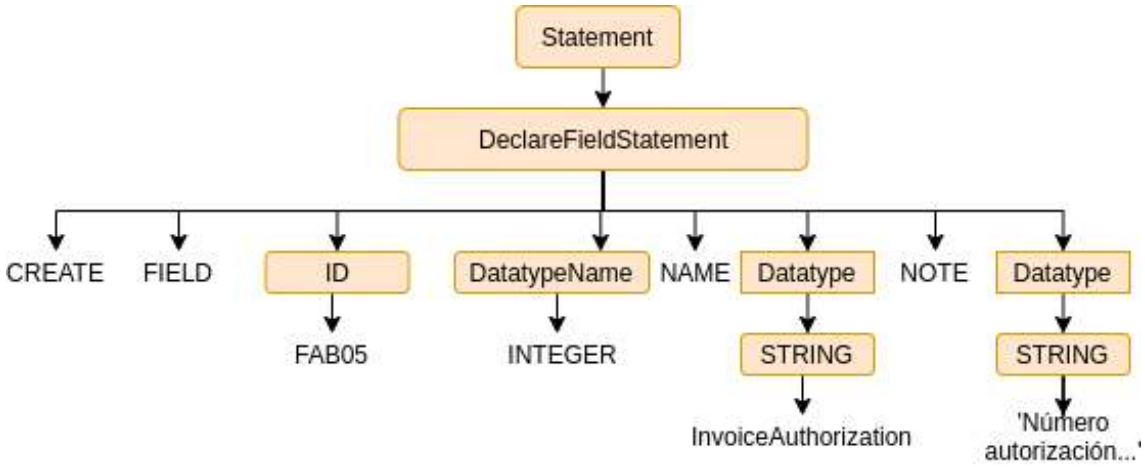

Figura 3-13.: AST Ejemplo 2. Declaración de campo de factura. Fuente: Propia.

Ejemplo 3:

En esta sentencia se crea un documento tipo factura (Invoice) y se le agregan los diferentes campos y sus respectivos valores.

```
1 CREATE DOCUMENT INVOICE f1(FAA01 = NULL,
2 FAB05 = '18760000001',
3 FAB07 = 15,
4 FAB10=TRUE);
```

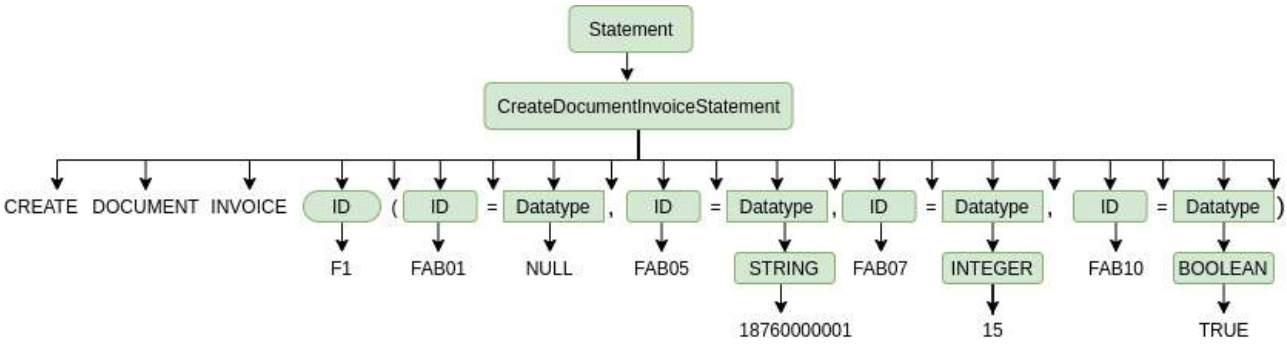

Figura 3-14.: AST Ejemplo 3. Creación de un documento tipo factura. Fuente: Propia.

Ejemplo 4:

En esta sentencia se crea una consulta de selección con una función que genera el archivo XML de la factura electrónica.

```
1 SELECT INVOICE xml('f1','/path/out.xml');
```

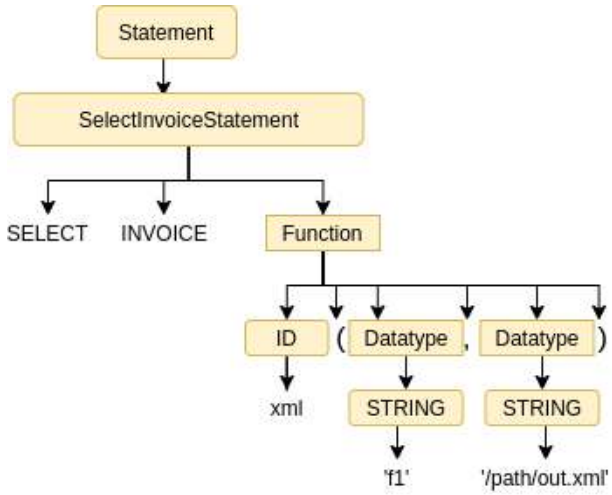

Figura 3-15.: AST Ejemplo 4. Consulta de selección con una función. Fuente: Propia.

Ejemplo 5:

En este ejemplo se realiza una consulta de selección simple.

```
1 SELECT INVOICE WHERE FAB05='18760000001'
```

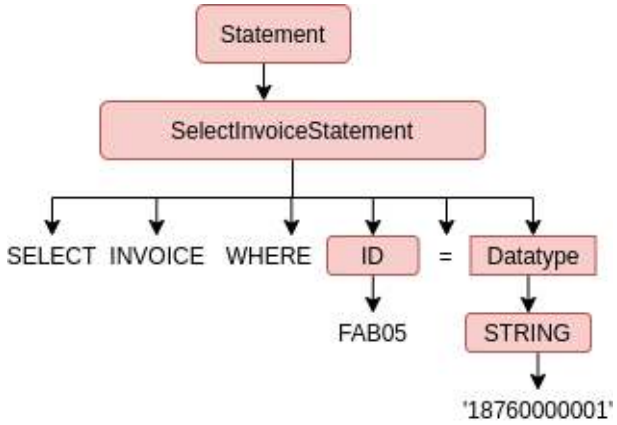

Figura 3-16.: AST Ejemplo 5. Consulta de selección simple. Fuente: Propia.

3.6. Diseño de las transformaciones

Para realizar las transformaciones se hace u recorrido por el metamodelo empezando por la rama del dominio, luego la arquitectura y por último la tecnología, además, se identifican las relaciones entre las ramas.

Para transformar sentencias invoiceQL a documentos XML(factura electrónica) se plantean dos estrategias: Transformación indirecta, y transformación directa.

3.6.1. Transformación indirecta

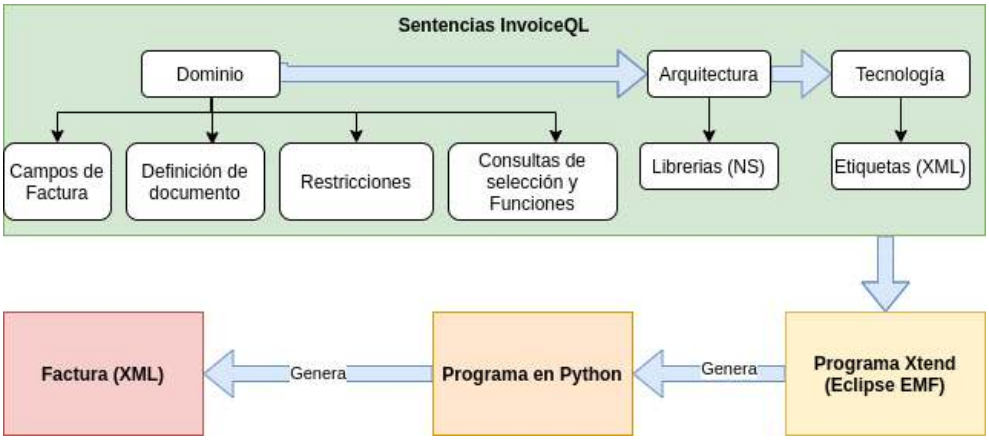

Figura 3-17.: Estrategia de transformación indirecta. Fuente: Propia.

