APLICACIÓN DE INGENIERÍA DIRIGIDA POR MODELOS PARA LA CONSTRUCCIÓN DE UN PROTOTIPO DE HERRAMIENTA DE MODELADO DE DOMINIO ESPECÍFICO (DSM) PARA MODULOS DE SISTEMAS DE PLANIFICACIÓN DE RECURSOS EMPRESARIALES EN ODOO

Andrés Hamir Cobos Prada

Andrea Stefania Sicachá Arenas

Facultad de Ingeniería, Proyecto Curricular de Ingeniería de Sistemas

Bogotá D.C., Colombia

2016

APLICACIÓN DE INGENIERÍA DIRIGIDA POR MODELOS PARA LA CONSTRUCCIÓN DE UN PROTOTIPO DE HERRAMIENTA DE MODELADO DE DOMINIO ESPECÍFICO (DSM) PARA MODULOS DE SISTEMAS DE PLANIFICACIÓN DE RECURSOS EMPRESARIALES EN ODOO

Andrés Hamir Cobos Prada
Andrea Stefania Sicachá Arenas

Trabajo de grado presentado como requisito parcial para optar al
Título de:
Ingeniero de Sistemas

Director:
Dr. Carlos Enrique Montenegro Marín

Línea de Investigación:
Ingeniería de Software

Universidad Distrital Francisco José de Caldas

Facultad de Ingeniería

Bogotá D.C, Colombia

2016

Nota de aceptación:

_____

_____

_____

_____

_____

_____

Firma del Director del Trabajo de Grado

_____

Firma del jurado

_____

Bogotá, 2016

Dedicatoria

A nuestros padres por estar siempre presentes,

Por su apoyo incondicional y por su infinito amor

Andrés Hamir Cobos Prada,

Andrea Stefania Sicachá Arenas

Agradecimientos

A nuestros padres porque siempre han sido parte de nuestras vidas y nos han ofrecido su amor, sus consejos, su apoyo y su cariño.

A nuestros abuelos que son como unos segundos padres siempre pendientes, amorosos y preocupados por nosotros.

A nuestros tíos por su apoyo y colaboración.

A nuestro director de proyecto de grado el Dr. Carlos Enrique Montenegro Marín por sus enseñanzas y su acompañamiento durante el proceso de creación de este proyecto de grado.

A la universidad Distrital Francisco José de Caldas por la educación y formación recibidas a lo largo de nuestras carreras.

Queremos darles nuestros más sinceros agradecimientos.

Resumen

En los últimos años la mayoría de las empresas han optado por digitalizar su información y por automatizar sus procesos, debido a esto nacieron los sistemas de planeación de recursos mejor conocidos como ERP, sistemas que tienen como objetivo ayudar a las compañías a organizar sus recursos y sus procesos internos tales como nomina, relaciones con los clientes (CRM), inventarios, entre otros.

Sin embargo como es demostrado por el trabajo de Siswanto [1] según los requerimientos de la empresa, se deben tener en cuenta los módulos a utilizar y desarrollar dependiendo de la necesidad. De manera que algunas de las funcionalidades ofrecidas por ERP pueden no ser útiles para algunas compañías. Esta es la razón por la cual se decidió manejar una arquitectura modular así el cliente puede adquirir solo los módulos que realmente requiere y estos a su vez son personalizables.

Uno de los sistemas ERP más importantes actualmente en el mercado es Odoo ya que además de ser libre es estable y su arquitectura permite que se adapte a las necesidades del cliente ya que esta funciona de manera modular. Sin embargo un inconveniente es que el desarrollo de estos módulos tiende a ser una tarea trivial y repetitiva que podría automatizarse implementando una propuesta basada en ingeniería dirigida por modelos (MDE).

Este proyecto propone un DSL gráfico para el diseño de Módulos ERP para el sistema Odoo basado en la Ingeniería Dirigida por Modelos (MDE) específicamente en la Arquitectura Dirigida por Modelos (MDA) parte desde la capa M3 que es la capa de abstracción más alta hasta llegar a la capa M0 y plantea un meta-modelo que puede llegar a ser utilizado en un futuro para otros sistemas ERP diferentes a Odoo.

Palabras clave: ***Ingeniería Dirigida por Modelos (MDE), Arquitectura Dirigida por Modelos (MDA), Planeación de Recursos Empresariales (ERP), Odoo***

Abstract

In recent years most companies has chosen to digitize their information and automate their processes. Because of this the Enterprise resource planning (ERP) was born. The ERP main goal is to help the companies by organizing their resources and intern processes such as payroll, customer relationship management, inventory between others.

However is logic that not all companies are the same so they cannot be managed in the same way. That's the reason why some ERP features may be useless for some companies. Because of this ERP make the decision to work in a modular way then the client can just acquire the necessary modules and those are customizable.

Currently Odoo is an important ERP system in the market. That's because Odoo is either free or stable and also its architecture let it to adapt to the client needs. Like it was saying before Odoo works in a modular way. However a problem around the development of Odoo modules is that can be both trivial and also repetitive but there is a way to automate it by implementing a purpose based on model driven engineering (MDE)

This project proposes a graphic DSL to design ERP modules for the Odoo system. It is based on model driven engineering (MDE) more specifically on model driven architecture (MDA). the project goes from the M3 layer (highest abstraction layer) to the M0 layer(lowest abstraction layer).the project also pose a meta-model that can be used for other ERP systems besides Odoo in the future.

Key words: Model Driven Engineering (MDE), Model Driven Architecture (MDA), enterprise resource planning (ERP), Odoo

Tabla de Contenido

Introducción 14
1. Presentación 15
Problema 16
Objetivos 17
Objetivo General 17
Objetivos Específicos 17
Marco Teórico 18
MDE 18
MDA 19
ERP 21
ODOO 23
2.Selección del meta-metamodelo 26
EMF 26
Componentes de ecore 28
MOF 30
EMOF 31
CMOF 37
Elección del Meta-metamodelo 42
Factores decisivos para la elección del meta-metamodelo 42
3.Diseño del metamodelo 44
Obtención de requerimientos 44
Módulos: 45
Modelos 46
Relaciones 47
Vistas: 47
Abstracción 47
Metadatos 48
Definición de Atributos 49
Relaciones 50
Implementación del meta modelo de módulos del sistema Odoo: 51
Paso a paso proceso de implementación: 51
Resultado final del meta-modelado 64
4.Diseño del DSL gráfico 65

GMF 65
Sirius 68
Proceso con Sirius 70
Creación del modelo generador de código 71
Creación del viewpoint 76
Configuración de la representación de nodos y aristas en el viewpoint 80
Configuración de la barra de herramientas en el viewpoint 92
Transformación del modelo en módulos para el sistema Odoo 104
5.Diseño de un módulo Odoo con OdooDSL 107
Descripción del requerimiento 107
Abstracción de elementos requeridos 107
Modulo Clientes 107
Modulo Ventas 108
Modelado dentro de la herramienta OdooDsl 108
6.Prueba de eficiencia del DSL gráfico 110
Metodología de prueba 110
Prueba 110
Desarrolladores 110
Enunciado del requerimiento 111
Tiempos esperados 111
Resultados 112
Análisis Estadístico 115
Datos 115
Análisis 116
Conclusiones y Trabajo Futuro 118
Bibliografia 120

Lista de Ilustraciones

Ilustración 1 arquitectura genérica de un sistema ERP. Fuente: sistemas de planificación de recursos empresariales: un caso real. 23
Ilustración 2 Jerarquia de los componentes de ecore. Fuente: Eclipse ORG, Package org.eclipse.emf.ecore 28
Ilustración 3 Interface EObject de EMF. Fuente: Eclipse ORG, “Package org.eclipse.emf.ecore. 30
Ilustración 4Modelo EMOF. Fuente: OMG Meta Object Facility(MOF) core specification 32
Ilustración 5 Tipos de datos. EMOF Fuente: OMG Meta Object Facility (MOF) core specification 33
Ilustración 6 Paquete EMOF. Fuente: OMG Meta Object Facility (MOF) core specification 33
Ilustración 7 paquetes CMOF. Fuente: OMG Meta Object Facility(MOF) core specification. 38
Ilustración 8 Creación del Archivo ECORE. Fuente: Elaboración Propia 51
Ilustración 9 Archivo ECORE presente en el entorno. Fuente: Elaboración Propia 52
Ilustración 10 Configuración metadatos paquete Ecore. Fuente: Elaboración Propia 52
Ilustración 11 Metamodelado de la Eclass ModuloOdoo. Fuente: Elaboración Propia 54
Ilustración 12 Metamodelado EClass ModeloOdoo. Fuente: Elaboración Propia 54
Ilustración 13 Metamodelado ECLass AtributoOdoo. Fuente: Elaboración Propia 55
Ilustración 14 Metamodelado del Enum "AtributoListaOdoo". Fuente: Elaboración Propia 57
Ilustración 15 Metamodelado de la EClass RelacionOdoo. Fuente: Elaboración Propia 58
Ilustración 16 Metamodelado de ENum "tipoRelacionOdoo". Fuente: Elaboración Propia 59
Ilustración 17 Metamodelado de la relación del módulo con sus modelos. Fuente: Elaboración Propia 60
Ilustración 18 Metamodelado de la relación de un módulo con sus modelos. Fuente: Elaboración Propia 61
Ilustración 19 Metamodelado de relación AtributoOdoo a AtributoListaOdoo. Fuente: Elaboración Propia 61
Ilustración 20 Metamodelado RelacionOdoo a TipoRelacionOdoo. Fuente: Elaboración Propia 62
Ilustración 21 Metamodelado RequerimientoOdoo. Fuente: Elaboración Propia 63
Ilustración 22 Ecore después de finalizare el proceso de Metamodelado. Fuente: Elaboración Propia. 64
Ilustración 23 Proceso para la implementación de un editor gráfico. Fuente: Raising the Level of Abstraction in the Development of GMF-based Graphical Model Editors 66
Ilustración 24 Estructura Archivo aird. Fuente: Elaboración Propia 71
Ilustración 25 Diagrama ade Eclass de odoo8. Fuente: Elaboración Propia 72
Ilustración 26 Configuracion Genmodel. Fuente: Elaboración Propia 73
Ilustración 27 Configuración paquete Odoo8 del archivo odoo8.genmodel. Fuente: Elaboración Propia 73
Ilustración 28 Proyectos Generados a partir del modelo generador de código. Fuente: Elaboración Propia 74
Ilustración 29 Estructura Proyecto de Prueba. Fuente: Elaboración Propia 75
Ilustración 30 Estructura del archivo testing.odoo8. Fuente: Elaboración Propia 75
Ilustración 31 Prueba del modelo Generador. Fuente: Elaboración Propia 75
Ilustración 32 Viewpoint specification project. Fuente: Elaboración Propia 77
Ilustración 33 Configuración de las dependencias. Fuente: Elaboración Propia 78

Ilustración 34 Configuraciones del viewpoint. Fuente: Elaboración Propia 79
Ilustración 35configuracion de la representación tipo Diagrama. Fuente: Elaboración Propia 79
Ilustración 36 Configuraciones Nodo NModulosOdoo. Fuente: Elaboración Propia 81
Ilustración 37 Configuraciones del estilo para el nodo NModuloOdoo. Fuente: Elaboración Propia .. 81
Ilustración 38 Configuraciones Nodo NModeloOdoo. Fuente: Elaboración Propia 82
Ilustración 39 Configuraciones del estilo para el nodo NModeloOdoo. Fuente: Elaboración Propia .. 83
Ilustración 40 Configuraciones Nodo NAtributoOdoo. Fuente: Elaboración Propia 83
Ilustración 41Configuraciones del estilo para el nodo NAtributoOdoo. Fuente: Elaboración Propia .. 84
Ilustración 42 Viewpoint con nodos configurados. Fuente: Elaboración Propia 84
Ilustración 43Configuraciones del estilo para el nodo NAtributoOdoo. Fuente: Elaboración Propia .. 86
Ilustración 44 Configuración de relación modulo_tiene_modelos. Fuente: Elaboración Propia 87
Ilustración 45configuraciones de la EClass RelacionOdoo. Fuente: Elaboración Propia 88
Ilustración 46 Configuración estilos condicionales. Fuente: Elaboración Propia 90
Ilustración 47configgguracion de visualización del viewpoint. Fuente: Elaboración Propia 91
Ilustración 48Prueba Visualización Viewpoint. Fuente: Elaboración Propia 92
Ilustración 49 Estructura de la herramienta de creación de modulo. Fuente: Elaboración Propia 94
Ilustración 50 estructura de la herramienta de creación de Modelo. Fuente: Elaboración Propia 96
Ilustración 51 Estructura de la herramienta de creación de modelo. Fuente: Elaboración Propia..... 97
Ilustración 52 Estructura herramienta creación relación Modulo - Modelo. Fuente: Elaboración Propia 98
Ilustración 53 Estructura Herramienta creación Modelo - Atributo. Fuente: Elaboración Propia 100
Ilustración 54 Código creación de relaciones. Fuente: Elaboración Propia..... 101
Ilustración 55 Seccion de herramientas del DSL. Fuente: Elaboración Propia 103
Ilustración 56 Visualización y barra de Herramientas del proyecto. Fuente: Elaboración Propia..... 104
Ilustración 57 flujo de funcionamiento del script. Fuente: Elaboración propia 105
Ilustración 58 Ejecución del Script. Fuente: Elaboración propia 105
Ilustración 59 Estructura de datos generada por el script..... 105
Ilustración 60 Modelo representante del requerimiento. Fuente: Elaboración Propia 109
Ilustración 61 Resultado de la transformación. Fuente: Elaboración Propia 109
Ilustración 62 gráfica desarrollo manual vs OdooDSL 116
Ilustración 63 representaciòn de la regla de 3 para calcular el porcentaje de tiempo de desarrollo con OdooDSL..... 117
Ilustración 64 Regla de 3 117

Lista de tablas

Tabla 1 Descripción componentes de ECORE Fuente: elaboración propia Realizado con información sacada de: Eclipse ORG, "Package org.eclipse.emf.ecore"……27

Tabla 1 tiempos de desarrollo de los módulos de prueba con OdooDSL, sacada de: Elaboración propia……110

Tabla 3 Resultados en tiempo de la prueba aplicada al desarrollador José Javier Vargas sacada de: Elaboración propia……110

Tabla 4 Resultados en tiempo de la prueba aplicada al desarrollador Cinxgler Mariaca Minda sacada de: Elaboración propia……110

Tabla 5 Resultados en tiempo de la prueba aplicada al desarrollador Cristian Alejandro Rojas Quintero sacada de: Elaboración propia……110

Tabla 6 Resultados en tiempo de la prueba aplicada al desarrollador Steven Sierra Forero sacada de: Elaboración propia……110

Tabla 7 Resultados en tiempo de la prueba aplicada al desarrollador Cristian Chaparro Africano sacada de: Elaboración propia……110

Tabla 8 Resumen tiempos de desarrollo con y sin la herramienta OdooDSL……116

Introducción

En este documento se presenta el diseño y la implementación de un Lenguaje de Dominio Especifico (DSL) gráfico para el desarrollo de módulos ERP en el sistema Odoo. La motivación principal para el diseño e implementación de este DSL es que le permitirá a los desarrolladores de Módulos Odoo ahorrar tiempo de desarrollo y mejorar sus prácticas. La primera parte de este documento contempla los objetivos del proyecto, el origen de la problemática y un marco teórico para referenciar al lector.

El segundo capítulo consta de la elección del meta-meta modelo del cual se partirá para el diseño del metamodelo y de las capas subsiguientes , para dicha elección se contemplaron dos meta-metamodelos Ecore y MOF, en este capítulo encontraremos la descripción de cada uno de ellos y los criterios que se tuvieron en cuenta para la elección.

En el tercer capítulo se puede observar el diseño del meta-modelo paso a paso, es decir la abstracción que tuvo que llevarse a cabo para el diseño, la selección de las características más importantes, la definición de los objetos y las relaciones respectivamente y finalmente el proceso de modelamiento en el meta-metamodelo elegido en el capítulo anterior.

El cuarto capítulo contempla el diseño del DSL gráfico a partir del meta modelo antes generado, se mostrará paso a paso las herramientas utilizadas y las características configuradas para la creación del DSL gráfico.

El capítulo quinto contiene el desarrollo de un módulo ERP de Odoo hecho con la herramienta desarrollada a lo largo de este proyecto y tiene como objetivo demostrar el funcionamiento adecuado de la misma.

Finalmente el ultimo capitulo se encuentran una prueba de rendimiento hecha con varios desarrolladores ajenos al desarrollo de este proyecto.

1 Presentación

Odoo es una suite moderna de aplicaciones para negocios lanzada bajo la licencia AGLP (Affero General Public License) que es una licencia de software libre, esta suite ofrece módulos como CRM (Gestión de relaciones con el cliente), HR (recursos humanos), ventas, contabilidad, fabricación, gestión de almacenes, gestión de proyectos entre otros.

La ERP de Odoo funciona de manera modular y su modelo de negocio se basa en la prestación de servicios ya que como se dijo anteriormente su código es libre, los servicios prestados por la plataforma ERP de Oddo son básicamente migraciones, soporte, formación y desde la versión 6.0 también se presta el servicio de software as a service, es decir el software en la nube.

Un módulo de Odoo es independiente de los otros y es adaptable a las necesidades de cada compañía, dichos módulos se encuentran desarrollados en Python y son carpetas con una estructura predefinida que contienen archivos XML y .PY. Los módulos definen la estructura de los datos, formularios, informes, menús, procedimientos y flujos de trabajo entre otras cosas.

El inicio de la creación de un módulo de Odoo contempla el modelado de los atributos del mismo, este modelado se transforma en código Python utilizando como base las especificaciones de desarrollo de Odoo para cada tipo de atributo, es decir, una vez se tiene el modelo del módulo, la tarea de convertirlo en código es una tarea sistemática.

Por otra parte, cuando las especificaciones de desarrollo de módulos del sistema Odoo cambian, no debería significar la recodificación manual de los módulos creados, el modelo inicial del sistema debe ser capaz de soportar dicha migración.

Los estándares de codificación son un aspecto importante en la calidad de un sistema, asegurar dichos estándares puede llegar a ser una tarea que conlleve procesos extra en la creación de un módulo para Odoo, una herramienta automatizada debe ser capaz de generar el código del módulo de una manera automática

Problema

En la actualidad se ofrecen diversas soluciones de software para la planificación de recursos empresariales (ERP), en su mayoría dichas soluciones son diseñadas como un conjunto de elementos arquitectónicos tales como módulos, capas, clases y servicios, cada uno de estos elementos cumple con una función específica dentro del sistema[2].

Programar funcionalidades para Odoo la mayor parte del tiempo significa la creación de módulos propios [3], La programación de estos módulos hasta la fecha debe hacerse de forma manual o mediante el uso de programas de terceros que no están soportados por la arquitectura dirigida por modelos.

Además de lo mencionado anteriormente, como se evidencia en el trabajo de Agren[4], los pasos apropiados para la generación de software contempla 3 pasos iniciales como lo son requerimientos, diseño e implementación, otro inconveniente que se presenta es el caso en el cual el programador pasa directamente de la obtención de requerimientos a la codificación del módulo. Esto puede terminar en una mala interpretación de los requerimientos y en una afectación directa a la lógica del negocio.

También es necesario tener en cuenta que en los sistemas modulares, como enuncia Chang[5] el soporte modular de un sistema consiste en un conjunto de reglas o interfaces que

deben cumplirse para el correcto funcionamiento del software, en este caso, el desarrollo de un módulo requiere que el código se acople a los distintos estándares propuestos por Odoo, esto con el fin de asegurar la comunicación del módulo con el resto del sistema. Dichos estándares son iguales en todos los módulos, de manera que, al escribir uno nuevo, el desarrollador tendrá que escribir el mismo código que ha escrito en versiones anteriores.

De las características antes mencionadas se puede observar que la codificación de un módulo para ERP sobre ODOO en sus partes básicas debería ser trivial, estándar y automatizada. Como no se está realizando de esta forma se incurren en gastos innecesarios de tiempo para los desarrolladores lo cual puede llegar a afectar de manera negativa un proyecto.

Objetivos

Objetivo General

Generar un lenguaje de dominio específico gráfico (DSL) para los módulos de sistemas de gestión empresarial de la herramienta Odoo utilizando la Ingeniería Dirigida por Modelos (MDE) para automatizar la codificación de los mismos.