Con esta estrategia la factura electrónica (documento XML) se genera al ejecutar un programa en Python que es generado por Xtend (y la ayuda de las herramientas del framework EMF de eclipse) el cual se basa en las reglas descritas en sentencias escritas con invoiceQL.

Como se ve en la figura **3-17** en las sentencias invoiceQL se utilizan todos los elementos del metamodelo (dominio, arquitectura y tecnología) pues el programa generado en Python utiliza el modulo 'xml.etree' que requiere las librerías (namespaces) y las etiquetas para generar el documento XML.

El programa en python generado tiene tres elementos:

- InvoiceQlBuilder.py : Es una clase en la que se mapean los campos de la factura electrónica y se construye el XML.
- InvoiceQlManager.py : Es el programa que se ejecuta e instancia la clase InvoiceQlBuilder. En este fichero se encuentra la función main().

- InvoiceQlData.json : Son los valores de la factura que se asignaron con las sentencias escritas en InvoiceQL.

3.6.2. Transformación directa

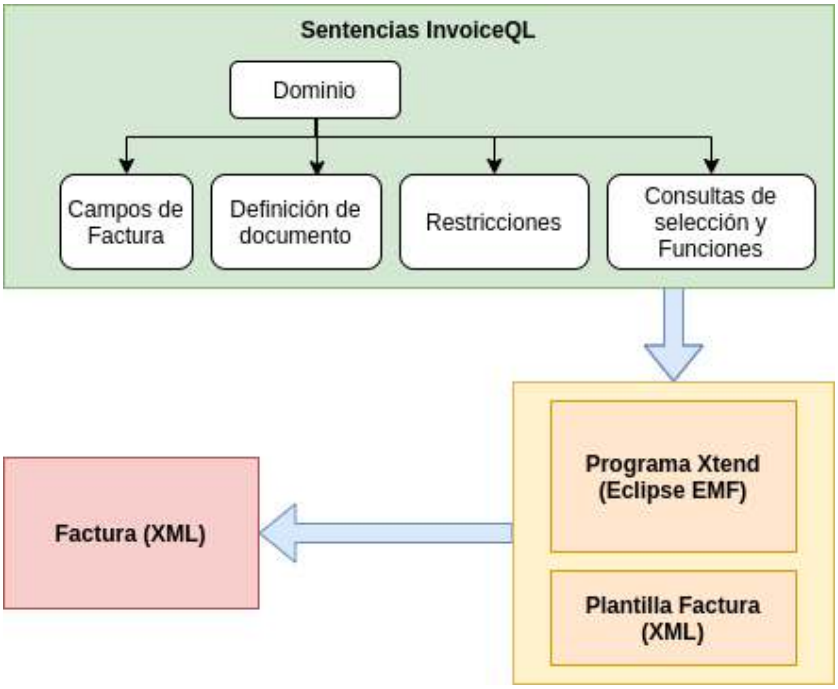

Figura 3-18.: Estrategia de transformación directa. Fuente: Propia.

Esta estrategia de transformación consiste en tomar las sentencias escritas con invoiceQL y por medio de un programa en Xtend y una plantilla de la factura se genera directamente el documento XML.

La estrategia es simple, se reemplazan los valores de los campos de la factura (que se suministran en las sentencias invoiceQL) en la plantilla.

En la figura **3-18** se puede observar que para esta estrategia solo se utilizan los elementos del dominio pues la plantilla tiene definidas las librerías y estructura del documento XML.

3.7. Implementación del DSL InvoiceQL.

Para la implementación de invoiceQL se utilizaron las herramientas de modelado de eclipse, puntualmente el framework de modelado de eclipse (EMF, por sus siglas en inglés).

3.7.1. Definición del lenguaje invoiceQL con Xtext.

Con base en el diseño presentado en las secciones anteriores el DSL escrito con Xtext es el siguiente:

```
1  grammar org.invoice.InvoiceqlDsl with org.eclipse.xtext.common.
       Terminals
2
3  generate invoiceqlDsl "http://www.invoice.org/InvoiceqlDsl"
4
5  //DOMAIN
6  Script:
7          ((statementList+=Statement) ';')+;
8
9  Statement:
10         DeclareFieldStatement | CreateDocumentInvoiceStatement |
               DeclareConstraintCheckStatement | SelectInvoiceStatement |
11         AddUblLibraryStatement | DefineTagStatement;
12
13 DeclareConstraintCheckStatement:
14         'CREATE' 'CONSTRAINT' name=ID 'IN' '(' items+=Datatype (',')
               items+=Datatype)* ')';
15
16 DeclareFieldStatement:
17         'CREATE' 'FIELD' name=ID datatype=DatatypeName (notnull='NOT'
               'NULL')? ('CONSTRAINT' constraint=ID)? ('COMPONENT'
18         component=ID)? ('NAME'
19         nameField=STRING)? ('NOTE'
20         note=STRING)?;
21
22 CreateDocumentInvoiceStatement:
23         type='CREATE' 'DOCUMENT' 'INVOICE' name=ID '(' fields+=ID '='
               values+=Datatype (',' fields+=ID '=' values+=Datatype)*
24         ')';
25
26 SelectInvoiceStatement:
27         'SELECT' ('INVOICE' (where='WHERE' whereId='ID' '=' idfield=ID
               | where='WHERE' idfield=ID '=' valWhere=Datatype) |
28         function=Function);
29
30 Function:
31         name=ID '(' args+=Datatype? (',' args+=Datatype)* ')';
```

```
32
33          Datatype:
34                  integerVal=INTEGER | stringVal=STRING | doubleVal=DOUBLE |
                        boolVal=BOOLEAN | nullVal='NULL';
35
36          DatatypeName:
37                  'INTEGER' | 'STRING' | 'DOUBLE' | 'BOOLEAN' | 'DATE' | 'TIME';
38
39                  //ARCHITECTURE
40          AddUblLibraryStatement:
41                  'ADD' 'LIBRARY' name=ID 'PREFIX' prefix=STRING 'NS' ns=STRING;
42
43                  //TECNOLOGY
44          DefineTagStatement:
45                  'DEFINE' 'TAG' name=ID ('LIBRARY' library=ID)* ('PARENT'
                        parent=ID)* ('FIELD' field=ID)* ('(' attrsName+=ID '='
                        attrsVal+=Datatype (','
46                  attrsName+=ID '=' attrsVal+=Datatype)* ')')*;
47
48                  //TEMINAL
49          terminal BOOLEAN:
50                  'TRUE' | 'FALSE';
51
52          terminal INTEGER:
53                  '-'? ('0'..'9')+;
54
55          terminal DOUBLE:
56                  '-'? ('0'..'9')+ '.' ('0'..'9')+;
57
58          @Override
59          terminal STRING:
60                  '"' ('\\' . /* 'b'|'t'|'n'|'f'|'r'|'u'|'"'|"'"|'\\' */ | !('\\
                        ' | '"'))* '"';
```

De la línea 6 a la 38 se definen los elementos del dominio (Reglas DIAN), de la 39 a la 42 los elementos de la arquitectura (UBL) y de la 43 a la 47 los elementos de la tecnología (XML). De la línea 48 en adelante se definen los elementos terminales del lenguaje.

3.7.2. Definición de las transformaciones con Xtend.

Como se explicó en la sección 3.6 para el prototipo desarrollado en este trabajo se siguieron dos estrategias de transformación: directa e indirecta. Para seleccionar uno u otro modo se escriben sentencias de invoiceQL que instancien la función correspondiente (Para una mejor comprensión en la sub-sección 3.7.3 se muestran algunos ejemplos).

A continuación se muestra la función principal que realiza las transformaciones. (El código

completo se anexa al final)

```
1         override void doGenerate(Resource resource, IFileSystemAccess2 fsa,
              IGeneratorContext context) {
2                 mapConstraint = new HashMap();
3                 listField = new ArrayList();
4                 mapLibrary = new HashMap();
5                 lstTag = new ArrayList();
6
7                 for (script : resource.allContents.toIterable.filter(Script))
                      {
8                     // 1- Buid Objects
9                     for (statement : script.statementList) {
10                        if (statement instanceof
                              DeclareConstraintCheckStatement) {
11                            compileDeclareConstraintCheckStatement
                                  (statement, fsa)
12                        }
13                        if (statement instanceof DeclareFieldStatement
                              ) {
14                            compileDeclareFieldStatement(statement
                                  , fsa)
15                        }
16                        if (statement instanceof
                              CreateDocumentInvoiceStatement) {
17                            compileCreateDocumentInvoiceStatement(
                                  statement, fsa)
18                        }
19                        if (statement instanceof
                              CreateDocumentInvoiceStatement) {
20                            compileCreateDocumentInvoiceStatement(
                                  statement, fsa)
21                        }
22                        if (statement instanceof
                              AddUblLibraryStatement) {
23                            compileAddUblLibraryStatement(
                                  statement, fsa)
24                        }
25                        if (statement instanceof DefineTagStatement) {
26                            compileDefineTagStatement(statement,
                                  fsa)
27                        }
28                    }
29                    // 2- Select Querys
30                    for (statement : script.statementList) {
31                        if (statement instanceof
                              SelectInvoiceStatement) {
32                            compileSelectInvoiceStatement(
                                  statement, fsa)
```

```
33                                      }
34                                  }
35                              }
36                      }
```

En la línea 7 hay un ciclo 'for' que recorre todas las sentencias invoiceQL escritas en un 'script', y a continuación dependiendo del tipo de sentencia se ejecuta una función que realiza el respectivo procesamiento de la sentencia.

3.7.3. Ejemplo de un 'script' escrito en invoiceQL.

Un 'script' escrito en invoiceQL se muestra a continuación.

```
1  /*************************************************************
2   *  D O M A I N
3   *************************************************************/
4
5  //Create Constraints
6  CREATE CONSTRAINT TaxTypeId IN ('01','02','03','04','05','06','07','20');
7  CREATE CONSTRAINT TaxTypeName IN ('IVA','IC','ICA','INC','ReteIVA','ReteFuente
       ','ReteICA','FtoHorticultura');
8
9  //Create Fields of Invoice
10 CREATE FIELD FAA01 STRING;
11 CREATE FIELD FAB05 INTEGER NAME 'InvoiceAuthorization' NOTE 'Nuni...';
12 CREATE FIELD FAB07 DATE NAME 'StartDate' NOTE 'Fecha de inicio...';
13 CREATE FIELD FAB08 DATE NAME 'EndDate' NOTE 'Fecha final de la...';
14 CREATE FIELD FAB10 STRING NOT NULL NAME 'Prefix' NOTE 'Prefijo de...';
15 CREATE FIELD FAB11 STRING NAME 'From' NOTE 'Valor inicial del...';
16 CREATE FIELD FAB12 STRING NAME 'To' NOTE 'Valor final del...';
17 CREATE FIELD FAB14 STRING NAME 'IdentificationCode';
18 CREATE FIELD FAB19 STRING NAME 'ProviderID' NOTE 'Identificador del...';
19 CREATE FIELD FAB24 STRING NAME 'softwareID' NOTE 'Identificador Soft...';
20 CREATE FIELD FAB27 STRING NAME 'SoftwareSecurityCode' NOTE 'Huella del...';
21 CREATE FIELD FAB31 STRING NAME 'AuthorizationProviderID';
22 CREATE FIELD FAB36 STRING NAME 'QRCode';
23 CREATE FIELD FAJ40 STRING CONSTRAINT TaxTypeId NAME 'ID' NOTE 'Identifi...';
24 CREATE FIELD FAJ41 STRING CONSTRAINT TaxTypeName NAME 'Name' NOTE 'Nombre...';
25
26 //Create particular invoice
27 CREATE DOCUMENT INVOICE f1(
28         FAA01 = NULL,
29         FAB05 = '18760000001',
30         FAB07 = '2019-01-19',
31         FAB08 = '2030-01-19',
32         FAB10 = 'SETP',
33         FAB11 = 990000000,
34         FAB12 = 995000000,
```

```
35          FAB14 = 'CO',
36          FAB19 = '800197268',
37          FAB24='56f2ae4e-9812-4fad-9255-08fcfcd5ccb0',
38          FAB27 ='
                a8d18e4e5aa00b44a0b1f9ef413ad8215116bd3ce91730d580eaed795c83b5a32fe6f0823abc7
                ',
39          FAB31 ='800197268',
40          FAB36 = 'QRcode'
41  );
42
43
44  //Queries over the invoice
45  SELECT INVOICE WHERE ID=f1;
46
47  SELECT INVOICE WHERE FAB05='18760000001';
48
49  //With Python : Undirect
50  SELECT loadAppPython('/opt/invoiceql/python/invoiceql-gen-example');
51
52  //With Template : Direct
53  SELECT xml('f1','factura-f1.xml');
54
55  /*************************************************************************
56   *  A R C H I T E C T U R E
57   *************************************************************************/
58  //Declare Libraries
59  ADD LIBRARY UblInvoice PREFIX '' NS 'urn:oasis:names:specification:ubl:schema:
        xsd:Invoice-2';
60  ADD LIBRARY CommonAggregateComponents PREFIX 'cac' NS 'urn:oasis:names:
        specification:ubl:schema:xsd:CommonAggregateComponents-2';
61  ADD LIBRARY CommonBasicComponents PREFIX 'cbc' NS 'urn:oasis:names:
        specification:ubl:schema:xsd:CommonBasicComponents-2';
62  ADD LIBRARY xmldsig PREFIX 'ds' NS 'http://www.w3.org/2000/09/xmldsig#';
63  ADD LIBRARY CommonExtensionComponents PREFIX 'ext' NS 'urn:oasis:names:
        specification:ubl:schema:xsd:CommonExtensionComponents-2';
64  ADD LIBRARY DianLibrary PREFIX 'sts' NS 'dian:gov:co:facturaelectronica:
        Structures-2-1';
65  ADD LIBRARY xades PREFIX 'xades' NS 'http://uri.etsi.org/01903/v1.3.2#';
66  ADD LIBRARY xades PREFIX 'xades141' NS 'http://uri.etsi.org/01903/v1.4.1#';
67  ADD LIBRARY XMLSchema_instance PREFIX 'xsi' NS 'http://www.w3.org/2001/
        XMLSchema-instance';
68
69
70  /*************************************************************************
71  * T E C N O L O G Y
72   *************************************************************************/
73  //Declare tags
74  DEFINE TAG Invoice;
```

```
75 DEFINE TAG UBLExtensions LIBRARY CommonExtensionComponents PARENT Invoice;
76 DEFINE TAG UBLExtension LIBRARY CommonExtensionComponents PARENT UBLExtensions
       ;
77 DEFINE TAG ExtensionContent LIBRARY CommonExtensionComponents PARENT
       UBLExtensions;
78 DEFINE TAG DianExtensions LIBRARY DianLibrary PARENT ExtensionContent;
79 DEFINE TAG InvoiceControl LIBRARY DianLibrary PARENT DianExtensions;
80 DEFINE TAG InvoiceAuthorization LIBRARY DianLibrary PARENT InvoiceControl;
81 DEFINE TAG AuthorizationPeriod LIBRARY DianLibrary PARENT InvoiceControl;
82 DEFINE TAG StartDate LIBRARY CommonBasicComponents PARENT AuthorizationPeriod
       FIELD FAB07;
83 DEFINE TAG EndDate LIBRARY CommonBasicComponents PARENT AuthorizationPeriod
       FIELD FAB08;
84 DEFINE TAG AuthorizedInvoices LIBRARY DianLibrary PARENT InvoiceControl;
85 DEFINE TAG Prefix LIBRARY DianLibrary PARENT AuthorizedInvoices FIELD FAB10;
86 DEFINE TAG From LIBRARY DianLibrary PARENT AuthorizedInvoices FIELD FAB11;
87 DEFINE TAG To LIBRARY DianLibrary PARENT AuthorizedInvoices FIELD FAB12;
88 DEFINE TAG InvoiceSource LIBRARY DianLibrary PARENT DianExtensions;
89 DEFINE TAG IdentificationCode LIBRARY CommonBasicComponents PARENT
       InvoiceSource FIELD FAB14(listAgencyID='6',listAgencyName='United␣Nations␣
       Economic␣Commission␣for␣Europe',listSchemeURI='urn:oasis:names:
       specification:ubl:codelist:gc:CountryIdentificationCode-2.1');
90 DEFINE TAG SoftwareProvider LIBRARY DianLibrary PARENT DianExtensions;
91 DEFINE TAG ProviderID LIBRARY DianLibrary PARENT SoftwareProvider FIELD FAB19(
       schemeAgencyID='195',schemeAgencyName='CO,␣DIAN␣(Direccion␣de␣Impuestos␣y␣
       Aduanas␣Nacionales)',schemeID='4',schemeName='31');
92 DEFINE TAG SoftwareID LIBRARY DianLibrary PARENT SoftwareProvider FIELD FAB24(
       schemeAgencyID='195',schemeAgencyName='CO,␣DIAN␣(Direccion␣de␣Impuestos␣y␣
       Aduanas␣Nacionales)');
93 DEFINE TAG SoftwareSecurityCode LIBRARY DianLibrary PARENT DianExtensions
       FIELD FAB27(schemeAgencyID='195',schemeAgencyName='CO,␣DIAN␣(Direccion␣de␣
       Impuestos␣y␣Aduanas␣Nacionales)');
94 DEFINE TAG AuthorizationProvider LIBRARY DianLibrary PARENT DianExtensions;
95 DEFINE TAG AuthorizationProviderID LIBRARY DianLibrary PARENT
       AuthorizationProvider FIELD FAB31(schemeAgencyID='195', schemeAgencyName='
       CO,␣DIAN␣(Direccion␣de␣Impuestos␣y␣Aduanas␣Nacionales)',schemeID='4',
       schemeName='31');
96 DEFINE TAG QRCode LIBRARY DianLibrary PARENT DianExtensions FIELD FAB36;
```