Objetivos Específicos

- Elegir el Meta-metamodelo que será utilizado para la creación del prototipo de la herramienta propuesta, a través de un análisis de los meta-metamodelos ECORE y MOF y seleccionar el que mejor se ajuste al proyecto.

- Construir un metamodelo teniendo en cuenta las características principales de los módulos ERP de Odoo, que permita el modelado de módulos teniendo en cuenta los parámetros definidos.

- Definir un lenguaje de dominio específico gráfico (DSL) para el modelamiento de módulos ERP de Odoo, a partir del metamodelo construido.

- Construir un módulo ERP para Odoo, comenzando desde la fase de diseño utilizando la herramienta DSL hasta el despliegue del módulo, con el fin de comprobar el funcionamiento de la herramienta.

Marco Teórico

A continuación se presenta un pequeño marco teórico que tiene como objetivo dar una pequeña introducción al lector con respecto a los temas que se consideran claves para el desarrollo de este proyecto.

MDE

La Ingeniería Dirigida por Modelos por sus siglas en inglés MDE, hace referencia a una metodología para el desarrollo de software que se basa en la creación de modelos de dominio, es decir en la representación abstracta de la aplicación que se quiere construir.

MDE evoluciono como un paradigma proveniente de la tecnología orientada a objetos, donde el principio es que todo se constituirá como un objeto. En el paradigma de ingeniería dirigida por modelos se parte del principio de que todo es un modelo. MDE es acerca de modelos, pero es también acerca de relaciones entre el modelo y el objeto de estudio (estos pueden ser artefactos de software o elementos de dominio real, meta-modelos, y transformaciones de los modelos). De acuerdo a Jean-Marrie Favre, MDE es un campo de la ingeniería de sistemas en el cual todo el proceso se basa en el uso de modelos e ingeniería de modelado [6].

MDE surgió como una respuesta a la industrialización del desarrollo de software y tiene como objetivo simplificar el proceso de diseño haciendo más fácil una estandarización según un modelo de dominio específico buscando con esto facilitar la comunicación de todo el equipo y el desarrollo de la aplicación.

Las herramientas MDE imponen restricciones específicas de dominio y hacen revisiones que pueden detectar y prevenir muchos errores al inicio del ciclo de vida de un software [7].

MDA

La arquitectura dirigida por modelos MDA por sus siglas en inglés [8], es un estándar propuesto por la OMG que busca guiar el desarrollo de un producto de software separando este en distintos niveles de abstracción, los cuales se comunican entre sí a través de modelos o artefactos, es decir de cada nivel de abstracción se desprende un modelo que será tomado como guía por el siguiente nivel.

A continuación se encuentran definidos los niveles de abstracción propuestos por el OMG, para dicha definición se tomo como referencia el artículo *Model driven Architecture, applying MDA to enterprise computing* referenciado en [9]:

- M3 Meta-metamodelo: el Meta-metamodelo también conocido como MOF (Meta Object Facility) hace referencia a un lenguaje abstracto y autodefinido para la especificación, construcción y gestión de los meta-modelos independientes de la plataforma. Es una base para definir cualquier lenguaje de modelado, como UML o el mismo MOF. MOF incluso define la columna vertebral para la implementación de un repositorio de metadata, descrito por metamodelo.

- M2 Metamodelo: en el metamodelo encontramos todos los estándares definidos por el usuario en el MOF, un ejemplo de un metamodelo es UML que se entiende como un lenguaje para la especificación, visualización y documentación de sistemas de software.

- M1 Modelo: los modelos del mundo real que son representados por conceptos de un metamodelo definido en la capa M2 se encuentran en el nivel M1.

- M0 Realidad: hace referencia al mundo real, es decir lo que se planea modelar.

DSL

Un lenguaje de dominio especifico o DSL por sus siglas en inglés es un lenguaje de programación o de especificación ejecutable que a través de abstraccione3s y notaciones apropiadas ofrece un poder expresivo centrado y hasta restringido a un problema de dominio particular [10].

De acuerdo al artículo *Domain-Specific Language Design Requires Feature Descriptions* [10] Los DSL ofrecen modelado orientado a un dominio en particular, brindando de esta manera facilidad de modelado con respecto a los GPL (General Purpose Language).

El objetivo de un DSL es representar, resolver y solucionar un problema o situación específicos y la creación de un DSL puede llevarse a cabo, o vale la pena que se lleve a cabo cuando ninguno de los lenguajes existentes puede resolver un problema particular.

Un ejemplo de DSL es el lenguaje SQL que se creó para hacer consultas y transacciones a una base de datos, este lenguaje tiene como único objetivo resolver los problemas de comunicación con la base de datos por lo tanto es un lenguaje creado para resolver un problema particular, es decir es un DSL.

Hay varios tipos de DSL, sin embargo los que son relevantes para este proyecto son los DSL textuales y gráficos, a continuación se explicará cada uno de ellos tomando como base el artículo *"Desarrollo de un lenguaje de dominio específico para sistemas de gestión*

de aprendizaje y su herramienta de implementación "KiwiDSM" mediante ingeniería dirigida por modelos" [11]

DSL Textual

La mayoría de los lenguajes informáticos son textuales y están formados por un conjunto ordenado de sentencias. Un ejemplo muy conocido de DSL textual es SQL utilizado para realizar consultas a una base de datos. Una forma de crear DSLs textuales es mediante la creación de una determinada gramática (por ejemplo utilizando EBNF - Extended Backus-Naur Form) y posteriormente crear o utilizar un convertidor para dicha gramática, que en etapas posteriores puede interpretar el DSL o generar código.

DSL Gráfico

En los últimos años están ganando gran aceptación los lenguajes gráficos, podrían citarse como ejemplo UML. La creación de un lenguaje grafico es similar a la de un lenguaje textual, la única diferencia es que en lugar de usar texto para representarlos conceptos, se utilizan conectores y figuras simples.

ERP

Un sistema de planeación de recursos empresariales o ERP tiene como objetivo gestionar cada una de las áreas de una empresa de manera sistematizada y cuyas características principales son ser modulares y configurables. Se dice que un ERP es modular porque se compone de módulos que se encargan de tareas específicas o propias de una o varias áreas de una empresa, al ser un sistema modular una compañía puede elegir que módulos desea y cuáles no, sin embargo hay unos módulos base de los cuales dependen otros más específicos y se dice que es configurable porque los módulos son adaptables a las necesidades de cada empresa sin importar a que se dedique esta.

Los sistemas de planeación de recursos han sido descritos como uno de los sistemas más prometedores pero a la vez más retadores sistemas de información desarrollados, distintas organizaciones no han escatimado tiempo en comprar e implementar sistemas ERP [12].

Un sistema ERP es una aplicación informática que permite gestionar todos los procesos de negocio de una compañía en forma integrada. Sus siglas provienen del término en inglés ENTERPRISE RESOURCE PLANNING. Por lo general este tipo de sistemas está compuesto de módulos como Recursos Humanos, Ventas, Contabilidad y Finanzas, Compras, Producción entre otros, brindado información cruzada e integrada de todos los procesos del negocio. Este software debe ser parametrizado y adaptado para responder a las necesidades específicas de cada organización. Una vez implementado un ERP permite a los empleados de una empresa administrar los recursos de todas las áreas, simular distintos escenarios y obtener información consolidada en tiempo real [13].

ARQUITECTURA DE UN SISTEMA ERP

Desde que aparecieron los sistemas ERP a finales de los ochenta la arquitectura bajo la que se han desarrollado ha ido variando desde cliente servidor a tres capas y finalmente tras la aparición de la web se mudó a un sistema multicapas.

La Figura 1 muestra una arquitectura genérica descrita por David Sprott (2000), en ella se muestra todos los componentes que forman un sistema ERP. Los sistemas de ventajas competitivas son la automatización de los procesos de negocio que están soportados en dos infraestructuras: la infraestructura operacional de negocios y la infraestructura técnica [14].

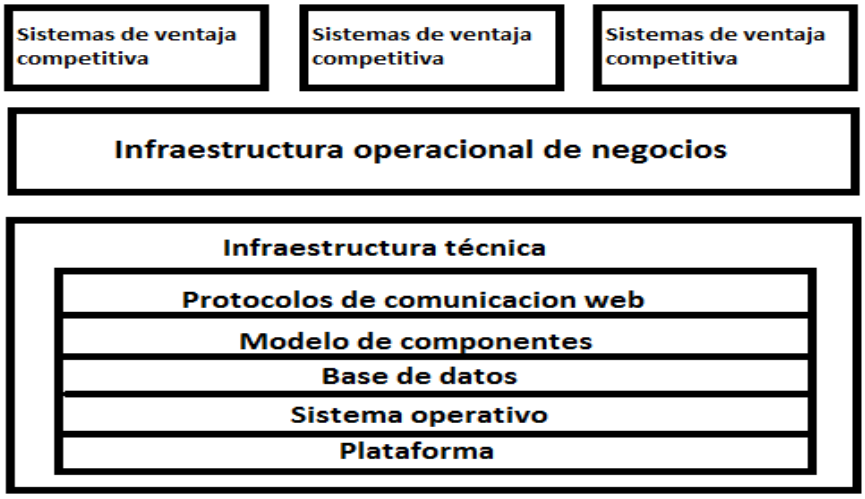


Ilustración 1 arquitectura genérica de un sistema ERP. Fuente: sistemas de planificación de recursos empresariales: un caso real.

ODOO

Odoo es un sistema ERP que actualmente es muy popular en el mercado y cuya licencia es libre, a continuación se encuentra una descripción de Odoo y de sus características más relevantes, se tomó como guía la información publicada en el libro *"The OpenERP Solution Odoo"*[15]que puede ser encontrado en la pagina web del sistema Odoo.

Odoo llamado anteriormente OpenERP y originalmente llamado TinyERP es un software administrativo que actualmente está siendo descargando más que cualquiera otro en el mundo, con más de 600 descargas diarias. Hoy está disponible en más de 18 idiomas y tiene una red global de partners y contribuyentes. Más de 800 desarrolladores participan en el proyecto con el sistema de desarrollo colaborativo.

Odoo es el único sistema de gestión en el cual es utilizado de forma rutinaria no solo por grandes empresas pero también por empresas muy pequeñas e independientes. Esta diversidad es una imagen de la flexibilidad del software: una coordinación elegante entre las expectativas de las personas del software y su facilidad de uso.

Y esta diversidad también se encuentra en varios sectores y oficios a los que da soporte el software, incluyendo productos agrícolas, textiles, subastas públicas, IT, y las asociaciones profesionales.

Por último, Odoo ha surgido de la combinación de código de alta calidad, una arquitectura bien pensada y el uso de tecnologías libres.

Odoo funciona de manera modular y viene provisto con algunos módulos y/o aplicaciones estándar que tomando como base a [16] se describirán a continuación:

- Accounting in human resources: ofrece un conjunto completo de herramientas que permite a las empresas gestionar los activos más importantes, su mano de obra.

  También proporciona las herramientas para controlar las licencias, asistencias y registros de horas.

- CRM: permite monitorear las actividades de ventas desde el primer contacto con cada cliente además de un seguimiento con todos los correos y documentos intercambiados con el mismo.

- Purchase: ofrece una completa aplicación de gestión de compras que permite hacer seguimiento a cada orden de compra que se genera, incluyendo la recepción de productos y la información de los proveedores.

- Point of Sale and inventory: este módulo permite la gestión multi-almacén y se basa en la estructura jerárquica de lugares, contempla tanto almacenes como bodegas de almacenamiento. Además tiene la capacidad de gestionar lotes y números de serie. Finalmente el exclusivo sistema de doble entrada de inventario, permite administrar sus propias ubicaciones tanto interna (dentro de una bodega), como externa (en almacenes, con clientes o en puntos de fábrica).

- Project: este módulo permite gestionar proyectos de cualquier naturaleza que pueden estar relacionados con los servicios de la compañía o ser de apoyo. Tiene la capacidad de dividir actividades en tareas, asignar recursos y tiempos, enviar correos a los socios acerca del avance del proyecto entre otras características.

2 Selección del meta-Metamodelo

En este capítulo se contemplará todo el proceso llevado a cabo para la selección del Meta-metamodelo, esto incluye un análisis exhaustivo a cada uno de los meta-metamodelos considerados es decir ECORE y MOF y los criterios que se tuvieron en cuenta para la selección.

EMF

EMF (eclipse modeling framework) es un framework de modelamiento y de generación de código para herramientas y aplicaciones basadas en un modelo de datos estructurado. Sobre la base de una especificación de modelo expresada en XMI, EMF provee herramientas y soporte en tiempo de ejecución para producir un conjunto de clases java para el modelo, un conjunto de clases adaptadoras que habilitan la vista y la edición basada en comandos del modelo y un editor básico. Los modelos pueden ser especificados usando comentarios java, documentos XML, o herramientas de modelamiento como Rational Rose y posteriormente pueden ser importados como un EMF [17].

Una característica importante de EMF es que provee la base para la interoperabilidad con otras herramientas y aplicaciones basadas en el mismo. Según la Eclipse ORG referenciada en [18] EMF se compone de 3 partes fundamentales:

- EMF:

  El core EMF incluye un metamodelo (ECORE) para describir modelos y soporte en tiempo de ejecución para estos incluyendo notificaciones, soporte en persistencia con serialización XMI por defecto, y una muy eficiente y reflexiva API para manipular objetos de manera genérica.

- EMF Edit:

  El framework EMF edit incluye clases genéricas y reusables para la construcción de editores para modelos EMF.

- EMF codagen:

  El generador de código EMF codagen es capaz de generar todo lo necesario para construir por completo un editor para un modelo EMF. Este incluye una interfaz gráfica de usuario en la cual las opciones de generación pueden ser especificadas y los generadores pueden ser invocados.

La generación de código puede soportar tres niveles:

- Modelo

  Provee interfaces JAVA y clases de implementación para cada clase en el modelo. Además otra clase para la implementación de fábrica y paquete (meta datos).

- Adaptadores

  Generan clases de implementación llamadas Item Providers que adaptan las clases del modelo para ser editadas y mostradas.

- Editor

  Produce un editor debidamente estructurado que se ajusta al estilo de editores recomendado por Eclipse y EMF y sirve como punto de inicio para la personalización.

Componentes de ecore

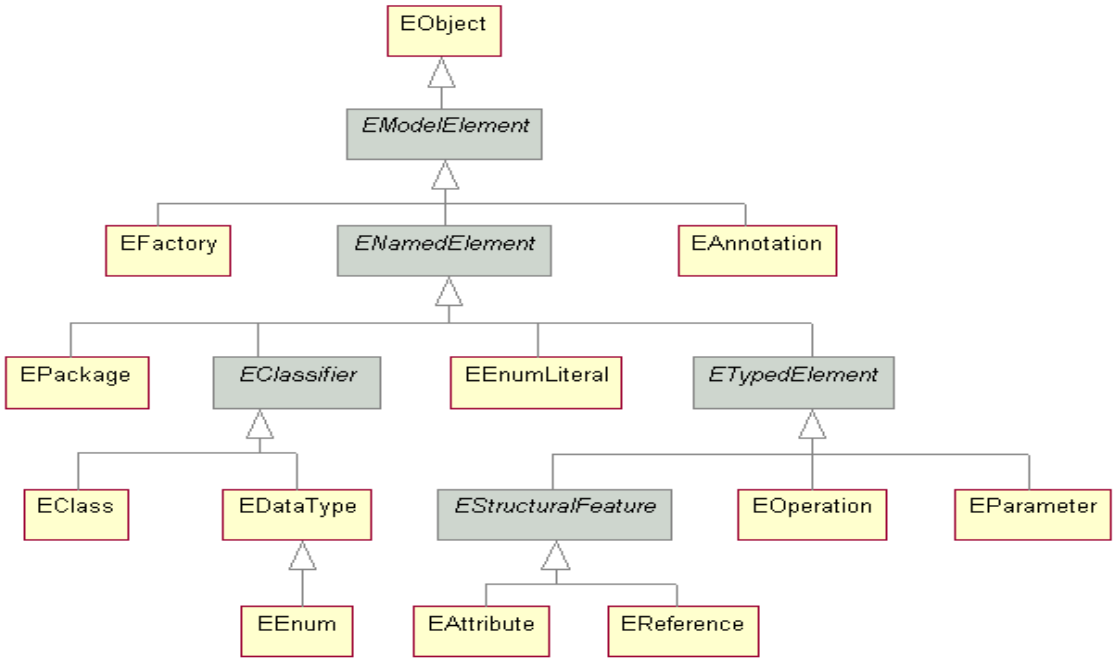


Ilustración 2 Jerarquía de los componentes de ecore. Fuente: Eclipse ORG, Package org.eclipse.emf.ecore

Tabla 2

Descripción componentes de ECORE Fuente: elaboración propia Realizado con información sacada de: Eclipse ORG, "Package org.eclipse.emf.ecore"

EObject	Un representación del modelo de objeto Object
EModelElement	Un representación del modelo de objeto ModelElement
EFactory	patrón Factory para el modelo
ENamedElement	Un representación del modelo de objeto NamedElement
EAnnotation	Un representación de del modelo de objeto Annotation
EPackage	El paquete para el modelo
EClassifier	Un representación del modelo de objeto Classifier
EENumLiteral	Un representación del modelo de objeto ENumLiteral
ETypedElement	Un representación del modelo de objeto TypedElement
EClass	Un representación del modelo de objeto Class
EDataType	Un representación del modelo de objeto DataType
EStructuralFeature	Un representación del modelo de objeto StructuralFeature
EOperation	Un representación del modelo de objeto Operation
EParameter	Un representación del modelo de objeto Parameter
EENum	Un representación del modelo de objeto ENum
EAttribute	Un representación del modelo de objeto Attribute
EReference	Un representación del modelo de objeto Reference

Interface Eobject, de ella se desprenden todos los objetos que tienen lugar en EMF.

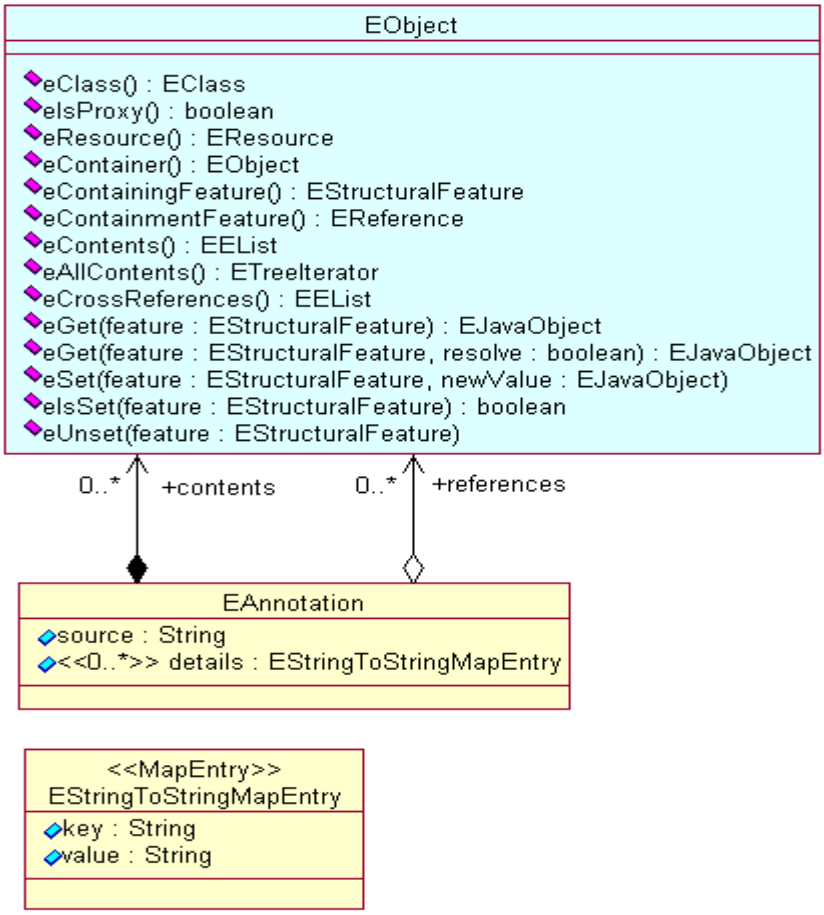


Ilustración 3 Interface EObject de EMF. Fuente: Eclipse ORG, "Package org.eclipse.emf.ecore.