(Fin del Script)

Algunas observaciones acerca del código mostrado anteriormente son:

- Los comentarios son equivalentes a los utilizados en lenguaje Java. '//' y '/* */'.
- Las palabras reservadas del lenguaje se escriben en mayúscula sostenida. Ej. DEFINE, TAG, FIELD, SELECT, etc.

- De la línea 5 a la 8 se definen las diferentes restricciones que pueden tener los valores de los campos de acuerdo a la documentación técnica de la DIAN.
- De la línea 9 a la 25 se definen los campos de la factura indicando el tiempo de dato y opcionalmentente: el nombre del campo (según UBL), la obligatoriedad, y observaciones acerca del campo.
- De la línea 26 a la 41 se escribe una sentencia con la que se crea una factura electrónica y se indican los valores de cada campo.
- En la línea 45 se escribe una sentencia de selección en el que se consulta una factura por su identificador.
- En la línea 47 hay una sentencia de consulta de selección en donde el criterio de búsqueda es un campo en particular. (En este ejemplo el campo FAB05).
- En la línea 50 se escribe una sentencia de selección de función, en este caso se invoca la función loadAppPython() a la que se le envía como argumento la ruta en donde se desean generar los archivos .py de la aplicación que generará la factura electrónica (De acuerdo a la estrategia de transformación indirecta explicada en secciones anteriores).
- En la línea 53 se invoca la función xml() que recibe como argumentos el identificador de la factura y la ruta en que se desea generar el archivo XML (factura electrónica). De acuerdo a la estrategia de transformación directa mencionada en secciones anteriores.
- De la línea 55 a la 68 se definen las diferentes librerías de componentes (UBL o externas) que se utilizarán en la factura electrónica.
- De la línea 70 a la 96 se define el árbol XML por medio de las palabras reservadas 'DEFINE TAG'.

3.7.4. IDE para invoiceQL basado en Eclipse.

Como parte de las herramientas de Eclipse EMF esta la posibilidad de generar un IDE que reconoce la sintaxis del DSL y ayuda a generar los artefactos de acuerdo a las instrucciones escritas en Xtend.
El IDE basado en eclipse para invoiceQL tiene las siguientes características:

- Reconocimiento de sintaxis resaltando palabras reservadas y expresiones. (Highlighting)
- Función de auto-completado.
- Indicador de errores de sintaxis.

- Generador de artefactos según las transformaciones escritas en Xtend.
- Explorador de proyectos.
- Consola de logs.

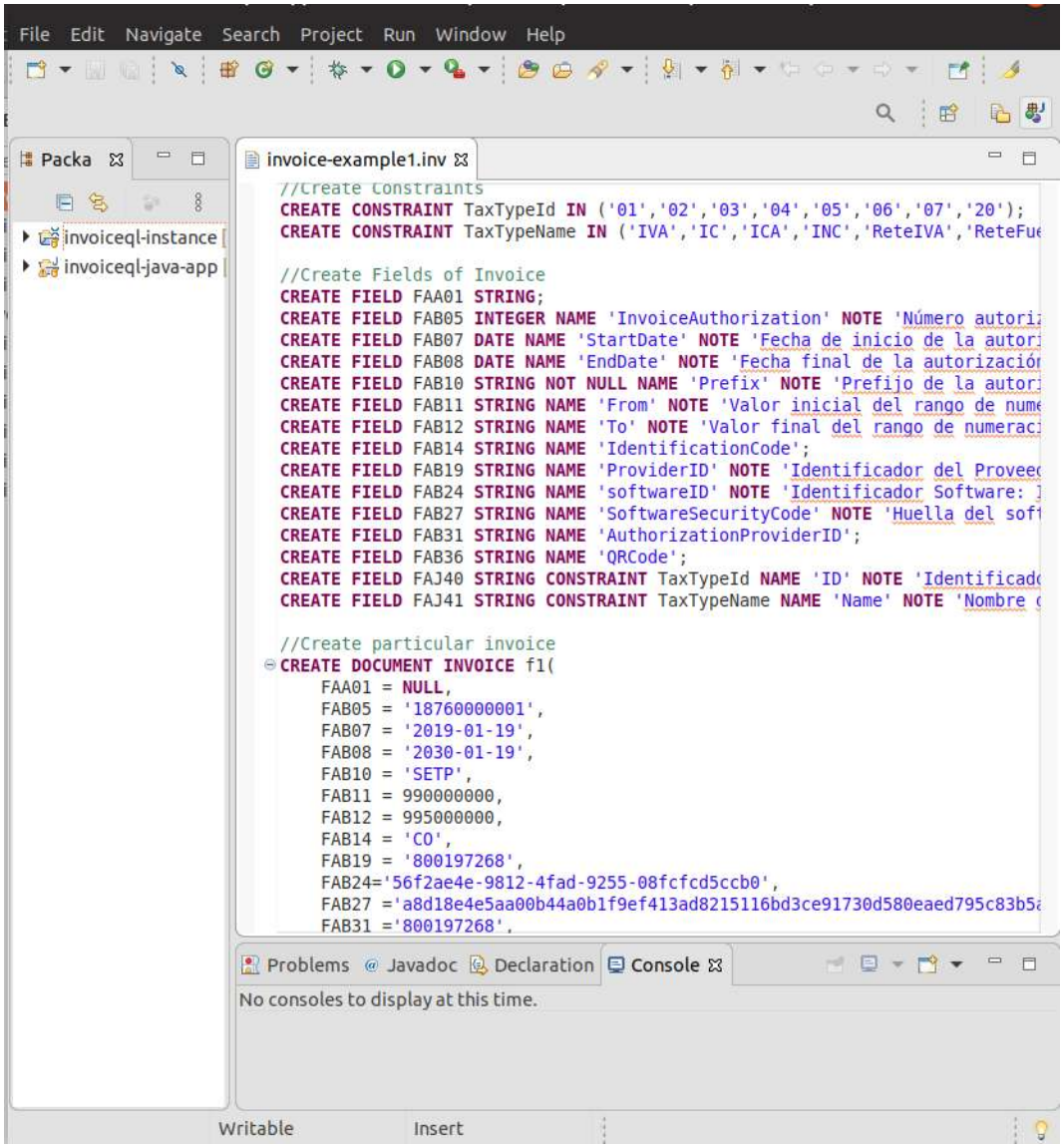

Figura 3-19.: IDE basado en Eclipse para InvoiceQL. Fuente: Propia.