MOF

MOF (meta Object Facility) es un estándar del OMG para la ingeniería dirigida por modelos, su propósito es proveer un sistema de tipo para entidades en la arquitectura CORBA y una serie de interfaces a través de las cuales los tipos pueden ser creados y manipulados

MOF puede ser usado para definir e integrar una familia de metamodelos usando conceptos básicos de modelado de clases, de acuerdo con [19].algunos beneficios de MOF son:

- Las reglas para modelar metadata son más simples (basta con entender un subconjunto de clases UML modeladas sin notaciones adicionales o modelos de construcción)
- Varias asignaciones de tecnología de MOF como XMI, JMI entre otras se pueden aplicar a un amplio rango de modelos UML incluyendo perfiles UML.
- Un soporte más amplio para el metamodelamiento (cualquier herramienta de modelado de UML puede ser usada para modelar meta data de manera más sencilla).

MOF se divide en dos grandes componentes CMOF y EMOF, los cuales serán explicados a continuación de acuerdo a *OMG Meta Object Facility(MOF) core specification* referenciado en [19].

EMOF

Es un subconjunto de MOF que corresponde cercanamente a las facilidades encontradas en OOPLs y XML. Lo más importante de EMOF es que este provee un sencillo framework para mapear desde modelos MOF hasta implementaciones tales como JMI y XMI para metamodelos simples. El primer objetivo de EMOF es permitir que metamodelos simples sean definidos usando conceptos igualmente simples.

El modelo EMOF usa clases de modelos UML restringidas e incluye capacidades de lenguaje adicionales definidas en un estándar internacional. También Mezcla la reflexión, los identificadores y los paquetes de extensión de capacidades que proveen servicios de descubrimiento, manipulación, identificación y extensión de la metadata.

EMOF como todos los metamodelos en la familia MOF es descrito como un modelo UML, sin embargo un soporte completo en EMOF requiere ser especificado por sí mismo removiendo cada paquete que se haya mezclado y cada redefinición que haya podido ser especificada en el modelo UML.

La relación entre EMOF e InfrastructureLibrary:Core:Basic requiere una compleja explicación. EMOF mezcla su base con las capacidades de MOF y algunas de sus extensiones. De manera ideal EMOF extendería su base usando subclases que proveen propiedades y operaciones adicionales. Entonces MOF podría ser formalmente especificado como EMOF sin requerir un paquete de mezcla. Sin embargo esto no es suficiente ya que la reflexión tiene que incluir objetos en la jerarquía de clases como una nueva superclase de Basic::Element que requiere una mezcla. Como resultado de esto, EMOF es un modelo separado que contiene un Basic pero no lo hereda.

Utilizando PackageMerge, EMOF es directamente compatible con archivos Basic XMI. Además el uso de paquetes mezclados también asegura que estará actualizado con cualquier cambio que se haga en Basic. La razón para especificar completamente el EMOF mezclado es proveer un metamodelo que pueda ser usado por las herramientas de metamodelo de bootstrap arraigadas en EMOF sin necesidad de una implementación de CMOF o de paquetes de mezclas semánticas .

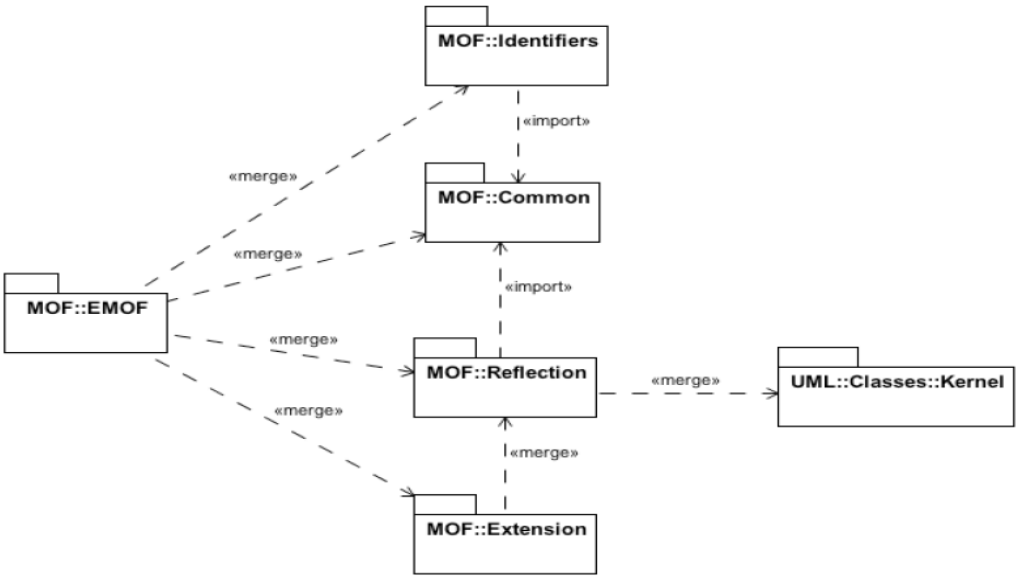


Ilustración 4Modelo EMOF. Fuente: OMG Meta Object Facility(MOF) core specification

Tipos de datos EMOF

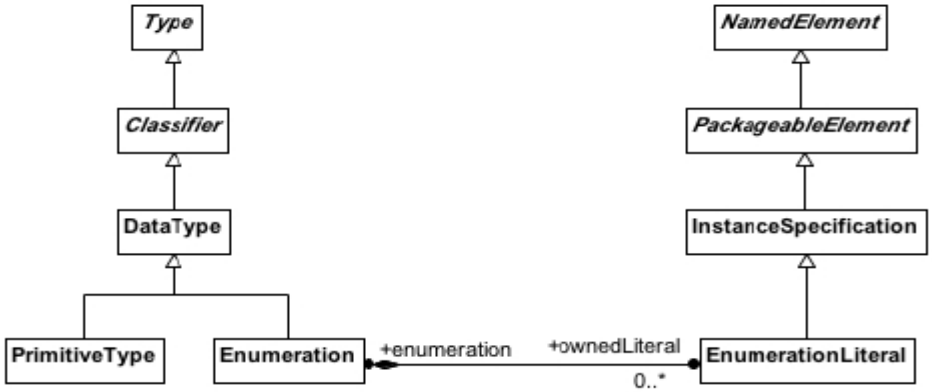


Ilustración 5 Tipos de datos. EMOF Fuente: OMG Meta Object Facility (MOF) core specification

Paquete EMOF

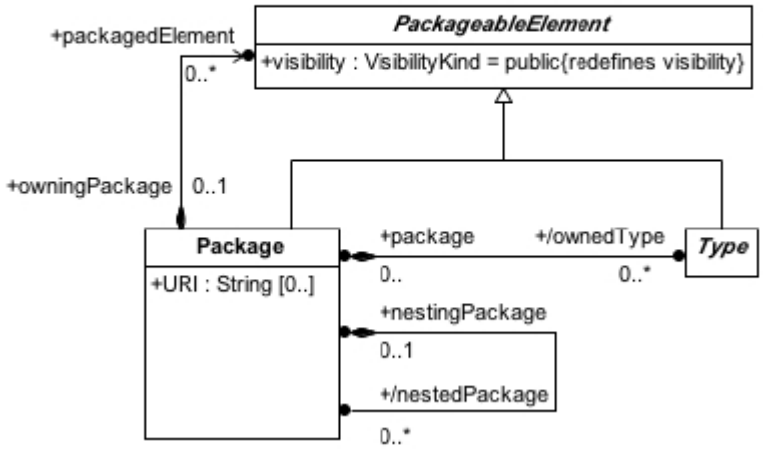


Ilustración 6 Paquete EMOF. Fuente: OMG Meta Object Facility (MOF) core specification

Restricciones EMOF

A continuación se presentaran las restricciones más importantes para el uso de EMOF:

[1] El tipo de operación ::raisedException está limitada a ser una clase y no un tipo.

[2] Notacionalmente la opción de suprimir las flechas de navegación esta deshabilitada, Así como las asociaciones bidireccionales son indistinguibles de las asociaciones no navegables.

[3] Los nombres son requeridos para todos los NamedElements(elementos nombrados) excepto para los ValueSpecifications(especificaciones de valor).

[4] Core::Basic y EMOF no soportan visibilidades. Todas las visibilidades propias deben ser explícitamente públicas donde sean aplicables, no es para todos los NamedElements(elementos nombrados), ElementImports(elementos importados) y PackageImports(paquetes importados). Es más no son permitidos los alias para ningún ElementImport(elemento importado).

[5] La propiedad idID solo puede ser verdadera en una propiedad por clase.

[6] Un meta-modelo EMOF está restringido a usar las siguientes meta-clases concretas de UML(descripción tomada de *"Unified Modeling language specification"*[20]):

- Asociación: Especifica los elementos relacionados, es una unión derivada
- Clase:
- Comentario: anotación textual que puede ser adjuntado a un conjunto de elementos.
- Tipo de dato: clase cuyas instancias son identificadas solo por su valor.
- Enumeración: tipo de dato cuyos valores se enumeran en el modelo como literales de enumeración.
- Enumeración literal: valor de datos definidos por el usuario para una enumeración.
- Generalización: Una generalización es una relación taxonómica entre un clasificador general y uno específico.
- Valor de instancia: especificación de valor que identifica una instancia
- Literal booleano: especificación de un valor booleano.

- Literal entero: especificación de un valor entero.
- Literal nulo: especifica la ausencia de un valor.
- Literal real: especificación de un valor real.
- Literal String: especificación de una cadena de texto.
- Literal ilimitado natural: especificación de un número natural ilimitado.
- Operación: característica del comportamiento de un clasificador que especifica nombre, tipo, parámetros y restricciones invocando un comportamiento asociado.
- Paquete: los paquetes proveen la principal capacidad de UML, la estructuración genérica.
- Parámetro: especificación de un argumento usado para transmitir información dentro o fuera de la invocación de una característica de comportamiento.
- Tipo primitivo: define un tipo de dato predefinido sin subestructuras.
- Propiedad: característica estructural.

[7] Las siguientes propiedades deben estar vacías: Association::navigableOwnedEnd, Class::nestedClassifier, Classifier::/general para instancias de datatype, Operation::bodyCondition, Operation::postcondition, Operation::precondition, Operation::redefinedOperation, Parameter::defaultValue, Property::qualifier, Property::redefinedProperty, Property::subsettedProperty.

[8] Las siguientes propiedades deben ser falsas: Association::isDerived, Classifier::isFinalSpecialization, Feature::isStatic, Property::isDerivedUnion, RedefinableElement::isLeaf.

[9] La generalizacion::isSubstitutable(es sustituible) debe ser verdadera.

[10] Una asociación tiene exactamente 2 miembros finales. Puede que nunca tenga un navigableOwnedEnd(propietario final navegable) es decir que siempre será propiedad de las clases y finalmente solo puede tener un ownedEnd(propietario final).

[11] Una operación puede tener hasta un parámetro cuya dirección es "return", es más una operación no puede tener ningún ParameterSet (conjunto de parámetros).

[12] Los comentarios solamente pueden anotar instancias de un NamedElement (elemento nombrado).

[13] Solo un atributo miembro de la clase puede tener la propiedad isID= verdadero.

[14] La propiedad::aggregation debe ser nula o compuesta.

[15] Las enumeraciones no deben tener atributos u operaciones necesariamente.

[16] BehavioralFeature(característica de comportamiento) debe ser secuencial.

[17] Una clase no debe estar activa.

[18] Un EnumerationLiteral(literal de enumeración) no debe tener un ValueSpecification (especificación de valor).

[19] Un parámetro de operación no debe tener efecto, excepciones o características de transmisión.

[20] Un TypedElement(elemento tipado) no puede ser clasificado por asociación.

[21] Un TypedElement(elemento tipado), un LiteralSpecification(literal de especificación) o un OpaqueExpression (expresión opaca) deben tener un tipo.

[22] Un TypedElement(elemento tipado) que es una especie de parámetro o propiedad clasificado por una clase no debe tener un valor por defecto.

[23] Para un TypedElement(elemento tipado) que es un tipo de parámetro o de propiedad clasificada por una enumeración, el valor por defecto, si lo hay debe ser de tipo instanceValue(valor de instancia).

[24] Para un TypedElement(elemento tipado) que es un tipo de parámetro o de propiedad clasificada por un PrimitiveType(tipo primitivo), el valor por defecto si lo hay debe ser de tipo LiteralSpecification(literal de especificación).

[25] Una propiedad de conjuntos compuesta con multiplicidad obligatoria no puede ser subconjunto de otra propiedad de conjuntos compuesta con multiplicidad obligatoria.

[26] Una propiedad clasificada por un tipo de DataType(tipo de dato) debe tener la propiedad agragation= none (agregación=ninguna).

[27] Una propiedad perteneciente a un DataType(tipo de dato) solamente puede ser clasificada por un DataType(tipo de dato).

[28] Cada asociación la propiedad MemberEnd(miembro final) debe ser clasificada por la clase.

[29] Una propiedad o parámetro multivalorado no puede tener un valor por defecto.

[30] Los valores mínimos y máximos de MultiplicityElement(elemento con multiplicidad) deben ser del tipo LiteralInteger(literal entero) y LiteralUnlimitedNatural(literal natural ilimitado) respectivamente.

CMOF

CMOF model es un metamodelo usado para especificar otros meta-modelos como UML". Esta construido a partir de EMOF y de Core::Constructs de UML2, el paquete modelo no define ninguna clase por sí mismo. En cambio mezcla paquetes con extensiones

que juntas definen capacidades básicas de meta-modelado.

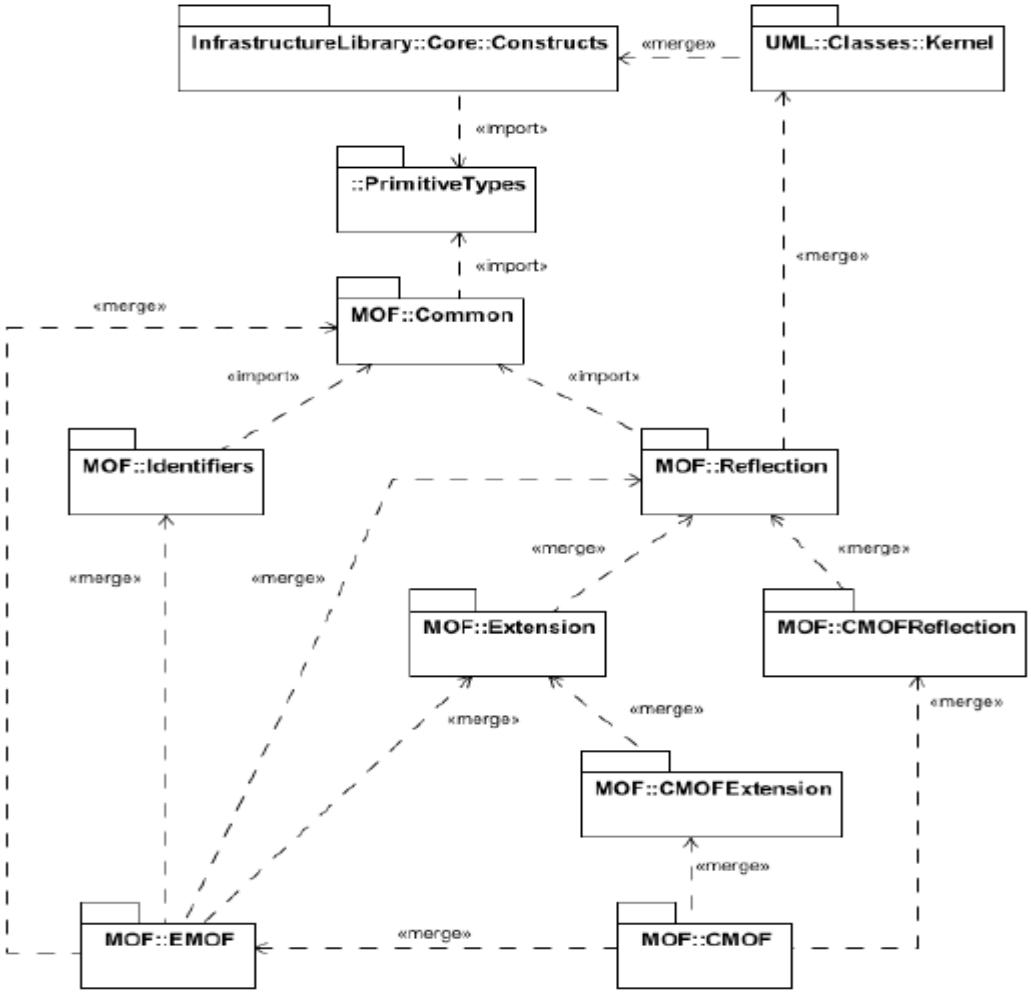


Ilustración 7 paquetes CMOF. Fuente: OMG Meta Object Facility(MOF) core specification.

El modelo CMOF mezcla el paquete EMOF de MOF el cual incluye algunos paquetes de capacidades de MOF como Identificadores, Reflexión y Extensión.

Restricciones de CMOF

Acá se detallan las restricciones del paquete CMOF que son aplicadas a metamodelos procesados por una implementación de CMOF.

[1] La multiplicidad de asociación es limitada a 2, no existen asociaciones n-arias, a diferencia de EMOF, las asociaciones de CMOF pueden tener propietarios de asociación finales navegables.

[2] El tipo de operación ::raisedException está limitado a ser una clase más que un tipo.

[3] Para soportar implementaciones limitadas de la clase Integer(entero), cada instancia de esta debe ocurrir como un valor de atributo de un elemento dentro del rango de enteros que pueden ser representados.

[4] Para soportar implementaciones limitadas de la clase String cada instancia de esta echa como un valor de atributo de un elemento tiene un tamaño que no excede los 65535 caracteres.

[5] Los nombres son requeridos para todos los NamedElements(Elementos nombrados) excepto para los ValueSpecifications(especificaciones de valor).

[6] CMOF no soporta visibilidades. Todas las visibilidades propias deben ser explícitamente públicas donde sean aplicables, no es para todos los NamedElements(elementos nombrados), ElementImports(elementos importados) y PackageImports(paquetes importados). Es más no son permitidos los alias para ningún ElementImport(elemento importado).

[7] Enumeraciones pueden no tener atributos u operaciones.

[8] La propiedad idID solo puede ser verdadera en una propiedad por clase.

[9] Un meta-modelo CMOF está restringido a usar las siguientes meta-clases del Kernel de UML(descripción tomada de *"Unified Modeling language specification"*[20]):

- Asociación: Especifica los elementos relacionados, es una unión derivada.
- Clase:

- Comentario: anotación textual que puede ser adjuntado a un conjunto de elementos.
- Tipo de dato: clase cuyas instancias son identificadas solo por su valor.
- Enumeración: tipo de dato cuyos valores se enumeran en el modelo como literales de enumeración.
- Enumeración literal: valor de datos definidos por el usuario para una enumeración.
- Generalización: Una generalización es una relación taxonómica entre un clasificador general y uno específico.
- Valor de instancia: especificación de valor que identifica una instancia
- Literal booleano: especificación de un valor booleano.
- Literal entero: especificación de un valor entero.
- Literal nulo: especifica la ausencia de un valor.
- Literal real: especificación de un valor real.
- Literal String: especificación de una cadena de texto.
- Literal ilimitado natural: especificación de un número natural ilimitado.
- opaqueExpresion: especificación de valor que especifica el cálculo de una colección de valores, ya sea en términos de un comportamiento UML o con base a una declaración textual en un lenguaje distinto.
- Operación: característica del comportamiento de un clasificador que especifica nombre, tipo, parámetros y restricciones invocando un comportamiento asociado.
- Paquete: los paquetes proveen la principal capacidad de UML, la estructuración genérica.
- Propiedad: característica estructural.
- Importación de paquetes: relación que importa todos los miembros no privados de un paquete dentro del namespace al que pertenece el paquete.

- Mezcla de paquetes: define como los contenidos de un paquete pueden ser extendidos por los comntenidos de otro paquete.
- Parámetro: especificación de un argumento usado para transmitir información dentro o fuera de la invocación de una característica de comportamiento.
- Tipo primitivo: define un tipo de dato predefinido sin subestructuras.