3.8. Caso de uso. InvoiceQL y el ERP ODOO.

Como se explicó en la sección 2.6 ODOO es un ERP de código abierto. Como parte de este trabajo se desarrolló un módulo para ODOO que genera facturas electrónicas utilizando

código fuente en lenguaje python generado por programa que realiza las transformaciones escritas en Xtend.

De acuerdo a la estrategia de transformación indirecta (que se explica en la sección 3.6.1) se genera código fuente en python que se utiliza en un módulo de facturación electrónica en ODOO.

La versión en la que se desarrolló el módulo es ODOO 13.0 que está desarrollada con python 3 y tiene una arquitectura MVC(modelo vista-controlador) [29] con algunos componentes previamente desarrollados que pueden reutilizarse.

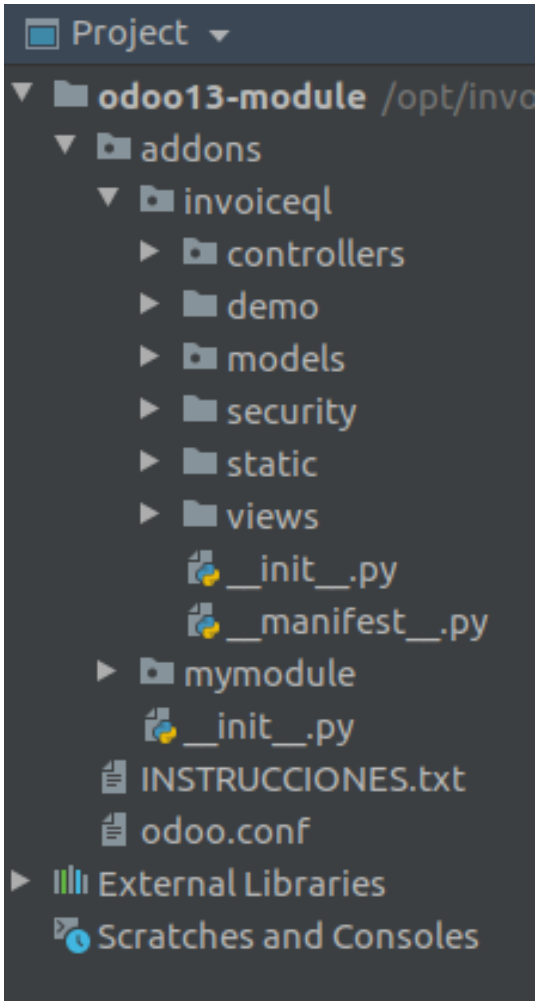

Figura 3-20.: Estructura de un módulo ODOO. Fuente: Propia.

En la figura **3-20** se puede ver la estructura de un módulo de ODOO. En esta estructura se encuentra un directorio en donde se guardan las vistas (view) que son descriptores escritos en

formato XML. También está la carpeta 'models' en donde hay clases en python que mapean las tablas de la base de datos siguiendo la técnica ORM (Object-Relational mapping por sus siglas en inglés). Y también está el directorio 'controllers' en donde se programan los controladores web que conectan elementos de la vista y del modelo.

El programa que realiza las transformaciones (escrito con Xtend) genera los archivos 'models.py' e 'InvoiceQlBuilder.py' que se ubican como lo indica la figura **3-21**.

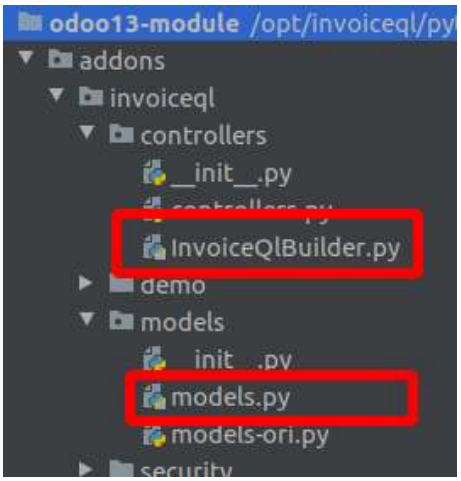

Figura 3-21.: Archivos generados para el módulo invoiceQL en ODOO. Fuente: Propia.

El archivo 'InvoiceQlBuilder.py' construye la estructura del documento XML utilizando las librerías xml.dom y xmletree de python. A continuación se muestra parte del código generado.

```
1         from xml.etree.ElementTree import Element, SubElement, Comment,
              tostring
2         import xml.etree.ElementTree as ET
3         from xml.dom import minidom
4
5         class  InvoiceQlBuilder:
6
7                 def __init__(self, dataJson):
8                         self.dataJson = dataJson
9
10                def getXml(self):
11                        # Pretty string
12                        invoice=self.buildXml()
13                        xmlstr = minidom.parseString(ET.tostring(invoice)).
                              toprettyxml(indent="   ")
14                        return xmlstr
15
16                def buildXml(self):
```

```
17                  #NAMESPACES
18                  ET.register_namespace('xades141', 'http://uri.etsi.org
                        /01903/v1.4.1#')
19                  ET.register_namespace('ext', 'urn:oasis:names:
                        specification:ubl:schema:xsd:
                        CommonExtensionComponents-2')
20                  ET.register_namespace('cbc', 'urn:oasis:names:
                        specification:ubl:schema:xsd:CommonBasicComponents
                        -2')
21                  ET.register_namespace('sts', 'dian:gov:co:
                        facturaelectronica:Structures-2-1')
22                  ET.register_namespace('xsi', 'http://www.w3.org/2001/
                        XMLSchema-instance')
23                  ET.register_namespace('ds', 'http://www.w3.org
                        /2000/09/xmldsig#')
24                  ET.register_namespace('cac', 'urn:oasis:names:
                        specification:ubl:schema:xsd:
                        CommonAggregateComponents-2')
25                  ET.register_namespace('', 'urn:oasis:names:
                        specification:ubl:schema:xsd:Invoice-2')
26
27
28                  #BUILD XML
29                  Invoice = Element('Invoice')
30                  UBLExtensions = SubElement(Invoice,'{urn:oasis:names:
                        specification:ubl:schema:xsd:
                        CommonExtensionComponents-2}UBLExtensions')
31                  UBLExtension = SubElement(UBLExtensions,'{urn:oasis:
                        names:specification:ubl:schema:xsd:
                        CommonExtensionComponents-2}UBLExtension')
32                  ExtensionContent = SubElement(UBLExtensions,'{urn:
                        oasis:names:specification:ubl:schema:xsd:
                        CommonExtensionComponents-2}ExtensionContent')
33                  DianExtensions = SubElement(ExtensionContent,'{dian:
                        gov:co:facturaelectronica:Structures-2-1}
                        DianExtensions')
34                  InvoiceControl = SubElement(DianExtensions,'{dian:gov:
                        co:facturaelectronica:Structures-2-1}
                        InvoiceControl')
35                  InvoiceAuthorization = SubElement(InvoiceControl,'{
                        dian:gov:co:facturaelectronica:Structures-2-1}
                        InvoiceAuthorization')
36                  AuthorizationPeriod = SubElement(InvoiceControl,'{dian
                        :gov:co:facturaelectronica:Structures-2-1}
                        AuthorizationPeriod')
37                  #FAB07
38                  StartDate = SubElement(AuthorizationPeriod,'{urn:oasis
                        :names:specification:ubl:schema:xsd:
```

```
                    CommonBasicComponents-2}StartDate' )
39                  StartDate.text=self.dataJson['FAB07']
40
41                  #FAB08
42                  EndDate = SubElement(AuthorizationPeriod,'{urn:oasis:
                        names:specification:ubl:schema:xsd:
                        CommonBasicComponents-2}EndDate' )
43                  EndDate.text=self.dataJson['FAB08']
44          ...
```

De la línea 17 a la 25 se declaran los 'NameSpace' de los componentes que se utilizarán en el documento XML. De la línea 18 en adelante se construye el documento XML campo por campo de la factura electrónica.

El archivo 'modells.py' crea la clase 'invoice' cuyos atributos corresponden a los campos de la facturación electrónica solicitada por la DIAN.

```
1           class invoice(models.Model):
2           _name = 'invoiceql.invoice'
3           _description = 'invoiceql.invoiceql'
4
5           FAA01 = fields.Char()
6           # Field: InvoiceAuthorization
7           # Note: Numero autorizacion: Numero del codigo de la resolucion
                otorgada para la numeracion
8           FAB05 = fields.Char(default=lambda self: self._default_FAB05())
9
10          # Field: StartDate
11          # Note: Fecha de inicio de la autorizacion de la numeracion
12          FAB07 = fields.Char(default=lambda self: self._default_FAB07())
13
14          # Field: EndDate
15          # Note: Fecha final de la autorizacion de la numeracion
16          FAB08 = fields.Char(default=lambda self: self._default_FAB08())
17
18          # Field: Prefix
19          # Note: Prefijo de la autorizacion de numeracion de facturacion dado
                por el SIE de Numeracion
20          FAB10 = fields.Char(default=lambda self: self._default_FAB10())
21
22          # Field: From
23          # Note: Valor inicial del rango de numeracion otorgado
24          FAB11 = fields.Char(default=lambda self: self._default_FAB11())
25
26          # Field: To
27          # Note: Valor final del rango de numeracion otorgado
28          FAB12 = fields.Char(default=lambda self: self._default_FAB12())
29  ...
```

El interprete de invoiceQL (escrito con Xtend) genera comentarios acerca de cada campo indicando el nombre e identificador de acuerdo con la documentación de la DIAN.

En la figura **3-22** se muestra el módulo invoiceQL una vez se despliega en ODOO.

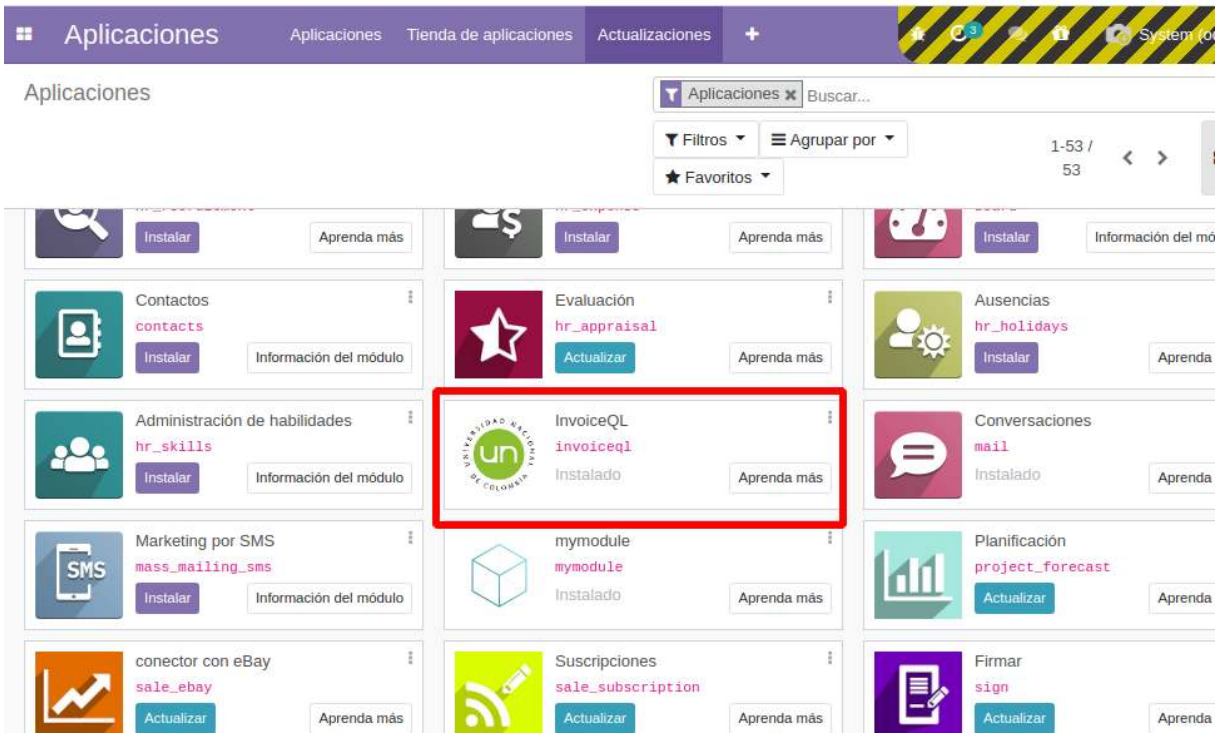

Figura 3-22.: Módulo invoiceQL en ODOO. Fuente: Propia.

Odoo genera un CRUD de forma automática basado en el modelo. En la figura **3-23** se muestra un ejemplo de formulario generado por ODOO.

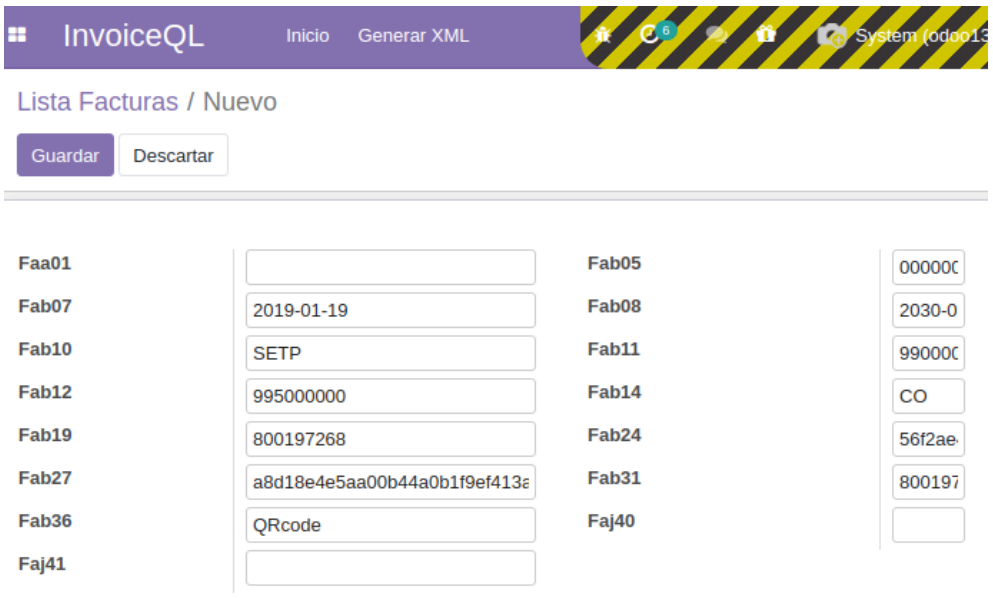

Figura 3-23.: Formulario generado por ODOO. Fuente: Propia.

En la lista de registros de facturas hay un 'link' (como se muestra en la figura **3-24**) que ejecuta las instrucciones del controlador el cual a su vez utiliza la clase 'InvoiceQlBuilder' para generar el documento XML que representa la factura electrónica.