[10] Las siguientes propiedades deben estar vacías: Class::nestedClassifier., Property::qualifier

[11] El valor de Feature::isStatic(característica: es estatico) debe ser falso.

[12] La generalización::isSubstitutable (es sustituible)debe sr verdadero.

[13] una propiedad o parámetro multivalorado no puede tener un valor por defecto. El valor por defecto de una propiedad o parámetro tipado como de tipo primitivo debe ser de la clase LiteralSpecification, El valor por defecto de una propiedad o parámetro tipado como una enumeración debe ser de la clase InstanceValue, una propiedad o parámetro tipeados por una clase no pueden tener valores por defecto.

[14] Los valores mínimo y máximo de un elemento de multiplicidad deben ser del tipo LiteralInteger y LiteralUnlimitedNatural respectivamente.

[15] La generalización::isSubstitutable(es sustituible) debe ser verdadera.

[16] Solamente un atributo miembro de una clase puede tener el atributo isId como verdadero, los demás(los que hayan sido heredados) deben ser redefinidos para que no estén disponibles o se debe cambiar un atributo isId a falso.

[17] La propiedad agregación debe ser nula o compuesta.

[18] BehavioralFeature(característica de comportamiento) debe ser secuencial.

[19] Clase no debe estar activa.

[20] Un EnumerationLiteral no debe tener un EnumerationLiteral.

[21] Un parámetro de operación no debe tener efecto, excepción o características de transmisión.

[22] Un TypedElement no puede ser tipado por asociación.

[23] Un TypedElement diferente a un LiteralSpecification o un OpaqueExpression debe tener un tipo.

[24] Un TypedElement que es un tipo de parámetro o propiedad tipada por una clase no debe tener un valor por defecto.

[25] Para un TypedElement que es un tipo de parámetro o propiedad tipada por Enumeración si acaso tiene un valor por defecto este debe ser del tipo InstanceValue.

Elección del Meta-metamodelo

Sopesando las dos opciones de meta-metamodelo las que se acaban de presentar y tras un análisis en el que se tuvieron en cuenta varios factores se llegó a la conclusión de que el meta-metamodelo que más se ajusta a las necesidades del proyecto es el eclipse modeling framework mejor conocido como ECORE.

Factores decisivos para la elección del meta-metamodelo

- Compatibilidad con editores gráficos:

  Uno de los puntos que se tuvieron en cuenta para la elección de ECORE como el meta-metamodelo del cual se iba a partir fue la compatibilidad con los editores

gráficos ya que la idea original siempre fue el desarrollo de un DSL gráfico, de las opciones evaluadas la mejor era ECORE ya que es compatible con GMF y con sirius que eran los editores gráficos contemplados desde el inicio.

- Complejidad del Metamodelo

  Como se pudo observar en el numeral anterior MOF se divide en EMOF y en CMOF siendo EMOF ideal para el modelamiento de metamodelos simples mientras que CMOF es usado para metamodelos de una mucha mayor complejidad, sin embargo al inicio de este proyecto no había una certeza de la complejidad que tendría el metamodelo a diseñar por lo cual era más viable hacer uso de ECORE ya que este no depende directamente de la complejidad del metamodelo.

- Documentación

  Para entender cómo funcionan cada uno de los meta-metamodelos estudiados fue necesario hacer uso de su documentación y de algunos ejemplos que presentan respectivamente el OMG y eclipse, siendo tanto los ejemplos como la documentación de ECORE más comprensible que la de MOF.

- Origen de Ecore

  Algunos autores afirman que Ecore está basado en EMOF [21], razón por la cual se consideró a la hora de la elección que las características de EMOF que fueran productivas para el proyecto se encontrarían en ECORE y a su vez este aportaría otras nuevas.

3 Diseño del metamodelo

En el capítulo anterior se definió que el meta-metamodelo en el cual se llevaría a cabo el diseño del metamodelo sería ECORE, por tanto en este capítulo contemplaremos todo el proceso que se llevó a cabo para la definición del mismo utilizando el meta-metamodelo ECORE.

A su vez este capítulo se divide en tres fases que separan de manera clara el proceso ejecutado para el diseño del modelo estas son: la obtención de requerimientos, la abstracción y la implementación del metamodelo.

Obtención de requerimientos

Esto hace referencia a la descripción de los componentes a modelar, dicha descripción es tomada directamente de la documentación de ODOO [22] según la cual el modulo se compone de los siguientes componentes:

- Objetos de negocios:

  Declarados como clases de Python, estos objetos son persistidos automáticamente por odoo basados en su configuración.

- Archivos de datos:

Archivos de tipo XML o CSV, los cuales declaran meta-datos, tales como vistas o flujos de trabajo.

- Controladores Web:

  Manejan las peticiones web.

- Archivos web estáticos:

  Imágenes, archivos Css o javascript utilizados por la interfaz web.

Módulos:

Como se puede evidenciar en la estructura de los módulos Odoo, un módulo del sistema consiste en un conjunto de archivos de distintas características los cuales bridan una funcionalidad, basados en la descripción se puede observar que el archivo /__openerp__.py contiene la definición de la funcionalidad de los mismos[22], dicha funcionalidad tiene la siguiente estructura:

- Nombre:

  Contiene el nombre legible del módulo que se está creando.

- Versión:

  Contiene la versión actual del módulo que se está creando.

- Descripción:

  Contiene una descripción extendida del módulo.

- Autor:

  Contiene el nombre del autor del módulo que se está creando

- Sitio Web:

  Contiene una dirección a un sitio web del módulo que se está creando.

- Licencia:

  Contiene el tipo de licencia de software a aplicar sobre el módulo.

- Categoría:

  Categoría del módulo que se está creando.

- Modelos:

  Modelos que componen el módulo

- Vistas:

  Archivos XML que definen la información a mostrar a los usuarios finales.

Modelos

Un módulo en Odoo está compuesto por uno o más modelos, estos modelos son encapsulamientos de distintas propiedades y funcionalidades y pueden interactuar con otros del mismo módulo o de uno diferente, a continuación se muestran las características propias de los modelos.

- Nombre: Nombre del modelo.

- Atributos: Atributos del modelo, son los campos a persistir por cada modelo.
- Relaciones: Relaciones que mantiene el modelo con otros modelos.

Relaciones

Las relaciones en el sistema Odoo, como se contempla en su documentación [22] se dan directamente en los atributos de cada modelo y pueden ser de la siguiente forma:

- muchos a uno
- uno a muchos
- muchos a muchos

Vistas:

Los archivos de vistas del sistema Odoo son los encargados de mostrar la información que resulta útil para los usuarios finales, estas vistas se configuran en formato XML y son leídas por Odoo en el momento de desplegar información y se pueden catalogar de la siguiente forma:

- Vistas de árbol o listas
- Vistas de formulario de creación y/o edición

Abstracción

Se refiere a la abstracción hecha a partir de la descripción de los componentes a modelar es decir la fase anterior a esta. Esta abstracción contiene los tipos de componentes que se deben modelar para lograr la construcción de módulos en Odoo.

Odoo, siendo un sistema modular y extensible permite un gran número de configuraciones en sus módulos, como es previsible, es necesario especificar qué características pueden ser modeladas dentro del meta modelado, a continuación se encuentra una descripción de los distintos elementos que son tenidos en cuenta en el proceso de meta modelado de la herramienta.

Metadatos

Los metadatos de un módulo de odoo que serán tenidos en cuenta para la fase de meta-metamodelado son los siguientes:

Modulo:

- Nombre: Contiene el nombre legible del módulo que se está creando.
- Versión: Contiene la versión actual del módulo que se está creando.
- Descripción: Contiene una descripción extendida del módulo.
- Autor: Contiene el nombre del autor del módulo que se está creando
- Sitio Web: Contiene una dirección a un sitio web del módulo que se está creando.
- Licencia: Contiene el tipo de licencia de software a aplicar sobre el módulo.
- Categoría: Categoría del módulo que se está creando.
- Modelos: Modelos que componen el módulo
- Modelos que contiene un módulo

Modelo:

- Nombre modelo: Nombre del modelo.
- Atributos: Atributos del modelo.

Definición de Atributos

Un modelo de Odoo está compuesto por uno o más atributos, estos atributos son persistidos por Odoo y brindan a los modelos la capacidad de almacenar distintos tipos y cantidades de datos, de manera que, en los atributos es donde reside la información del negocio que va a ser útil para los usuarios finales.

Características de atributo:

A continuación se encuentra una lista de las características a tener en cuenta en el meta modelo

- Nombre: Define el nombre del atributo.
- Tipo de atributo: Define el tipo de atributo a usar.
- Requerido: Define si el atributo es requerido o es opcional.
- Ayuda: Texto de ayuda que indica el propósito del atributo al usuario final.

Tipos de atributo:

Odoo por defecto tiene definidos una serie de tipos de atributos que pueden utilizarse en todos los modelos, es necesario tener en cuenta estos tipos a la hora de la creación del meta modelo, a continuación se encuentra la lista de tipos de atributos a modelar.

- Boolean: Almacena un campo de tipo booleano.
- Char: Almacena un campo de tipo char de longitud variable.
- Text: Almacena un campo de texto largo.
- HTML: Almacena texto en formato HTML (lenguaje de marcas de hipertexto).
- Integer: Almacena un campo de tipo entero.

- Float: Almacena un campo de tipo flotante, no soporta valores nulos (NULL).
- Date: Almacena un campo de fecha.
- DateTime: Almacena un campo de fecha y hora.
- Binary: Almacena en un campo, un archivo codificado con el algoritmo base64.
- Selection: Almacena un campo tipo texto, muestra un campo de tipo selección al usuario final.

Relaciones

Como se mencionó en la fase anterior los modelos pueden interactuar entre sí, ya sea que pertenezcan al mismo módulo o no, la forma de comunicación de estos es a través de las relaciones, y estas a su vez están directamente conectadas a un atributo de cada modelo, a continuación se encuentran las características a modelar para las relaciones:

- Atributo del modelo inicial: Muestra el atributo del cual parte la relación.
- Atributo del modelo final: Muestra el atributo en el cual termina la relación.
- Tipo de relación: Muestra el tipo de relación entre los modelos.

Tipos de relación:

Odoo contempla tres tipos de relación que se pueden dar entre los modelos, estas relaciones se ven reflejadas directamente en el componente relacional de Odoo

- Uno a Muchos
- Muchos a Muchos
- Se omite la relación “Muchos a Uno” ya que esta es una propiedad de inversión obligatoria en las relaciones “Uno a Muchos”.

Implementación del meta modelo de módulos del sistema Odoo:

Como su nombre lo indica en esta fase se va a ver paso a paso como fue el proceso para la implementación del metamodelo en Ecore de acuerdo a la abstracción realizada en la fase anterior.

Paso a paso proceso de implementación:

Agregar archivo Ecore al proyecto

El primer paso dentro de la creación del metamodelo es la creación del archivo Ecore y sus configuraciones básicas, para ello primero debemos crear un proyecto y dentro de este adicionamos un nuevo archivo tipo Ecore, el cual va a soportar el meta-modelado.

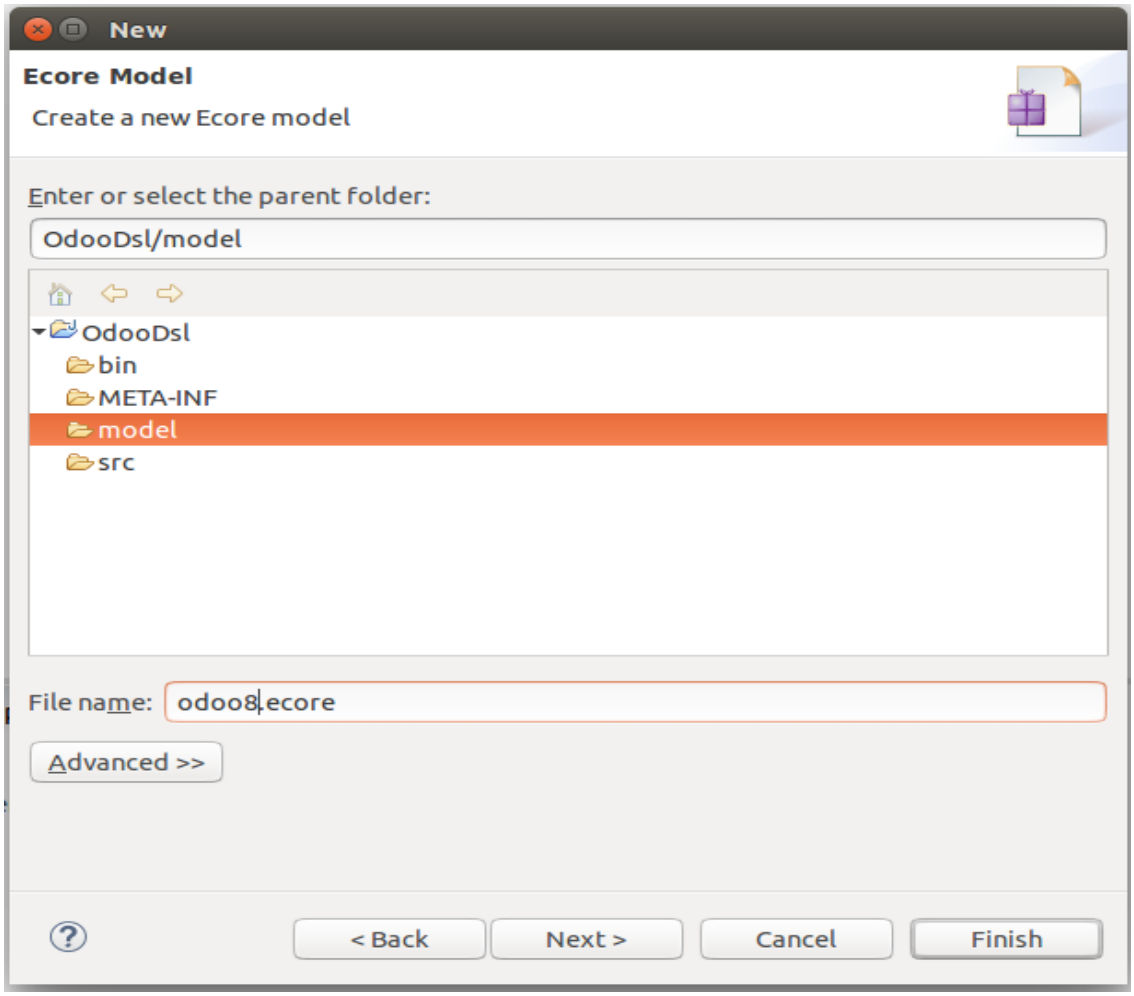


Ilustración 8 Creación del Archivo ECORE. Fuente: Elaboración Propia

A continuación se debe comprobar si el archivo fue agregado correctamente al entorno

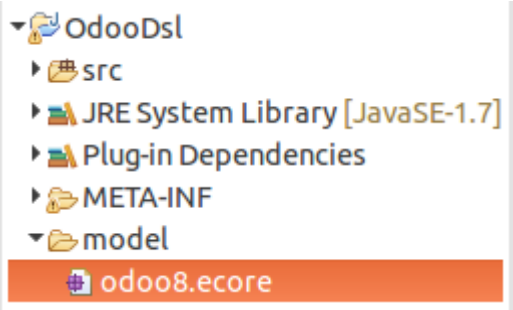


Ilustración 9 Archivo ECORE presente en el entorno. Fuente: Elaboración Propia

Configurar metadatos del paquete de Ecore

Ahora se deben configurar los metadatos de Ecore de la siguiente manera:

Name: odoo8

NS Prefix: odoo

Ns URI: http://odoo8modulesmetamodel

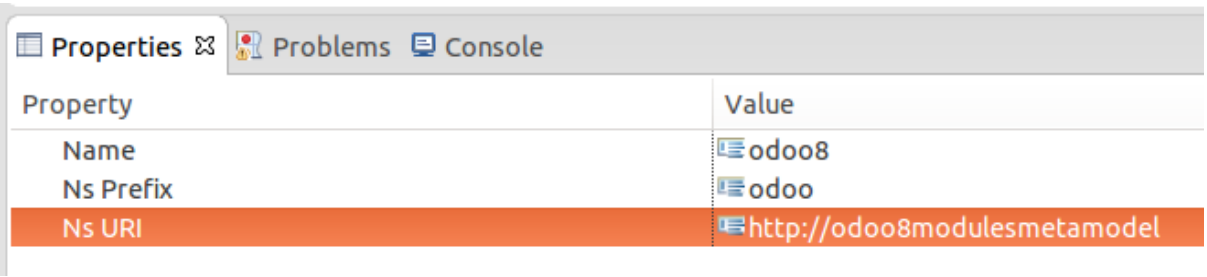


Ilustración 10 Configuración metadatos paquete Ecore. Fuente: Elaboración Propia

Proceso de Meta-modelado

Ecore debe tener una Eclass base, la cual contiene todos los componentes a modelar, en el caso de este modelo le nombre de la EClass será "RequerimientoOdoo", así que se debe agregar al paquete odoo8 una nueva Eclass llamada "RequerimientoOdoo".

Meta-modelado de módulo

Para este paso se debe agregar al Ecore una nueva EClass llamada "ModuloOdoo", en dicha clase deben estar los atributos propios del módulo con sus configuraciones, para este caso son los siguientes:

1. Nombre:
    a. Name: nombre
    b. Etype: EString
2. Versión:
    a. Name: version
    b. EType: EString
3. Descripción:
    a. Name: descripción
    b. EType: EString
4. Autor:
    a. Name: autor
    b. EType: EString
5. Sitio Web:
    a. Name: website
    b. EType: EString
6. Licencia:
    a. Name: licencia
    b. EType: EString
7. Categoría:
    a. Name: categoría

b. EType: EString

Después de agregar los atributos, el meta-modelo de la EClass “ModuloOdoo” se debe ver de la siguiente manera en Ecore:

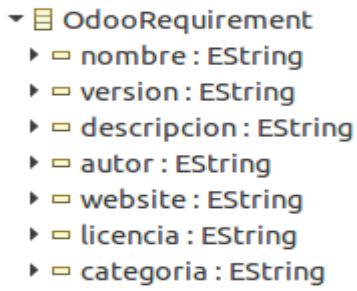


Ilustración 11 Metamodelado de la Eclass ModuloOdoo. Fuente: Elaboración Propia

Meta-modelado de modelo

Se agrega una nueva Eclass el Ecore llamada ModeloOdoo, la cual contiene los atributos pertenecientes a los modelos de Odoo, para este caso particular son los siguientes atributos:

1. Nombre:
    a. Name: nombre
    b. Etype: EString

Al finalizar el proceso de agregado de los campos, el meta-modelo de la EClass “ModeloOdoo” debe verse de la siguiente manera en Ecore:

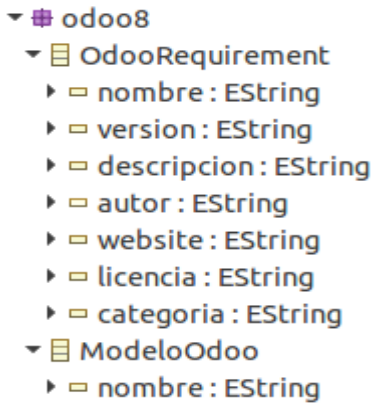


Ilustración 12 Metamodelado EClass ModeloOdoo. Fuente: Elaboración Propia

Meta-modelado de atributo

Se debe agregar una nueva Eclass el Ecore llamada “AtributoOdoo”, la cual contiene los atributos pertenecientes a los modelos de Odoo, para este caso particular son los siguientes atributos:

1. Nombre:
    a. Name: nombre
    b. EType: EString
2. Requerido:
    a. Name: requerido
    b. EType: EBoolean
3. Ayuda:
    a. Name: ayuda
    b. EType: EString

Al finalizar el proceso de agregación de los campos, el meta-modelo de la EClass “AtributoOdoo” se debe ver de la siguiente manera en Ecore.