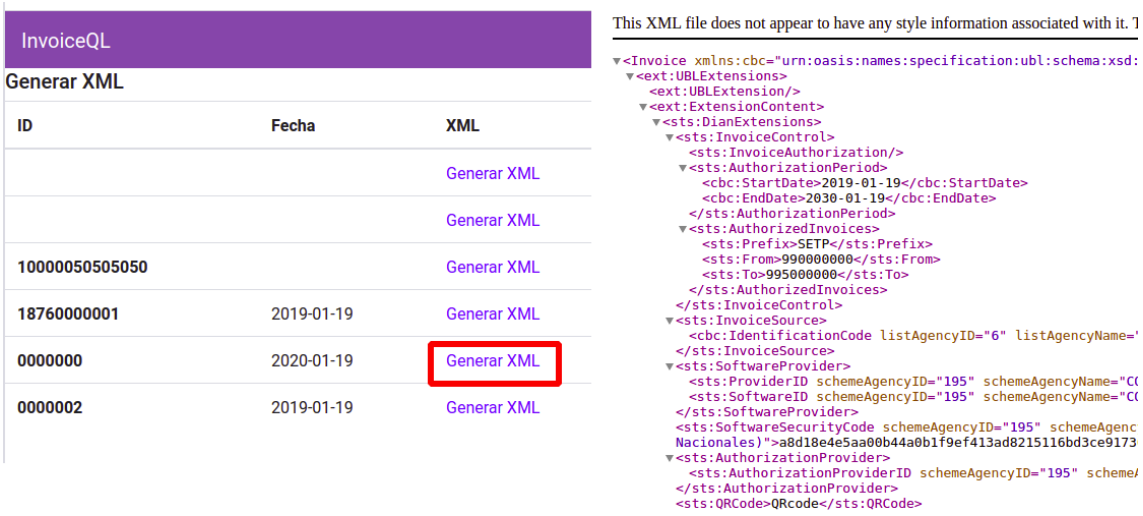

Figura 3-24.: Factura electrónica generada en ODOO. Fuente: Propia.

4. Evaluación

4.1. Análisis Conceptual

Para la evaluación conceptual se contrastan los elementos obtenidos durante el proceso de construcción de InvoiceQL con algunas características de la factura electrónica de acuerdo al manual técnico de la DIAN [25], como se muestra en la tabla **4-1**. En las columnas hay 5 elementos que se obtuvieron con la aplicación de la metodología: 1) el metamodelo, 2) el DSL, 3) la transformación directa (generación del XML mediante plantilla), 4) transformación indirecta (generación de código fuente en python teniendo en cuenta únicamente los elementos del dominio) y 5) la integración con el ERP ODOO.

Las filas de la tabla indican 7 características: 1) Elementos UBL: indica si los elementos de las columnas contemplan los diferentes componentes de UBL. 2) Generar Factura: indica si los elementos de las columnas tienen como resultado final generar la factura electrónica, 3) Firma digital: la firma digital es un elemento muy importante en la factura electrónica cuando esta es enviada a la DIAN para que sea legalizada, sin embargo, en este trabajo nos limitamos únicamente a generar la factura electrónica, 4) Campos de la factura: esta característica tiene que ver con si los elementos incluyen los campos requeridos para la factura electrónica, 5) Restricciones: indica si los elementos tienen en cuanta las restricciones de valor que pueden tener los diferentes campos, 6) Elementos XML: sirve para mostrar si los elementos en las columnas consideran la estructura de los documentos XML y 7) Generación en GPL: para indicar si los elementos de las columnas contemplan la generación de código fuente en lenguajes de programación de propósito general.

	Metamodelo	DSL	T. Directa	T. Indirecta	ODOO
Elementos UBL	X	X	X	X	X
Generar factura	X	X	X	X	X
Firma digital	X	X			
Campos de factura	X	X	X	X	X
Restricciones	X	X	X	X	X
Elementos XML	X	X		X	X
Generación en GPL				X	X

Tabla 4-1.: Evaluación Conceptual. Fuente: Propia.

De la evaluación mostrada en la tabla **4-1** se puede afirmar que:

- Los elementos de UBL, la generación de la factura y los campos completos de la factura electrónica son características que están presentes en todos los elementos evaluados.
- A pesar de que la firma digital y el envío de la factura no hacen parte de los alcances de este proyecto tanto el metamodelo como el DSL contemplan esta característica dado que permiten la creación de *funciones* y se puede implementar una función para la firma digital del documento.
- Dado el enfoque de este trabajo es de *desarrollo de funcionalidades dirigido por modelos* ni el metamodelo ni el DSL se enfocan en generar código fuente en un GPL pues se diseñaron para generar directamente la factura electrónica. Sin embargo, se puede generar código fuente en un GPL como python mediante la estrategia de transformación indirecta.

4.2. Resultado final respecto a la implementación de referencia

Utilizando la funcionalidad *diff* del conocido sistema de repositorios distribuidos *Git* se realizó una comparación entre la implementación de referencia (la factura electrónica de demostración suministrada por la DIAN) y una factura generada por el programa escrito con Xtend que sigue las reglas definidas en las transformaciones.

```
diff --git a/fac-generada.xml b/fac-generada.xml
index 0d9204e..4b569fc 100644
--- a/fac-generada.xml
+++ b/fac-generada.xml
@@ -5,15 +5,15 @@
             <ext:ExtensionContent>
                 <sts:DianExtensions>
                     <sts:InvoiceControl>
-                        <sts:InvoiceAuthorization>18760000001</sts:InvoiceAuthorization>
+                        <sts:InvoiceAuthorization>18760000002</sts:InvoiceAuthorization>^M
                         <sts:AuthorizationPeriod>
                             <cbc:StartDate>2019-01-19</cbc:StartDate>
                             <cbc:EndDate>2030-01-19</cbc:EndDate>
                         </sts:AuthorizationPeriod>
                         <sts:AuthorizedInvoices>
                             <sts:Prefix>SETP</sts:Prefix>
-                            <sts:From>990000000</sts:From>
-                            <sts:To>995000000</sts:To>
+                            <sts:From>990000011</sts:From>^M
+                            <sts:To>995000012</sts:To>^M
                         </sts:AuthorizedInvoices>
                     </sts:InvoiceControl>
                     <sts:InvoiceSource>
```

Figura 4-1.: Ejemplo de un cambio detectado con Git-Diff entre la implementación de referencia y una factura generada con invoceQL. Fuente: Propia.

En la figura **4-1** se resaltan dos diferencias típicas que se presentan entre la implementación de referencia y una factura generada con invoiceQL, estas diferencias tienen que ver con los cambios en los valores de los campos (en rojo el valor en la implementación de referencia y en verde el valor en la factura generada), es decir, hay diferencias porque los valores de cada factura dependen de la información de la transacción comercial.

Figura 4-2.: Resumen con Git-Diff de cambios entre la implementación de referencia y una factura generada con invoceQL. Fuente: Propia.

Al ejecutar el comando *git diff –compact-summary* como se muestra en la figura **4-2** se presentan 23 nuevos elementos que en su mayoría corresponden a campos agregados y necesarios para esta factura en particular; tambien se reportan 67 elementos eliminados que corresponden a lineas con información acerca de la firma digital del documento que como se mencionó en el anterior apartado no hacía parte de los alcances de este proyecto.

En términos generales el documento XML generado con invoiceQL es equivalente al XML de la implementación de referencia, y las únicas diferencias encontradas son los cambios en los valores de los campos (que dependen de cada transacción comercial particular) y la ausencia de la firma digital.

5. Conclusiones y recomendaciones

5.1. Conclusiones

Se desarrolló un lenguaje de dominio específico llamado InvoiceQL que con la ayuda de un programa interprete escrito en Xtend es capaz de generar facturas electrónicas (de forma directa e indirecta) siguiendo el estándar UBL y de acuerdo a los requerimientos técnicos solicitados por la DIAN.

El enfoque MDSD y la metodología utilizada permitió construir el lenguaje InvoiceQL, dicha metodología tiene las etapas necesarias para construir una herramienta funcional y útil para generar facturas electrónicas. El metamodelo es el artefacto de abstracción más importante pues de este se derivan las reglas gramaticales del DSL y las diferentes transformaciones.

Se implementaron dos estrategias para generar las facturas electrónicas. La forma directa permite que instrucciones escritas con invoiceQL genere facturas electrónicas de forma inmediata. La forma indirecta permite generar código fuente en un lenguaje de propósito general(en este caso en lenguaje python) que puede generar facturas electrónicas. Esta forma indirecta facilita en gran medida la implementación de la factura electrónica en un sistema de información y de igual forma facilita el mantenimiento del código pues ante los cambios en los requerimientos de la factura electrónica se pueden modificar las instrucciones en el DSL y volver a generar el código.