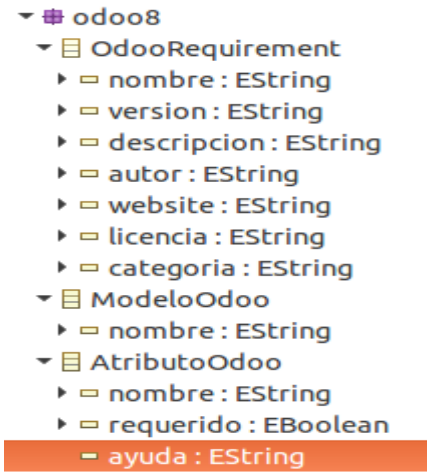


Ilustración 13 Metamodelado ECLass AtributoOdoo. Fuente: Elaboración Propia

Meta-modelado tipos de atributo

El siguiente paso es agregar un nuevo ENum (enumeracion) al Ecore llamado AtributoListaOdoo, este deberá contener los tipos de atributo que una entidad puede usar para el caso específico que se está tratando se agregara la siguiente lista:

1. Boolean
    a. Name: BOOLEAN
    b. Value: 0
2. Char
    a. Name: CHAR
    b. Value: 1
3. Text
    a. Name: TEXT
    b. Value: 2
4. Html
    a. Name: HTML
    b. Value: 3
5. Integer
    a. Name: HTML
    b. Value: 4
6. Float
    a. Name: FLOAT
    b. Value: 5
7. Date
    a. Name: DATE
    b. Value: 6

8. DateTime
    a. Name: DATETIME
    b. Value: 7
9. Binary
    a. Name: BINARY
    b. Value: 8
10. Selection
    a. Name: SELECTION
    b. Value: 9

Al finalizar el proceso de agregación de los campos, el meta-modelo del ENUM "AtributoListaOdoo" se debe visualizar de la siguiente manera en el Ecore.

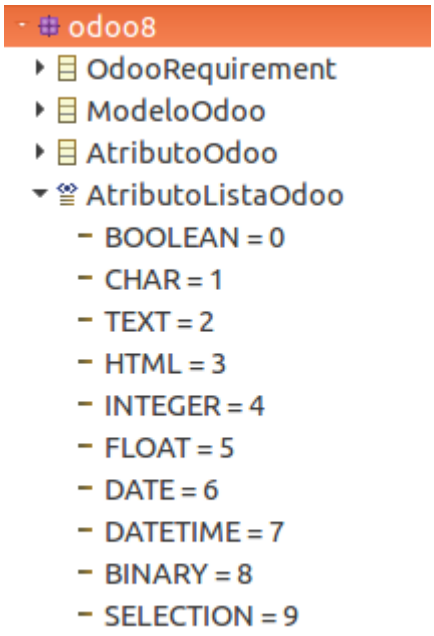


Ilustración 14 Metamodelado del Enum "AtributoListaOdoo". Fuente: Elaboración Propia

Meta-modelado relaciones

Se debe agregar una nueva Eclass el Ecore, para el caso específico se llamará "RelacionOdoo", y contendrá los atributos pertenecientes a las relaciones que son los siguientes:

1. Entidad Origen (EReference)
    a. Name: modeloOrigen
    b. EType: ModeloOdoo
    c. LowerBound: 0
    d. UpperBound: 1
2. Entidad Destino (EReference)
    a. Name: modeloDestino
    b. EType: ModeloOdoo
    c. LowerBound: 0
    d. UpperBound: 1

Al finalizar el proceso de agregación de los campos, el meta-modelo de la EClass "RelacionOdoo" se debe visualizar de la siguiente manera en el Ecore.

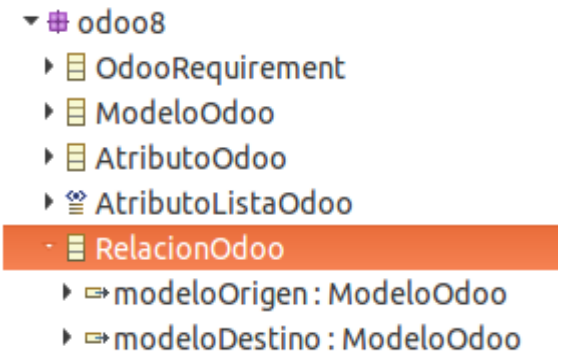


Ilustración 15 Metamodelado de la EClass RelacionOdoo. Fuente: Elaboración Propia

Meta-modelado tipos de relaciones

A continuación se agregara un nuevo ENum al Ecore llamado “TipoRelacionOdoo”, el cual contiene los tipos de relaciones que pueden ser utilizados en el modelo, después se agregará la siguiente lista de Literals al ENum “TipoRelacionOdoo”:

1. Uno a muchos
    a. Name: UNOAMUCHOS
    b. Value: 0
2. Muchos a Muchos
    a. Name: MUCHOSAMUCHOS
    b. Value: 1

Al finalizar el proceso de agregación de los campos, el meta-modelo del ENUM “TipoRelacionOdoo” se debe visualizar de la siguiente manera en el Ecore:

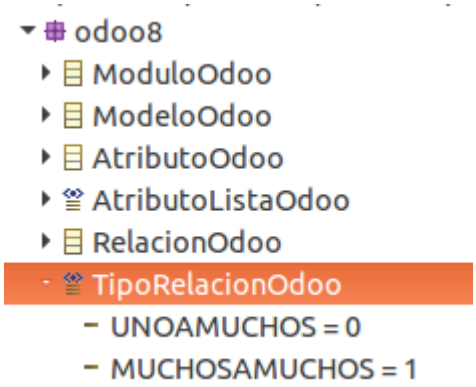


Ilustración 16 Metamodelado de ENum "tipoRelacionOdoo". Fuente: Elaboración Propia

Meta-modelado conexión de un módulo con sus modelos

Al Meta-modelo de ModuloOdoo es necesario agregarle una EReference que represente la relación de todos los modelos que componen el módulo, y se especifica de la siguiente forma:

1. Modulo tiene modelos
    a. Name: modulo_tiene_modelos
    b. EType: ModeloOdoo
    c. LowerBound: 0
    d. UpperBound: -1 (ilimitados)

Al finalizar, el Ecore de ModuloOdoo debe tener este aspecto:

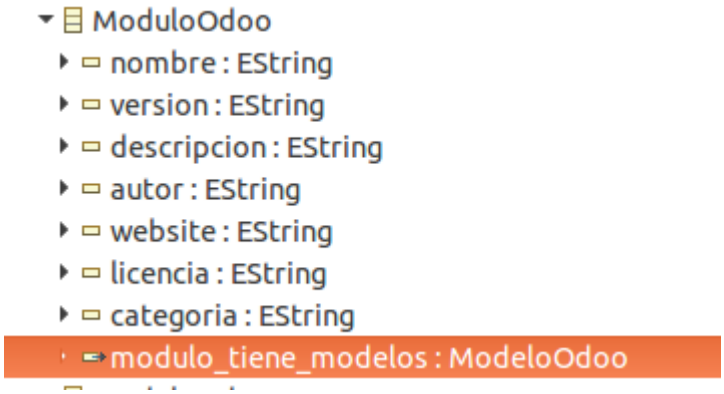


Ilustración 17 Metamodelado de la relación del módulo con sus modelos. Fuente: Elaboración Propia

Meta-modelado conexión de un modelo con sus atributos

Al Meta-modelo de ModeloOdoo es necesario agregarle una EReference que represente la relación de todos los atributos que componen el modelo

1. Modelo tiene atributos
    a. Name: modelo_tiene_atributos
    b. EType: ModeloOdoo
    c. LowerBound: 0
    d. UpperBound: -1 (ilimitados)

Al finalizar, el Ecore de AtributoOdoo debe tener este aspecto

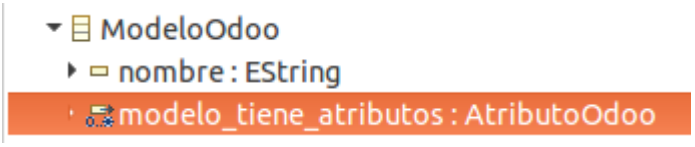


Ilustración 18 Metamodelado de la relación de un módulo con sus modelos. Fuente: Elaboración Propia

Meta-modelado relación AtributoOdoo a AtributoListaOdoo

Los tipos de campo que puede tomar un AtributoOdoo deben estar limitados a los tipos de atributo definidos en la enumeración AtributoListaOdoo, para definir esto se agrega un nuevo EAtribute al meta-modelo de AtributoOdoo.

1. Tipo de campo
    a. Name: tipoCampo
    b. EType: AtributoListaOdoo
    c. Default Value Literal: TEXT

Después de realizados los cambios, el Ecore de AtributoOdoo debe tener este aspecto

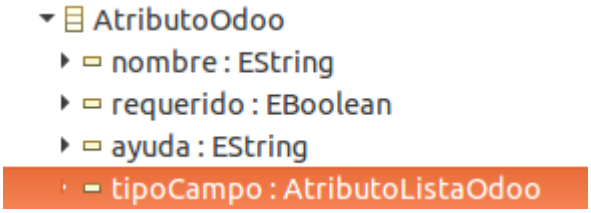


Ilustración 19 Metamodelado de relación AtributoOdoo a AtributoListaOdoo. Fuente: Elaboración Propia

Meta-modelado relación RelacionOdoo a TipoRelacionOdoo

Los tipos de relación que puede tomar una RelacionOdoo deben estar limitados a los tipos de relación definidos en la enumeración TipoRelacionOdoo, para definir esto se agrega un nuevo EAtribute al meta-modelo de RelacionOdoo.

1. Tipo de campo
    a. Name: tipoRelacion
    b. EType: TipoRelacionOdoo
    c. Default Value Literal: UNOAMUCHOS

Después de realizados los cambios, el Ecore de RelacionOdoo debe tener este aspecto



Ilustración 20 Metamodelado RelacionOdoo a TipoRelacionOdoo. Fuente: Elaboración Propia

Meta-modelado definición de la raíz del metamodelo

Un metamodelo en Ecore debe tener una EClass raíz, la cual contiene una relación a todas las demás EClases que componen el Ecore, en este caso esta clase tendrá el nombre de "RequerimientoOdoo", y contendrá las siguientes ERerefences

1) Requerimiento - módulos
    a) Name: requerimiento_tiene_modulos
    b) EType: ModuloOdoo
    c) LowerBound: 0

d) UpperBound: -1 (ilimitados)

2) Requerimiento - relaciones

a) Name: requerimiento_tiene_relaciones

b) EType: RelacionOdoo

c) LowerBound: 0

d) UpperBound: -1 (ilimitados)

3) Requerimiento - Modelos

a) Name: requerimiento_tiene_modelos

b) EType: ModeloOdoo

c) LowerBound: 0

d) UpperBound: -1 (ilimitados)

4) Requerimiento - Atributos

a) Name: requerimiento_tiene_atributos

b) EType: AtributoOdoo

c) LowerBound: 0

d) UpperBound: -1 (ilimitados)

Una vez agregadas las EReferences, el metamodelo de RequerimientoOdoo debe tener este aspecto:

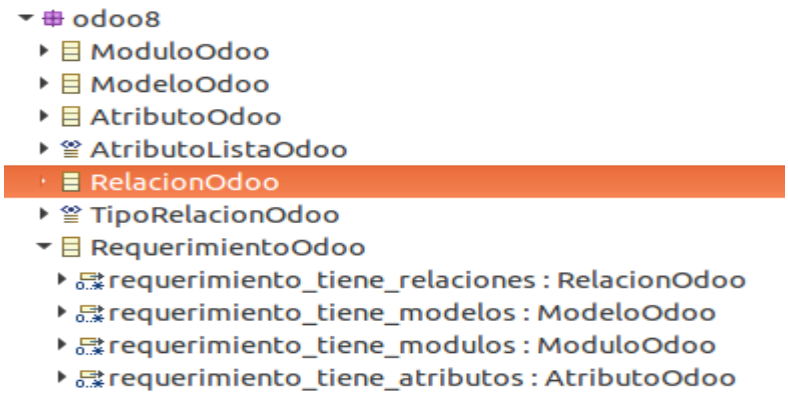


Ilustración 21 Metamodelado RequerimientoOdoo. Fuente: Elaboración Propia

Resultado final del meta-modelado

Ahora habiendo finalizado el proceso de Metamodelado el Ecore debe quedar como en la siguiente ilustración:



Ilustración 22 Ecore después de finalizare el proceso de Metamodelado. Fuente: Elaboración Propia.

4 Diseño del DSL gráfico

En este capítulo se diseñara e implementara el DSL gráfico con base en el metamodelo definido en el capítulo anterior, se verá un paso a paso del proceso incluyendo los errores cometidos y como se solucionaron.

El primer intento para el diseño del DSL grafico se realizó con el framework GMF (graphical modeling framework) de eclipse y el segundo se realizó con el editor gráfico sirius, este capítulo se distribuirá de la siguiente manera: primero una introducción a los dos editores gráficos mencionados y posteriormente el proceso que se llevó a cabo con cada uno de ellos.

GMF

El objetivo de GMF es proveer soporte para la implementación de editores basados en diagramas visuales más importantes de EMF con base en la estructura proporcionada por el robusto y ampliamente utilizado (GEF) eclipse Graphical Modeling Framework. Con el fin de implementar un editor gráfico para un lenguaje de modelado particular con GMF los desarrolladores necesitan construir un número de modelos intermedios que especifican la sintaxis visual del lenguaje. Estos modelos intermedios son entonces combinados y eventualmente consumidos por un conjunto de transformaciones modelo-a-texto con el fin de generar los plug-ins de eclipse que contienen el actual código Java que implementa el editor. Más específicamente como se muestra en la siguiente figura los desarrolladores tienen que

especificar tres modelos adicionales aparte del metamodelo, el modelo gráfico, el modelo de mapeo y el modelo de herramientas [23].

- Modelo gráfico: Específica las formas que serán usadas en el editor por ejemplo rectángulos representan clases, líneas representan relaciones etc.
- Modelo de Herramientas: especifica que herramientas estarán disponibles en la paleta del editor.
- Modelo de Mapeo: Especifica como los constructores del modelo grafico son mapeados desde el metamodelo.

Una vez que esos tres modelos han sido definidos estos son combinados en un modelo generador(genmodel) el cual es a su vez consumido por una serie de transformaciones modelo-a-texto que eventualmente generan el código JAVA y los plug-ins de eclipse que a la larga terminan siendo el editor ejecutable en concreto.

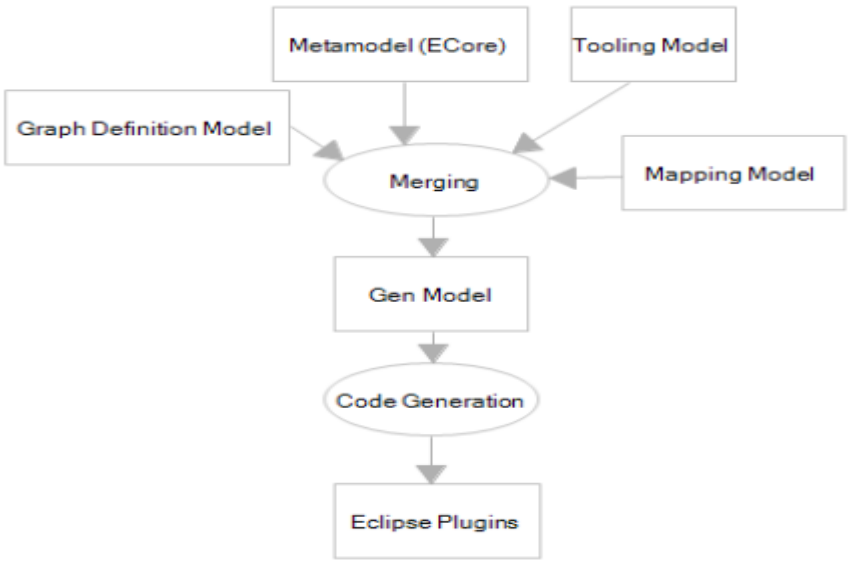


Ilustración 23 Proceso para la implementación de un editor gráfico. Fuente: Raising the Level of Abstraction in the Development of GMF-based Graphical Model Editors

En términos de soporte de la herramienta una vez que el metamodelo del DSL ha sido especificado GMF provee un wizard que es capaz de derivar primeras versiones de los modelos intermedios(grafico, de herramientas y de mapeo) directamente del metamodelo creado previamente en ecore.

Después de ser definidos los modelos intermedios estos pueden ser redefinidos o corregidos utilizando un conjunto de editores integrados basados en árboles, sin embargo hay algunos problemas relacionados a ese proceso, primero el wizard encargado de generar las versiones iniciales de los modelos intermedios funciona bien solo con metamodelos muy simples, esto es entendible ya que el wizard tiene solo una pequeña y limitada cantidad de información a su disposición. Incluso más allá, refinar los modelos intermedios requiere un buen conocimiento del respectivo metamodelo que está lejos de ser trivial.

Finalmente en caso de que el desarrollador cometa algún error de lógica en alguno de los modelos implicados GMF produce errores que son difíciles de comprender para un principiante o incluso la generación de código puede generar código erróneo.

Todos los aspectos mencionados anteriormente hacen particularmente difícil la implementación de GMF a la hora de adoptar MDE, sin embargo teniendo en cuenta que la implementación de un editor es uno de los primeros pasos para interactuar con MDE es muy probable que se encuentren grandes dificultades desde el inicio del proceso, lo que puede llevar a desalentar futuras investigaciones.

Sirius

A continuación tenemos una breve descripción de Sirius que fue hecha tomando como base la información encontrada en "*Raising the Level of Abstraction in the Development of GMF-based Graphical Model Editors"* referenciada en [24] y en "*Sirius: A Rapid Development of DSM Graphical Editor"* referenciada en [25].

Sirius es un proyecto de eclipse que permite crear de manera sencilla una plataforma de modelamiento grafico implementando las tecnologías de moldeamiento de eclipse incluidas EMF y GMF.

Sirius fue creado por Obeo y Thales para proveer una plataforma para la ingeniería y la arquitectura basada en modelos que puede ser fácilmente adaptada para satisfacer necesidades específicas.

Basándose en un enfoque de punto de vista, Sirius hace posible equipar equipos que tienen que lidiar con arquitecturas complejas en dominios específicos.

Una plataforma de modelamiento creada con sirius está compuesta de un conjunto de editores de Eclipse (diagramas, tablas y arboles) que le permiten al usuario crear editar y visualizar modelos EMF.

Los editores son definidos por un modelo que define la estructura, el comportamiento y todas las herramientas de edición y navegación de la plataforma de trabajo. Esta descripción de una plataforma de modelamiento de sirius es interpretada dinámicamente por un tiempo de ejecución en el IDE de eclipse.

Como se mencionó anteriormente sirius soporta tres dialectos diferentes: diagramas, tablas y árboles, sin embargo nuevos dialectos pueden ser adheridos por medio de la programación. Debido a que el problema de dominio normalmente hace necesaria la

colaboración de personas con diferentes preocupaciones o puntos de vista, sirius provee posibilidad de roles y análisis de los mismos datos con un diferente punto de vista en el mismo modelo de dominio, sirius también provee herramientas para especificar puntos de vista que son relevantes para el dominio de negocio del usuario, cualquiera que sea.

Sirius provee un modelador grafico para la creación de DSM's que define conceptos y sus relaciones de manera abstracta, después de la definición de un DSM sirius permite la fácil creación de representaciones especificas concretas de estos modelos y dichas representaciones pueden ser presentadas en más de un diagrama, tabla, matriz o árbol.

Las representaciones no son estáticas y modelan ambientes por completo en los cuales los usuarios pueden crear, modificar y validar sus diseños, pueden ser organizados lógicamente en puntos de vista los cuales pueden ser habilitados o deshabilitados por los usuarios finales, con la finalidad de proveer una diferente vista del mismo modelo.

En pocas palabras sirius simplifica el producto, reduce el tiempo de diseño e incrementa rápidamente la productividad a la hora de la construcción de un editor gráfico de dominio específico.

Las fortalezas de sirius son:

- La fundación de un estándar abierto y ampliamente usado en EMF.
- Es adaptable a cualquier DSM compatible con EMF.
- Una fuerte separación entre los modelos semánticos y de representación.
- El soporte para diferentes representaciones de modelos de dominio.
- Su facilidad de uso y de rápido desarrollo.
- El alto nivel de extensión.

Los 5 conceptos más importantes a tener en cuenta cuando se trabaja con sirius de acuerdo a [24] son:

- Puntos de vista: es un elemento del núcleo de sirius que es un conjunto de representaciones de especificaciones y de representaciones de especificaciones extendidas.

- Representación: es un grupo de construcción grafica que representa datos de dominio. También describe la estructura, apariencia y comportamiento de los modelos. Hay 4 representaciones disponibles: diagramas, tablas, matrices y árboles.

- Mapeo: depende fuertemente de las representaciones e identifica un sub-conjunto de elementos semánticos del modelo que suelen aparecer en la representación e indican cómo deben ser hechas las representaciones.

- Estilo: se usa para configurar la apariencia visual de los elementos

- Herramienta: describe el comportamiento del mapeo.

Ahora con esta breve descripción de los dos editores gráficos que se tuvieron en cuenta para el desarrollo de este proyecto el siguiente paso es mostrar como fue el proceso realizado con sirius ya que este fue el editor gráfico seleccionado ya que no se pudo llegar a buen término con GMF.

Proceso con Sirius

A continuación se describirá paso a paso como fue el proceso de desarrollo del DSL Gráfico con la herramienta Sirius, aparte de la descripción textual se hará una descripción gráfica.

Creación del modelo generador de código

La primera etapa de la creación de un DSL gráfico con Sirius consiste en la creación de un modelo generador de código basado en un metamodelo Ecore, dicho generador de código agrega el soporte necesario para generar las EClasses pertenecientes al meta-modelo Ecore sobre el cual fue basado.

Un modelo generador de código agrega la lógica para soportar el diseño de un modelo basado en un meta-modelo de manera gráfica, la lógica agregada se restringe a crear, editar y eliminar EClasses de un modelo

Creación de diagrama Ecore

Los archivos Ecore pueden ser representados en un diagrama llamado "Ecore Diagram", es una representación similar al diagrama de clases del estándar UML2, al generarlo se puede entender de una manera más sencilla el meta-modelo, una vez es generado, se crean una serie de archivos y carpetas dentro del proyecto, en este caso específico bajo el nombre de Odoo8.aird.

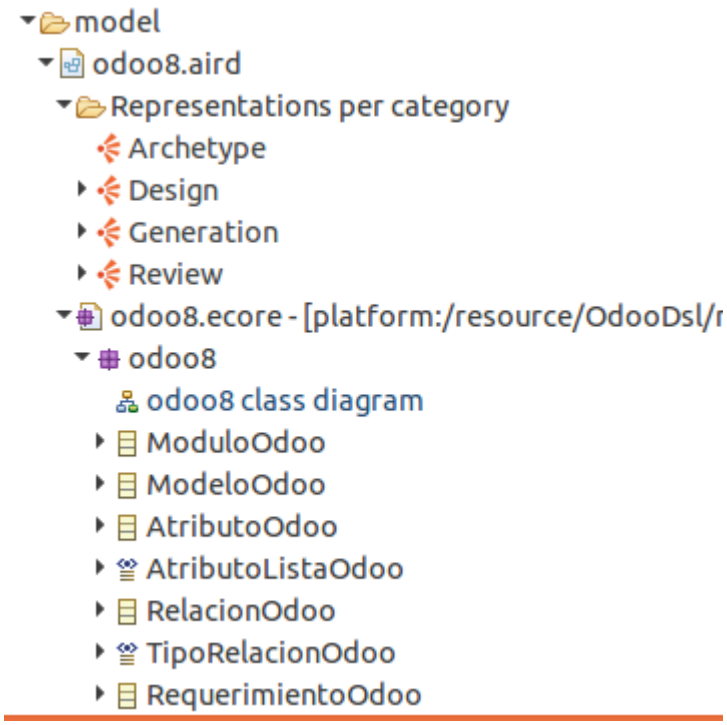


Ilustración 24 Estructura Archivo aird. Fuente: Elaboración Propia

El contenido del archivo “odoo8 class diagram” es el siguiente.

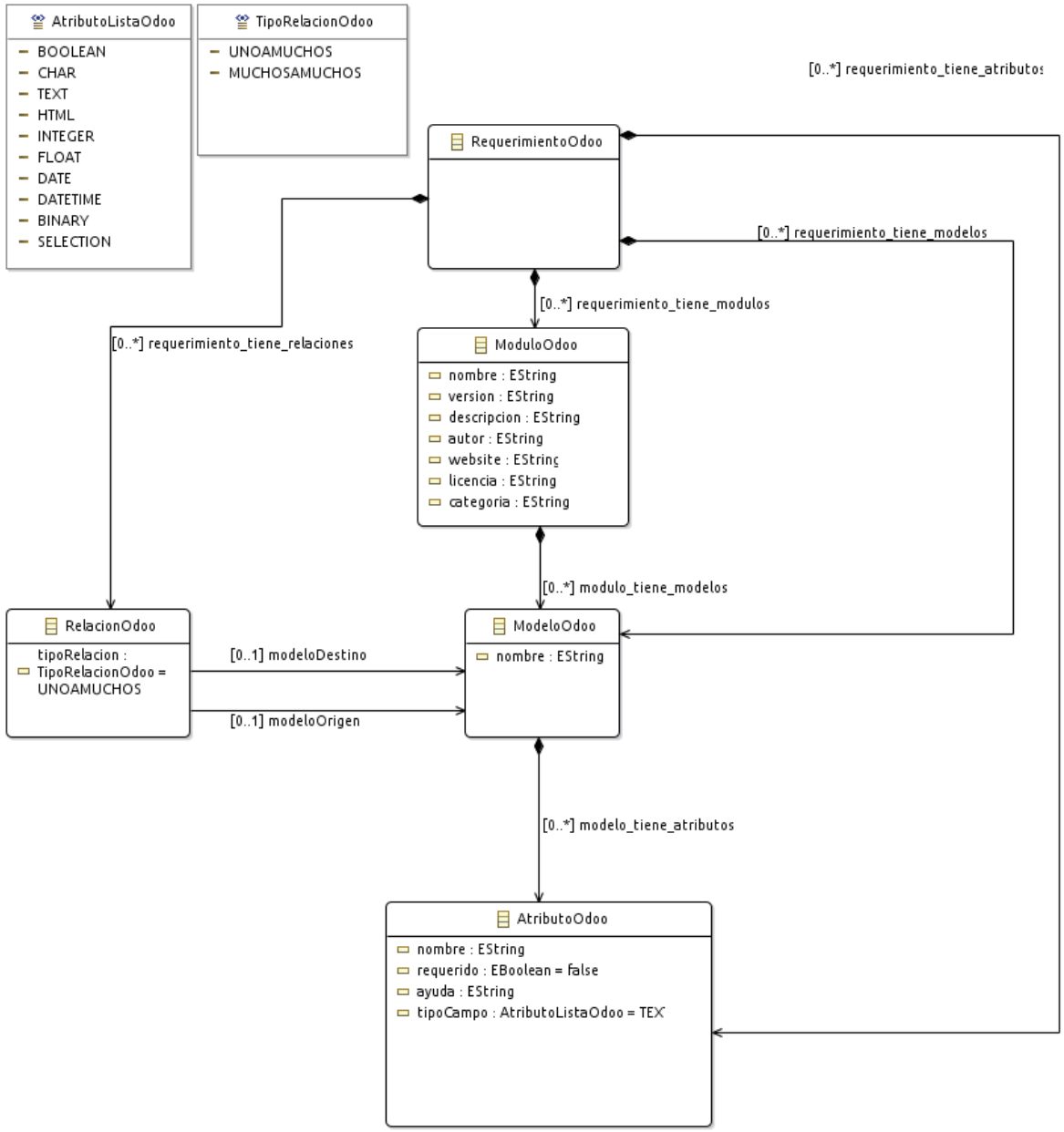


Ilustración 25 Diagrama ade Eclass de odoo8. Fuente: Elaboración Propia

Configuración de archivo genmodel

El proyecto de modelado debe contener un nuevo archivo de tipo “EMF Generator Model”, llamado “odoo8.genmodel”, el cual importa un modelo del tipo Ecore, y hace referencia el archivo odoo8.ecore creado en el capítulo anterior.

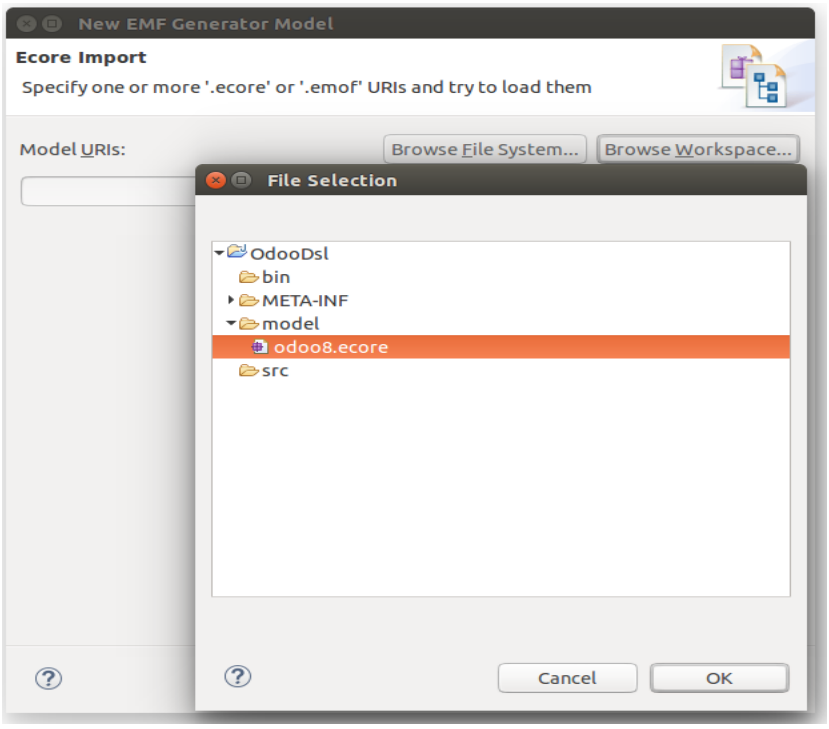


Ilustración 26 Configuración Genmodel. Fuente: Elaboración Propia

El archivo genmodel extrae todas las propiedades del metamodelo odoo 8, con el fin de generar el código necesario para la edición de un modelo, para ello es necesario configurar las siguientes características del paquete Odoo8 del archivo odoo8.genmodel:

- Base Package: odoo8

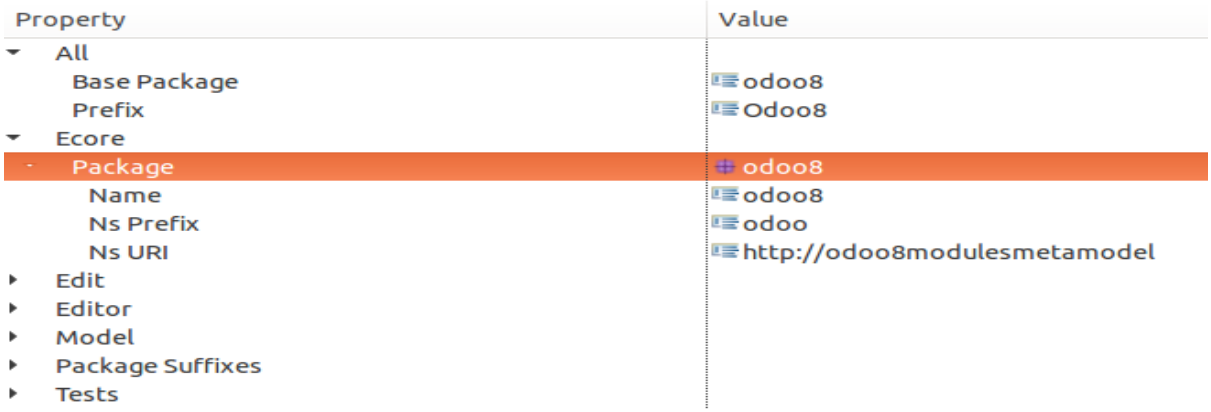


Ilustración 27 Configuración paquete Odoo8 del archivo odoo8.genmodel. Fuente: Elaboración Propia

Generación de proyectos generadores de código

El archivo genmodel permite generar tres proyectos, cada uno de estos cumple una funcionalidad dentro del DSL

1. OdooDsl.edit: Contiene las fábricas del editor, estas contienen el código necesario para visualizar y crear nuevos elementos.
2. OdooDsl.editor: Implementa las fábricas creadas en el proyecto Odoo8.edit, permitiendo la creación y edición de modelos.
3. OdooDsl.test: Proporciona el código necesario para realizar pruebas unitarias sobre los demás proyectos.

Una vez se ejecuta la orden "Generate All" usando el archivo odoo8.genmodel, en nuestro entorno de desarrollo se muestran los tres proyectos antes mencionados

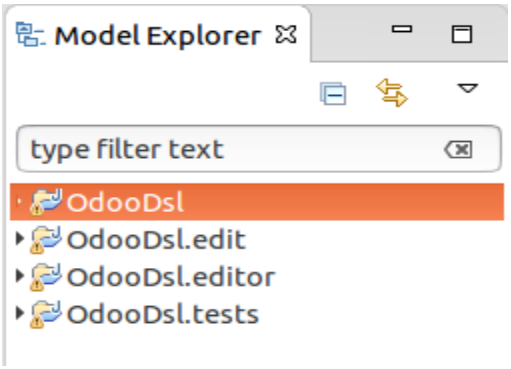


Ilustración 28 Proyectos Generados a partir del modelo generador de código. Fuente: Elaboración Propia

Prueba del proyecto OdooDsl.editor

Con el fin de validar el modelado hecho hasta este punto, se ejecuta el proyecto OdooDsl.editor como una aplicación eclipse, posteriormente se genera un nuevo proyecto de modelado "Modeling Project" llamado "Odoo8EditorTest", se agrega un nuevo "Odoo 8 Model" llamado testing.odoo8 y se especifica que el objeto de modelado sea RequerimientoOdoo.

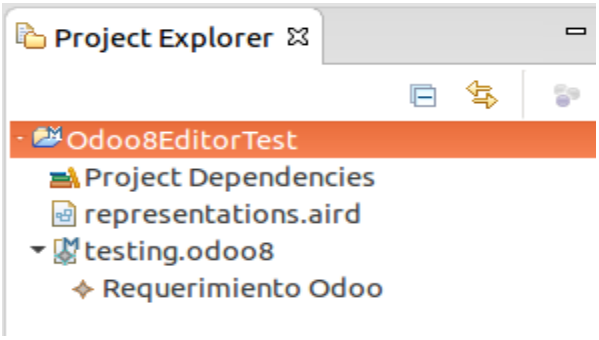


Ilustración 29 Estructura Proyecto de Prueba. Fuente: Elaboración Propia

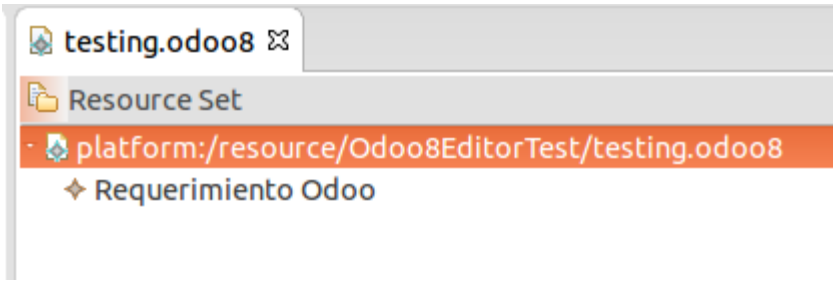


Ilustración 30 Estructura del archivo testing.odoo8. Fuente: Elaboración Propia

A continuación se genera una estructura dentro de testing.odoo8, se añaden dos nuevos módulos, cada uno con un modelo y cada modelo con un atributo, adicionalmente se crea una relación entre los dos modelos existentes

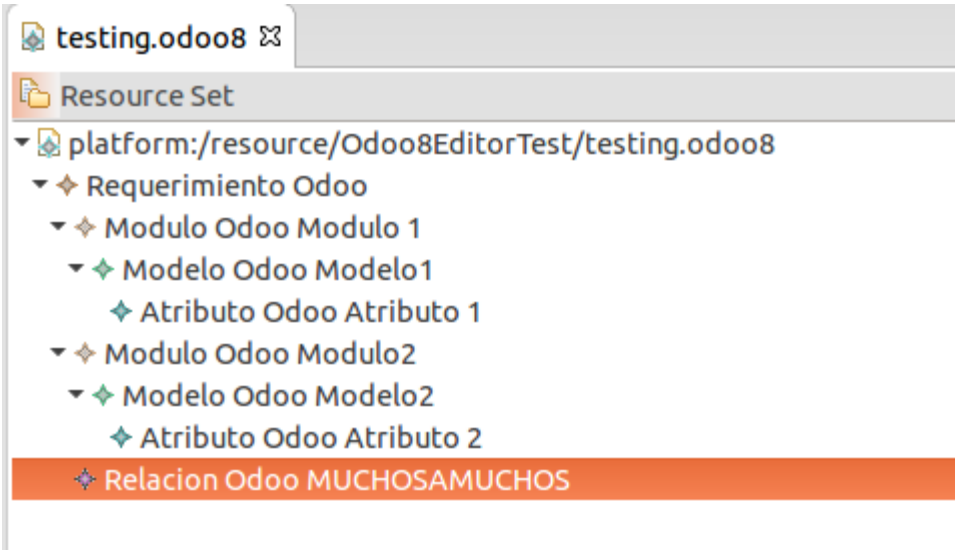


Ilustración 31 Prueba del modelo Generador. Fuente: Elaboración Propia

Creación del viewpoint

La creación de un DSL gráfico usando Sirius se basa en la generación de Viewpoints, los cuales pueden contener diferentes tipo de componentes listados a continuación:

- Representaciones
- Plantillas
- Validaciones
- Importaciones
- Extensiones

El componente de viewpoint que se debe usar para el DSL gráfico de Odoo 8, es una representación. Una representación hace una selección de elementos del modelo, permite crear, editar y mostrar las entidades del mismo.

Basado en distintas configuraciones, Sirius provee varias opciones de lenguajes de consulta como Acceleo que pueden ser ejecutadas sobre los modelos, las instancias del modelo que son acopladas por dichas consultas son las instancias que una representación muestra.