5.2. Trabajo futuro

Algunas ideas que pueden desarrollarse en el futuro con base en este trabajo pueden ser:

- Implementar los documentos electrónicos solicitados por la DIAN diferentes a la factura (Invoice). Nota Crédito (CreditNote), Nota Débito (DebitNote), Contenedor de documentos (AttachedDocument), y Registro de eventos (ApplicationResponse).
- Desarrollar un interprete que sea capaz de generar y enviar (a la DIAN) documentos electrónicos.
- Desarrollar un programa interprete de invoiceQL y con la ayuda de un motor de base de datos(puede ser NoSQL) permita almacenar y consultar facturas electrónicas.
- Desarrollar un sistema de inteligencia de negocios que permita aprovechar la información contenida en los documentos electrónicos, por ejemplo con fines comerciales. Para consultar y agrupar información puede ampliarse el lenguaje invoiceQL.

A. Implementación de referencia

Implementación de referencia de la factura electrónica.
(Puede consultarse en la siguiente URL)

https://github.com/edwarhub/invoiceql-unal/blob/main/FacturaGenerica.xml

B. Metamodelo

Metamodelo completo y claro.
(Puede consultarse en la siguiente URL)

https://github.com/edwarhub/invoiceql-unal/blob/main/metamodelo.png

C. Script ejemplo en InvoiceQL

Script de ejemplo escrito en InvoiceQL
(Puede consultarse en la siguiente URL)

https://github.com/edwarhub/invoiceql-unal/blob/main/invoice-example1.inv

Bibliografía

[1] Hira, A., & Boehm, B. (2016). Function Point Analysis for Software Maintenance. International Symposium on Empirical Software Engineering and Measurement, 08-09-Sept. https://doi.org/10.1145/2961111.2962613

[2] Andrew Ko, Brad Myers, Michael Coblenz, and Htet Aung. 2006. An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks. IEEE Transactions on Software Engineering IIEEE Trans. Software Eng. 32, 12 (2006), 971- 987. DOI= https://dx.doi.org/10.1109/tse.2006.116

[3] Ali, S. S., Shoaib Zafar, M., & Saeed, M. T. (2020, January 1). Effort Estimation Problems in Software Maintenance - A Survey. 2020 3rd International Conference on Computing, Mathematics and Engineering Technologies: Idea to Innovation for Building the Knowledge Economy, ICoMET 2020. https://doi.org/10.1109/iCoMET48670.2020.9073823

[4] Erdweg, S., Fehrenbach, S., & Ostermann, K. (2014). Evolution of software systems with extensible languages and DSLs. IEEE Software, 31(5), 68–75. https://doi.org/10.1109/MS.2014.99

[5] DIAN - Página de Resoluciones. (n.d.). Última visita Julio 17, 2020, url https://www.dian.gov.co/normatividad/Paginas/Resoluciones.aspx.

[6] Cuylen, A., Kosch, L., & Breitner, M. H. (2016). Development of a maturity model for electronic invoice processes. Electronic Markets, 26(2), 115–127. https://doi.org/10.1007/s12525-015-0206-x

[7] Nalcaci, I. G. (2017, July 25). Technical and communicational standards of e-invoicing: A Country Example: Turkey. Application of Information and Communication Technologies, AICT 2016 - Conference Proceedings. https://doi.org/10.1109/ICAICT.2016.7991785

[8] Koch Billentis, B. (2019). The e-invoicing journey 2019-2025. www.billentis.com

[9] DIAN, Factura Electrónica. Visitado 20 de Julio de 2020, Url: https://www.dian.gov.co/impuestos/factura-electronica/Paginas/inicio.aspx

[10] OASIS Open Standard (2013). Universal Business Language Version 2.1. Visitado 21 de Julio de 2020, Url: http://docs.oasis-open.org/ubl/UBL-2.1.html

[11] Marco Brambilla, Jordi Cabot, M. W. (2012). Model-Driven Software Engineering in Practice (Morgan &Claypool (ed.)). Morgan &Claypool.

[12] Chavarriaga, E., Jurado, F., & Díez, F. (2017). An approach to build XML-based domain specific languages solutions for client-side web applications. Computer Languages, Systems and Structures, 49, 133–151. https://doi.org/10.1016/j.cl.2017.04.002.

[13] Voelter, M. (2013). DSL Engineering: Designing, Implementing and Using Domain-Specific Languages. dslbook.org. dslbook.org

[14] Bettini, L. (2016). Implementing Domain-Specific Languages with Xtext and Xtend. In Packt Publishing. http://www.packtpub.com/implementing-domain-specific-languages-with-xtext-and-xtend/book

[15] Thomas Stahl, Markus Voelter, K. C. (2006). Model-Driven Software Development: Technology, Engineering, Management. 1, 6–8. https://doi.org/10.16309/j.cnki.issn.1007-1776.2003.03.004

[16] ODOO S.A. Odoo. Odoo Apps Store. Visitado Agosto 4, 2020, URL https://apps.odoo.com/apps

[17] Matus, A., Guerra, E., Fuertes, W., Gómez, M., Aules, H., Villacís, C., & Toulkeridis, T. (2017). On the development of an electronic invoicing solution to integrate SMEs with a tax-collection egovernment-platform. 2017 4th International Conference on EDemocracy and EGovernment, ICEDEG 2017, 94–101. https://doi.org/10.1109/ICEDEG.2017.7962518

[18] Cuylen, A., Kosch, L., & Breitner, M. H. (2016). Development of a maturity model for electronic invoice processes. Electronic Markets, 26(2), 115–127. https://doi.org/10.1007/s12525-015-0206-x

[19] Sharma, L., Rane, C., Puro, J., & Nimkar, A. V. (2020, February 1). ERPL: A Language for Structuring Business Processes in ERP Systems. International Conference on Emerging Trends in Information Technology and Engineering, Ic-ETITE 2020. https://doi.org/10.1109/ic-ETITE47903.2020.157

[20] Makrickienė, N., Gudas, S., & Lopata, A. (2019). Ontology and enterprise modelling driven software requirements development approach. Baltic Journal of Modern Computing, 7(2), 190–210. https://doi.org/10.22364/bjmc.2019.7.2.02

[21] Vogelgesang, T., Kaufmann, J., Becher, D., Seilbeck, R., Geyer-Klingeberg, J., Klenk, M., & Polyvyanyy, A. (2019). Celonis PQL: A Query Language for Process Mining Process Querying Methods.

[22] Vargara J. Andrés. (2017). A model-driven deployment approach for applying the performance and scalability perspective from a set of software architecture styles. Universidad Nacional de Colombia. Bogotá - Colombia.

[23] Jhon Alexander Cruz Castelblanco. (2014). A modular model-driven engineering approach to reduce efforts in software development teams.

[24] Neeraj, K. R., Janardhanan, P. S., Francis, A. B., & Murali, R. (2017, October 31). A domain specific language for business transaction processing. 2017 IEEE International Conference on Signal Processing, Informatics, Communication and Energy Systems, SPICES 2017. https://doi.org/10.1109/SPICES.2017.8091270

[25] DIAN, Dirección de impuestos y aduanas nacionales de Colombia. (2020). Resolución Número 000042 del 05 de mayo de 2020.

[26] DIAN Presentación Informativa. (2020). Factura Electrónica Fácil, Eficiente y Transparente. https://www.dian.gov.co/impuestos/factura-electronica/correo-recepcion-facturas/Documents/Presentacion-Recepcion-de-Facturas-Electronicas.pdf

[27] W3C. (2008). Extensible Markup Language (XML) 1.0 (Fifth Edition). https://www.w3.org/TR/xml/

[28] Wood, L. (1999). Programming the web: The W3C DOM Specification. IEEE Internet Computing, 3(1), 48–54. https://doi.org/10.1109/4236.747321

[29] ODOO S.A. (2020). Building a Module — odoo 13.0 documentation. Retrieved September 7, 2020, from https://www.odoo.com/documentation/13.0/howtos/backend.html

[30] Pons, C., Giandini, R., & Pérez, G. (2010). Desarrollo de Software dirigido por modelos: Conceptos teóricos y su aplicación práctica. Pag. 32.