Una representación puede mostrar diferentes tipos de descripciones, una descripción es una forma de desplegar datos, las representaciones de Sirius son las siguientes

- Diagrama
- Tablas
- Árboles
- Diagramas de secuencia

La descripción que se debe usar para el DSL gráfico de Odoo 8, es un Diagrama, ya que permite mostrar cualquier elemento sin ninguna restricción gráfica en el espacio de dibujo

Creación del ViewPoint Specification Project

Se crea un nuevo proyecto del tipo ViewPoint Specification Project, se le llama "Odoo8Dsl.design", dentro del proyecto se agrega un nuevo Viewpoint Specification Model y se le llama Odoo8Dsl.odesign

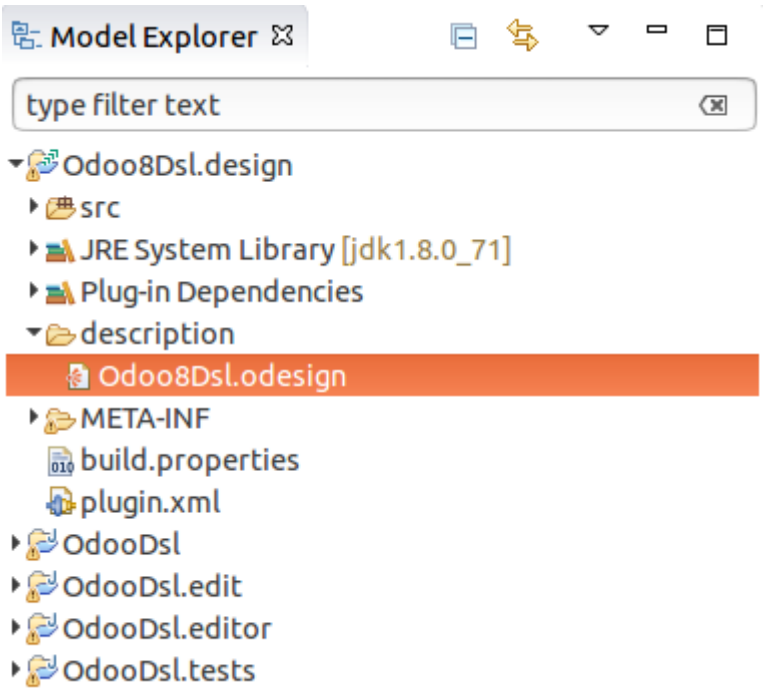


Ilustración 32 Viewpoint specification project. Fuente: Elaboración Propia

Configuración de dependencias

El Viewpoint Specification Project hace uso de los proyectos generados por el generator model, es necesario agregar los proyectos OdooDsl.edit, OdooDsl.editor, OdooDsl.tests y OdooDsl como dependencias del proyecto Odoo8Dsl.design

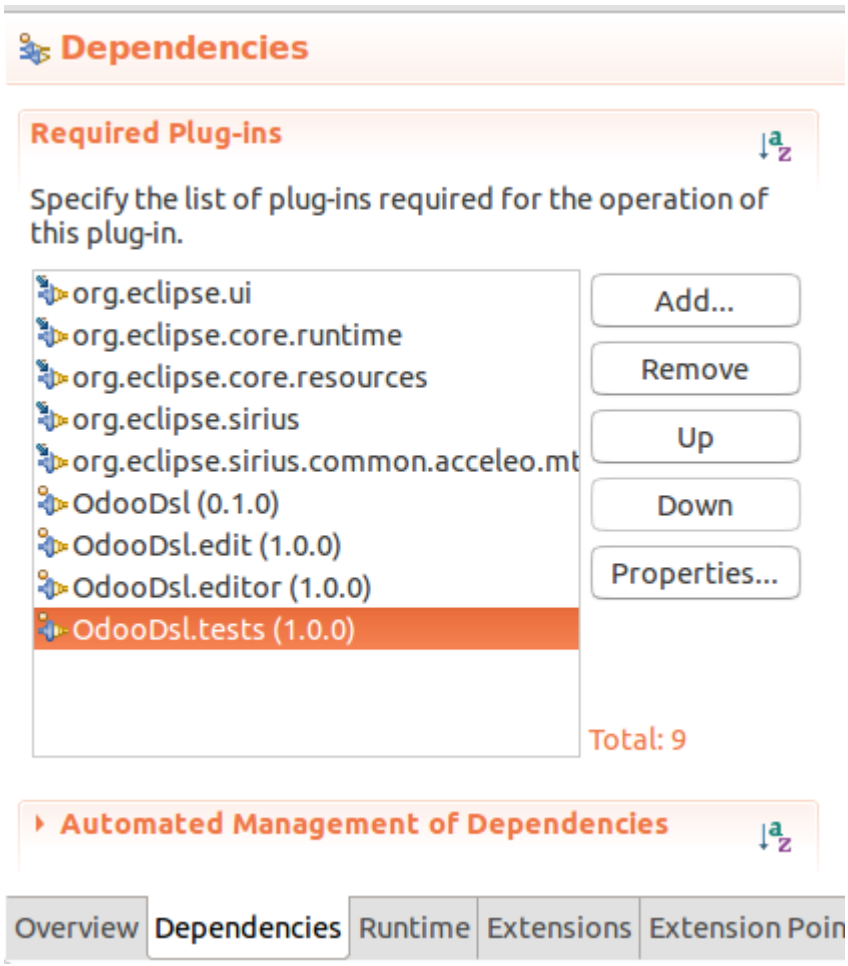


Ilustración 33 Configuración de las dependencias. Fuente: Elaboración Propia

Configuración del viewpoint

Al generar un viewpoint, existen ciertas propiedades que deben ser configuradas, en el caso específico del DSL gráfico que está siendo creado se configuran las siguientes características.

1. ID:

- Función: Identificador único de viewpoint
- Valor: odoo8_viewpoint

Label:

- Función: Etiqueta con el nombre que se muestra al usuario
- Valor: Odoo 8 Viewpoint

Model File Extension:

- Función: Extensión con la que se guarda el modelo
- Valor: odoo8

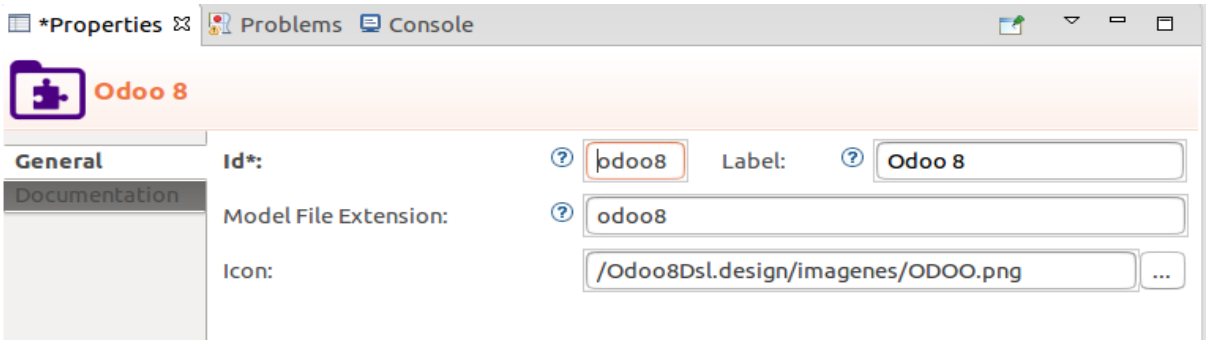


Ilustración 34 Configuraciones del viewpoint. Fuente: Elaboración Propia

Configuración de la representación

A continuación se agrega una nueva representación de tipo Diagrama al viewpoint, y se configuran las siguientes características usando los valores correspondientes:

1. ID

- Función: Identificador único del diagrama
- Valor: odoo8_diagrama_base

Domain Class

- Función: Determina la Eclass Base, todos los hijos de esta EClass serán candidatos para ser representados por el diagrama.
- Valor: odoo8.RequerimientoOdoo (Todos los hijos de un Requerimiento Odoo)

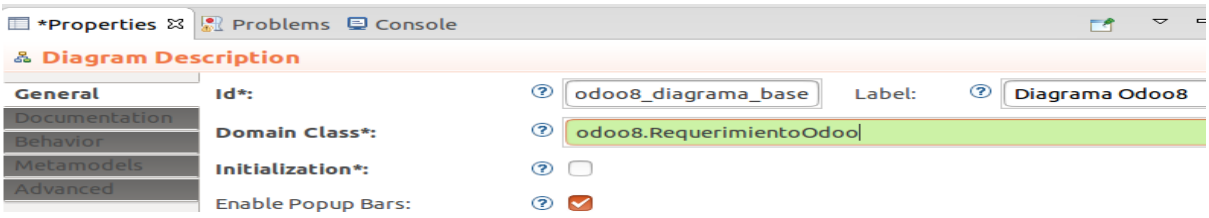


Ilustración 35configuracion de la representación tipo Diagrama. Fuente: Elaboración Propia

Configuración de la representación de nodos y aristas en el viewpoint

Ahora se describirá la configuración realizada a la representación de diagrama para lograr mostrar los elementos contenidos en el metamodelo Ecore, la representación se divide en dos partes.

La primera parte es la configuración de los nodos, algunas de las Eclasses modeladas en el archivo Odoo8.ecore, tienen una representación de nodo, estas clases se listan a continuación

1. ModuloOdoo
2. ModeloOdoo
3. AtributoOdoo

La segunda parte consiste en la configuración de las aristas, algunas de las EClasses y EReferences modeladas en el archivo Odoo8.ecore, tienen una representación de arista, es decir representan una conexión entre dos nodos. Las EClasses y EReferences se listan a continuación

1. EClass “RelacionOdoo”
2. EReference “modulo_tiene_modelos”
3. EReference “modelo_tiene_atributos”

Representación de nodos

Representación nodo ModuloOdoo

Usando como base el default layer de la representación de diagrama, se agrega un nuevo elemento de diagrama del tipo “Node”, las configuraciones para el elemento “Node” son las siguientes.

1. ID

- Función: Identificador único del tipo nodo
- Valor: NModuloOdoo

Domain Class:

- Función: Establece la EClass del archivo odoo8.ecore a representar
- Valor: odoo8.ModuloOdoo

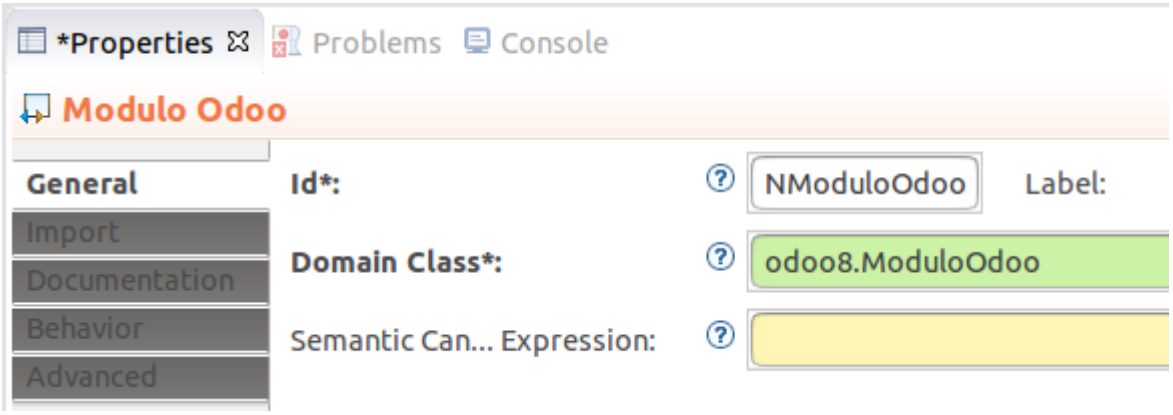


Ilustración 36 Configuraciones Nodo NModulosOdoo. Fuente: Elaboración Propia

A su vez al elemento de diagrama Node con id "NModuloOdoo" se le agrega un nuevo estilo de representación del tipo "WorkSpace Image" usando las siguientes configuraciones.

1. Image Path

- Función: Imagen que representa el nodo NModuloOdoo
- Valor: Imagen acorde tomada de un repositorio de imágenes libre.

Label Expresion

- Función: Define la propiedad a mostrar cómo label para el nodo NModuloOdoo, esta propiedad debe ser una perteneciente al meta-modelo odoo8.ecore.
- Valor: "feature:nombre" (propiedad nombre de la EClass ModuloOdoo).

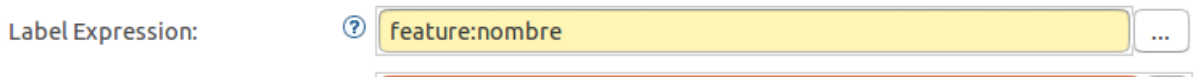


Ilustración 37 Configuraciones del estilo para el nodo NModuloOdoo. Fuente: Elaboración Propia

Representación nodo ModeloOdoo

Usando como base el default layer de la representación de diagrama, se agrega un nuevo elemento de diagrama del tipo “Node”, las configuraciones para el elemento “Node” son las siguientes:

1. ID

- Función: Identificador único del tipo nodo
- Valor: NModeloOdoo

2. Domain Class:

- Función: Establece la EClass del archivo odoo8.ecore a representar
- Valor: odoo8.ModeloOdoo

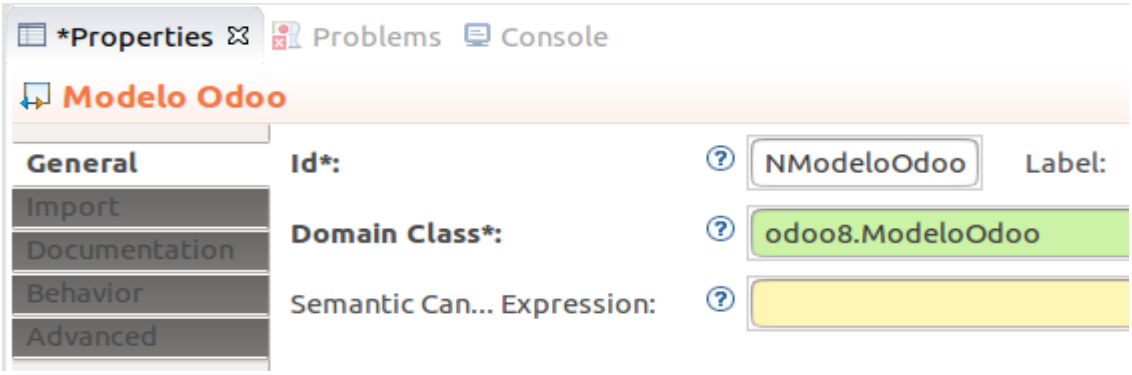


Ilustración 38 Configuraciones Nodo NModeloOdoo. Fuente: Elaboración Propia

A su vez al elemento de diagrama Node con id “NModeloOdoo” se le agrega un nuevo estilo de representación del tipo “WorkSpace Image” usando las siguientes configuraciones:

1. Image Path

- Función: Imagen que representa el nodo NModeloOdoo
- Valor: Imagen acorde tomada de un repositorio de imágenes libre

2. Label Expresion

- Función: Define la propiedad a mostrar cómo label para el nodo NModuloOdoo, esta propiedad debe ser una perteneciente al meta-modelo odoo8.ecore
- Valor: “feature:nombre” (propiedad nombre de la EClass ModeloOdoo)

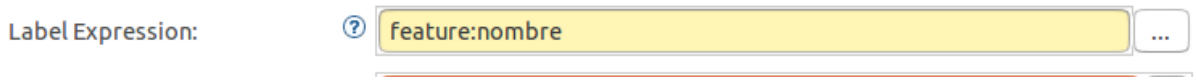


Ilustración 39 Configuraciones del estilo para el nodo NModeloOdoo. Fuente: Elaboración Propia

Representación nodo AtributoOdoo

Usando como base el default layer de la representación de diagrama, se agrega un nuevo elemento de diagrama del tipo “Node”, las configuraciones para el elemento “Node” son las siguientes:

1. ID

- Función: Identificador único del tipo nodo
- Valor: NAtributoOdoo

2. Domain Class:

- Función: Establece la EClass del archivo odoo8.ecore a representar
- Valor: odoo8.AtributoOdoo

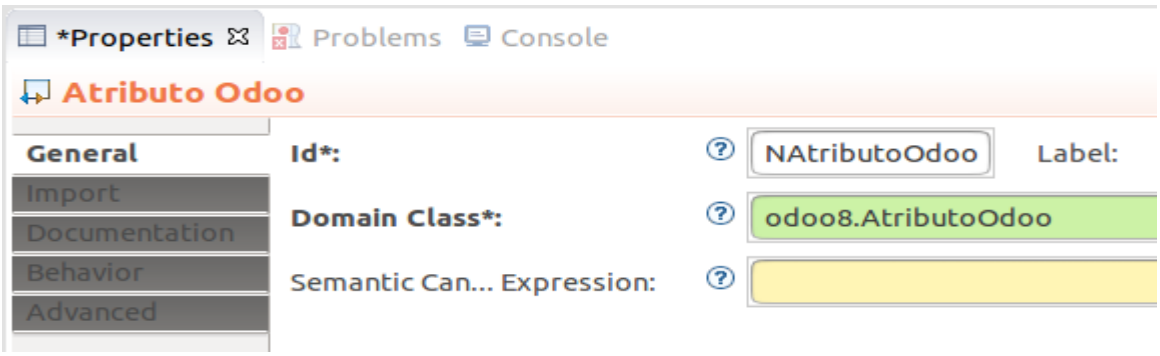


Ilustración 40 Configuraciones Nodo NAtributoOdoo. Fuente: Elaboración Propia

A su vez al elemento de diagrama Node con id “NAtributoOdoo” se le agrega un nuevo estilo de representación del tipo “WorkSpace Image” usando las siguientes configuraciones.

1. Image Path

- Función: Imagen que representa el nodo NAtributoOdoo
- Valor: Imagen acorde tomada de un repositorio de imágenes libre

2. Label Expresion

- Función: Define la propiedad a mostrar cómo label para el nodo NAtributoOdoo, esta propiedad debe ser una perteneciente al meta-modelo odoo8.ecore
- Valor: “feature:nombre” (propiedad nombre de la EClass AtributoOdoo)

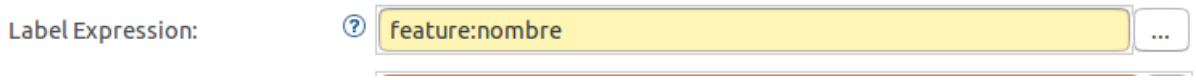


Ilustración 41Configuraciones del estilo para el nodo NAtributoOdoo. Fuente: Elaboración Propia

En este punto, el archivo Odoo8Dsl.odesign debe tener el siguiente aspecto:

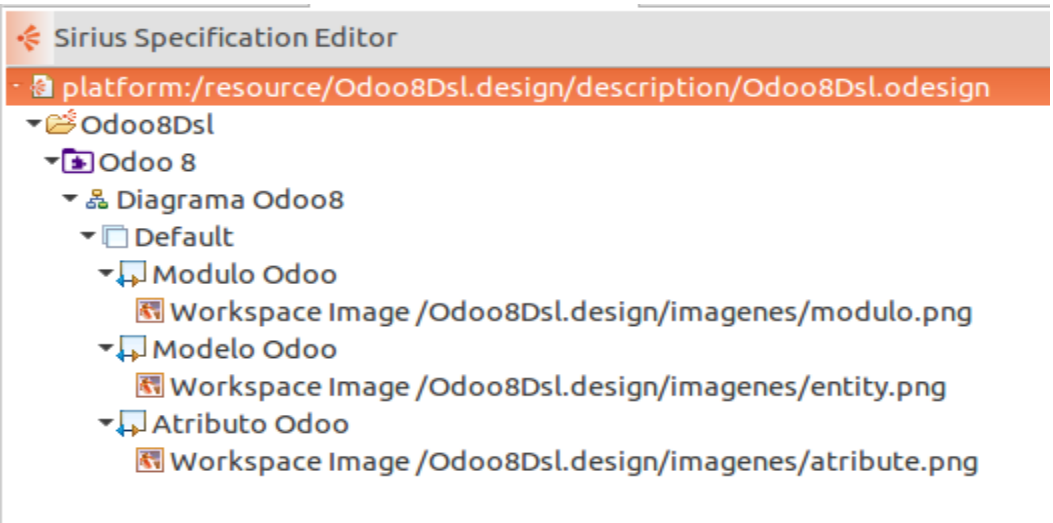


Ilustración 42 Viewpoint con nodos configurados. Fuente: Elaboración Propia

Representación de aristas

La representación de diagrama permite mostrar dos tipos de aristas, la primera de ellas es basada en EReferences esta muestra todas las EReferences usando un nombre, el segundo tipo de arista son las aristas basadas en EClasses, estas toman ciertas EClasses y las representan como aristas.

Representación de EReference “modulo_tiene_modelos”

Usando como base el default layer de la representación de diagrama, se agrega un nuevo elemento de diagrama del tipo “RelationBasedEdge”, las configuraciones para el elemento son las siguientes.

1. ID

- Función: Identificador único de la relación
- Valor: RModuloModelo

  Source Mapping
- Función: Determina el tipo de nodo desde el cual inicia la relación
- Valor: “Modulo Odoo” (Nodo tipo NModuloOdoo)

  Target Mapping
- Función: Determina el tipo de nodo donde termina la relación
- Valor: “ModeloOdoo” (Nodo tipo NModeloOdoo)

  Target Finder Expression
- Función: Define una consulta sobre el modelo, los resultados de esta consulta son las relaciones que se mostrarán en el diagrama
- Valor: feature:modulo_tiene_modelos (Todas las EReferences de nombre modulo_tiene_modelo)

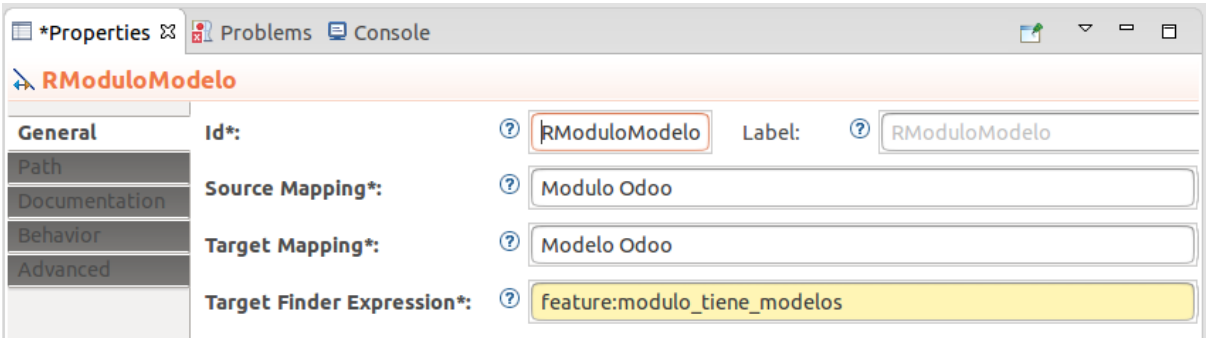


Ilustración 43Configuraciones del estilo para el nodo NAtributoOdoo. Fuente: Elaboración Propia

Representación de EReference "modelo_tiene_atributos"

Usando como base el default layer de la representación de diagrama, se agrega un nuevo elemento de diagrama del tipo "RelationBasedEdge", las configuraciones para el elemento son las siguientes.

1. ID

- Función: Identificador único de la relación
- Valor: RModeloAtributo

2. Source Mapping

- Función: Determina el tipo de nodo desde el cual inicia la relación
- Valor: "Modelo Odoo" (Nodo tipo NModeloOdoo)

3. Target Mapping

- Función: Determina el tipo de nodo donde termina la relación
- Valor: "AtributoOdoo" (Nodo tipo NAtributoOdoo)

4. Target Finder Expression

- Función: Define una consulta sobre el modelo, los resultados de esta consulta son las relaciones que se mostrarán en el diagrama
- Valor: feature:modelo_tiene_atributos (Todas las EReferences de nombre modelo_tiene_atributos)

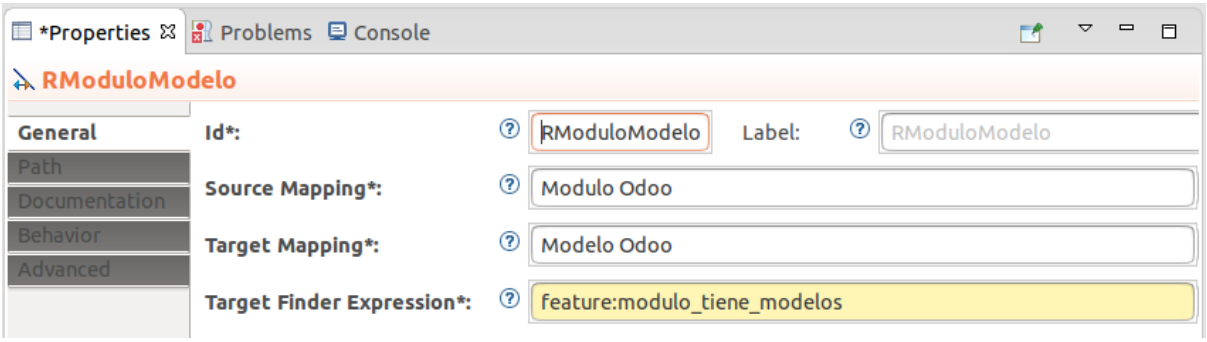


Ilustración 44 Configuración de relación modulo_tiene_modelos. Fuente: Elaboración Propia

Representación de EClass RelacionOdoo

Esta relación es distinta a las anteriores ya que está basada en una EClass y no en una EReference, por lo tanto su elemento de diagrama es distinto, así que al default layer se agrega un nuevo elemento de diagrama del tipo "ElementBasedEdge", las configuraciones para el elemento son las siguientes:

1. ID

- Función: Identificador único de la relación
- Valor: RModeloModelo

  Domain Class

- Función: EClass a ser representada como una relación
- Valor: odoo8.RelacionOdoo

  Source Mapping

- Función: Determina el tipo de nodo desde el cual inicia la relación
- Valor: "Modelo Odoo" (Nodo tipo NModeloOdoo)

  Source Finder Expression

- Función: Define una consulta sobre el modelo, el resultado de esta consulta determina la propiedad source de la relación.

- Valor: [self.modeloOrigen/] (atributo modeloOrigen de la EClass RelacionOdoo usando el lenguaje Acceleo)

  Target Mapping

- Función: Determina el tipo de nodo donde termina la relación
- Valor: “Modelo Odoo” (Nodo tipo NModeloOdoo)

  Target Finder Expression

- Función: Define una consulta sobre el modelo, el resultado de esta consulta determina la propiedad target de la relación.
- Valor: [self.modeloOrigen/] (atributo modeloDestino de la EClass RelacionOdoo usando el lenguaje Acceleo)

  Semantic Candidates Expression

- Función: Define una consulta sobre el modelo, los resultados de esta consulta determinan los candidatos a ser representados como una relación.
- Valor: feature:requerimiento_tiene_relaciones (todas las relaciones existentes dentro del RequerimientoOdoo)

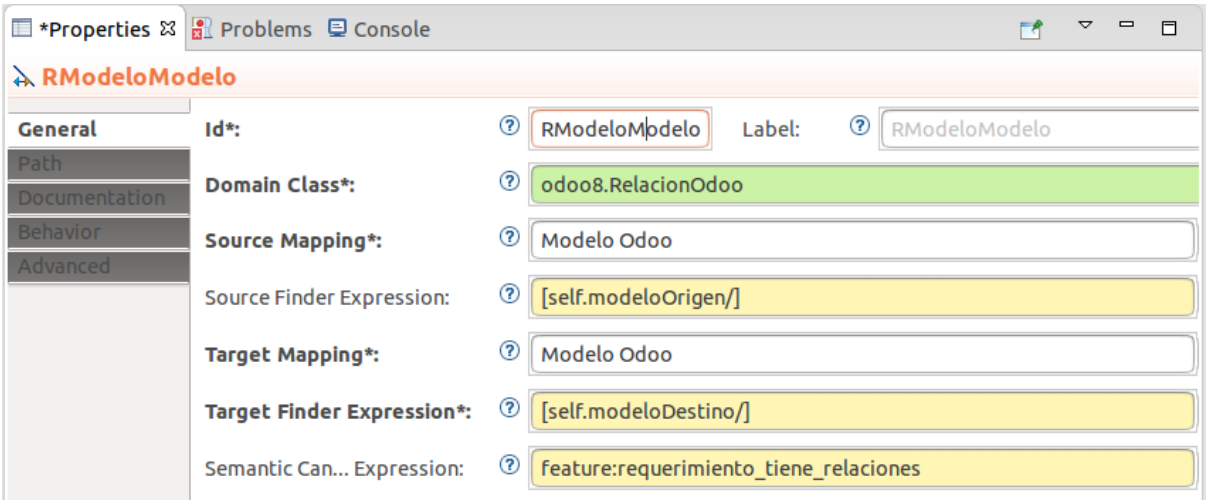


Ilustración 45configuraciones de la EClass RelacionOdoo. Fuente: Elaboración Propia

Definición de estilos condicionales

Una característica sin duda versátil de las representaciones de sirius, es la posibilidad de definir estilos basados en una condición, es decir si un elemento del modelo tiene una característica que cumple con una condición, será representada gráficamente con un estilo distinto al definido por defecto.

En el caso de la RelacionOdoo, es necesario definir dos estilos diferentes dependiendo de la propiedad “tipoRelacion”, si esta toma el valor de “UNOAMUCHOS”, la representación debe ser una flecha sencilla, de lo contrario, si esta toma el valor “MUCHOSAMUCHOS”, la representación debe ser una flecha bidireccional.

Para lograr lo mencionado anteriormente se agrega a la relación RModeloModelo un nuevo “Conditional Style” para el caso en que “tipoRelacion” tome el valor “UNOAMUCHOS”, y se configura la siguiente propiedad:

1. Predicate Expression

- Función: Define una consulta sobre el modelo, la cual debe retornar un valor booleano, en caso de que el booleano sea “1” ó “true” se aplicará el estilo a la representación
- Valor: [self.tipoRelacion.toString() = 'UNOAMUCHOS'/] (si el valor de la propiedad tipoRelacion es igual a “UNOAMUCHOS” la condición se cumple, usando el lenguaje Acceleo)

Se agrega a la relación RModeloModelo un nuevo “Conditional Style” para el caso en que “tipoRelacion” tome el valor “MUCHOSAMUCHOS”, y se configura la siguiente propiedad:

1. Predicate Expression

- Función: Define una consulta sobre el modelo, la cual debe retornar un valor booleano, en caso de que el booleano sea “1” ó “true” se aplicará el estilo a la representación

- Valor: [self.tipoRelacion.toString() = MUCHOSAMUCHOS/] (si el valor de la propiedad tipoRelacion es igual a “MUCHOSAMUCHOS” la condición se cumple, usando el lenguaje Acceleo)

En este último paso se debe modificar el estilo con las siguientes características.

1. Source Arrow

- Función: Define la forma gráfica de la relación en el extremo del origen de la relación
- Valor: InputArrow

  Target Arrow

- Función: Define la forma gráfica de la relación en el extremo del destino de la relación
- Valor: InputArrow

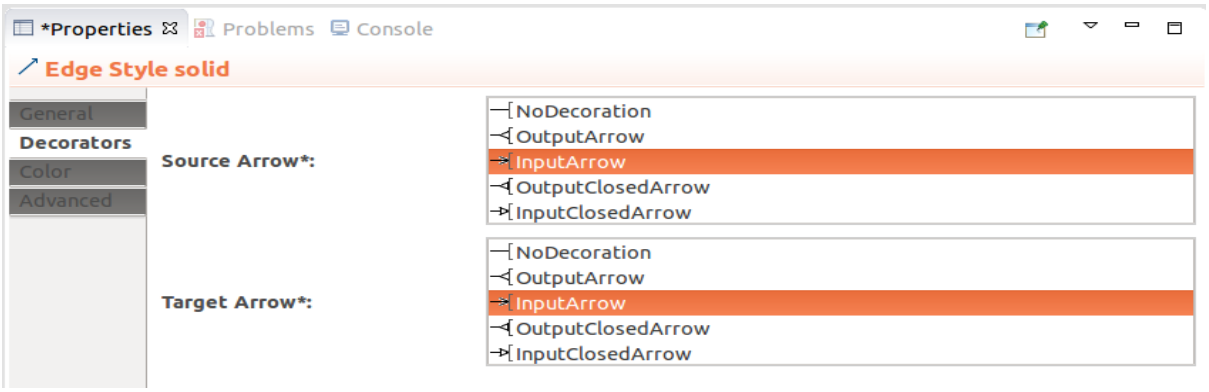


Ilustración 46 Configuración estilos condicionales. Fuente: Elaboración Propia

En este punto, la configuración del viewpoint para la visualización de los modelos ha sido completada y tiene el siguiente aspecto:

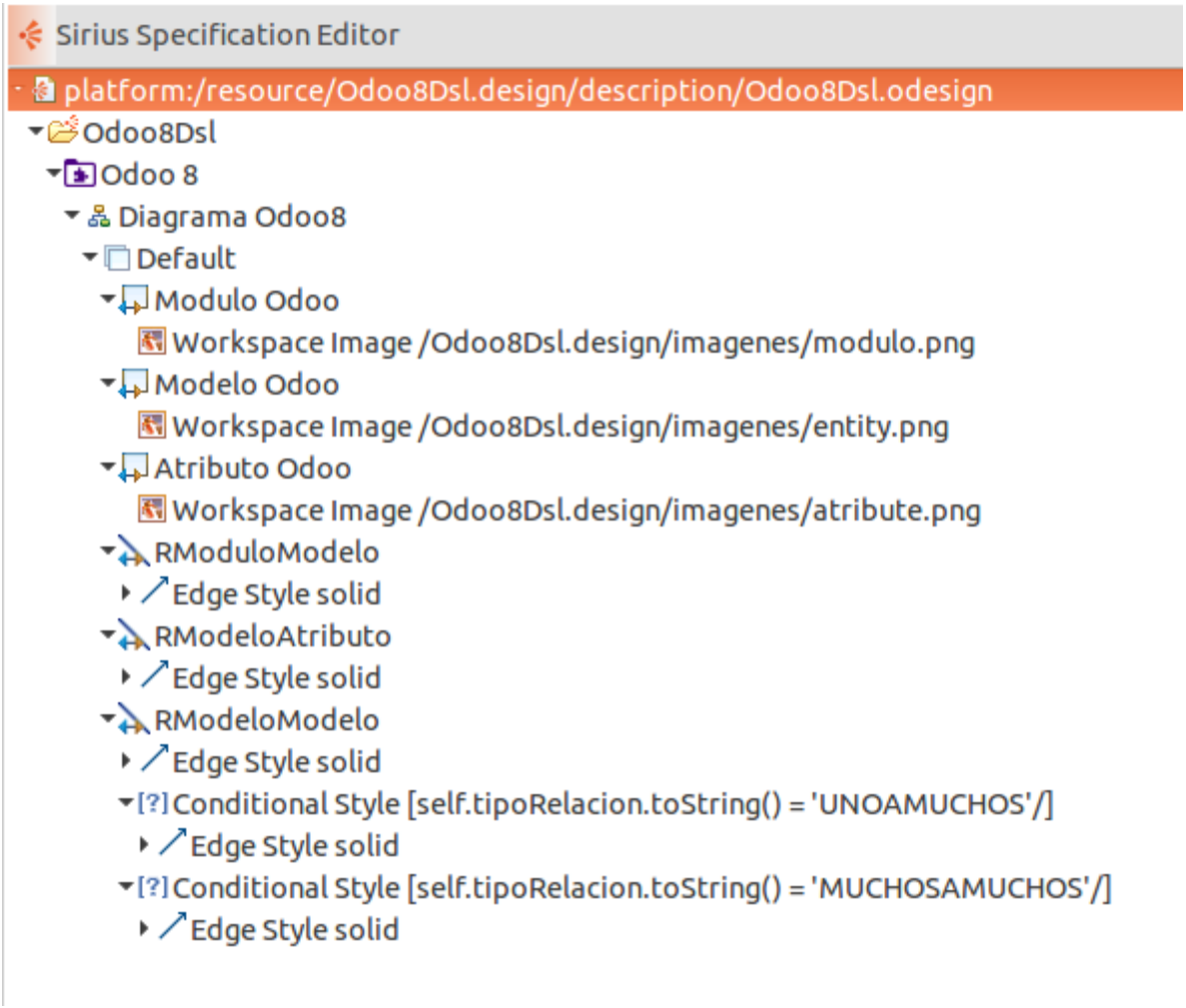


Ilustración 47configguracion de visualización del viewpoint. Fuente: Elaboración Propia

Prueba del funcionamiento del viewpoint

Para esto se debe ejecutar de nuevo el proyecto OdooDsl.editor, en el proyecto creado anteriormente “Odoo8EditorTest” se selecciona el viewpoint “odoo8”, posteriormente, en el modelo “testing.odoo8” se agrega una nueva representación “Diagrama Odoo 8”, y se debe ver de la siguiente manera:

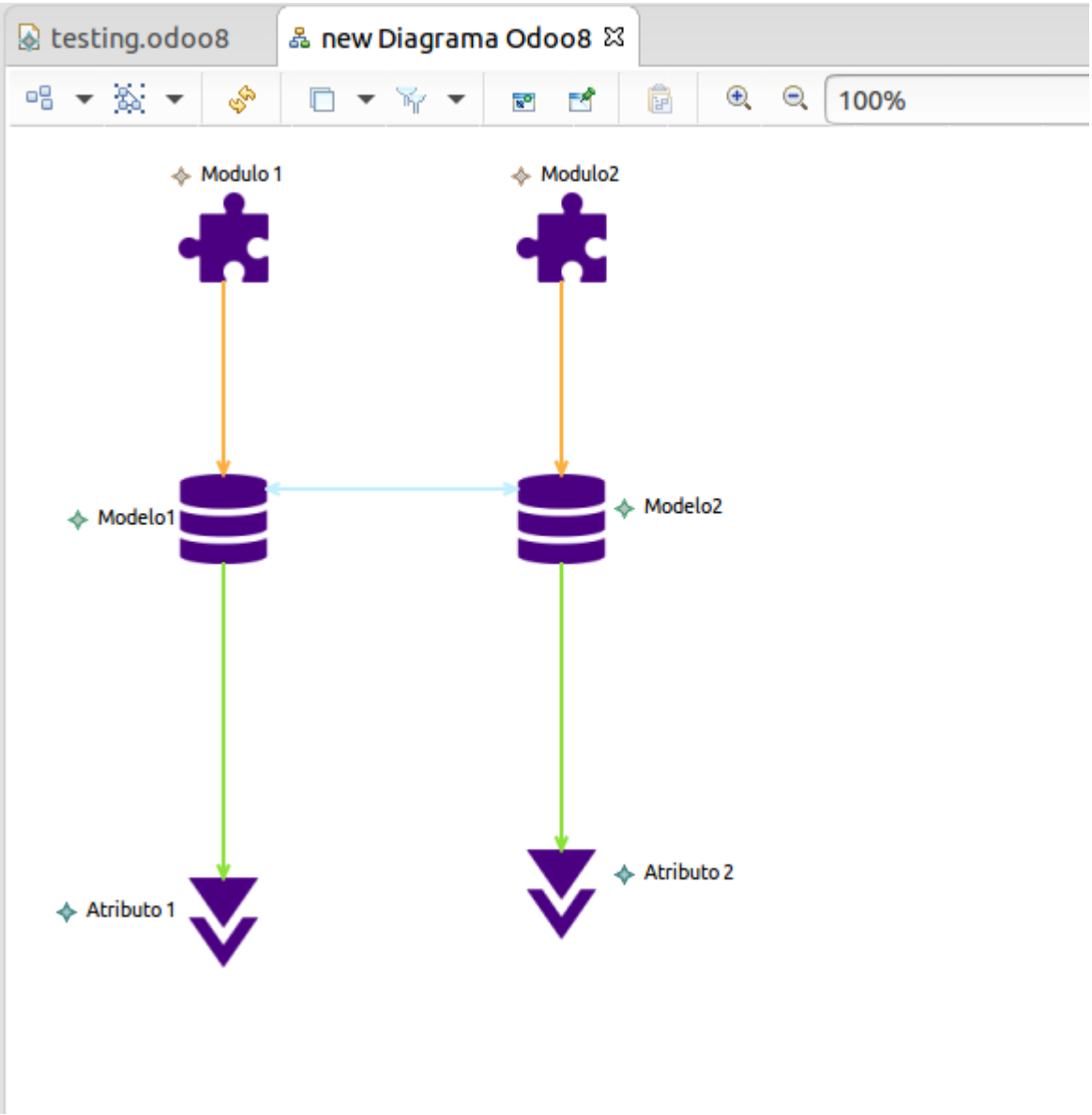


Ilustración 48Prueba Visualización Viewpoint. Fuente: Elaboración Propia

Configuración de la barra de herramientas en el viewpoint

En esta sección se describen las configuraciones aplicadas al viewpoint para la creación de una barra de herramientas que permitan agregar módulos, modelos, atributos y sus relaciones.

La primera parte consiste en la creación de las herramientas para los nodos del DSL contemplados en la siguiente lista:

1. ModuloOdoo

2. ModeloOdoo
3. AtributoOdoo

La segunda parte consiste en la creación de las herramientas para las relaciones modeladas en el DSL contempladas en la siguiente lista:

1. EClass “RelacionOdoo”
2. EReference “modulo_tiene_modelos”
3. EReference “modelo_tiene_atributos”

Para la creación de las herramientas, se debe agregar una nueva sección de herramientas llamada “herramientas” al default layer.

Creación de herramientas para los nodos

La creación de las herramientas para cada nodo contempla dos fases, la primera de ellas la configuración, donde se defina el tipo de nodo que una herramienta es capaz de crear, la segunda fase contempla el proceso que debe seguir una herramienta una vez se da la orden de crear un elemento en el entorno, esto se hace a través de la operación “Begin” la cual se llama al inicio de la creación de la instancia.

Creación de herramienta para ModuloOdoo

Se agrega a la sección de herramientas un nuevo elemento de creación del tipo “Node Creation”, las configuraciones para la herramienta son las siguientes:

1. ID

- Función: Define un identificador único para la herramienta
- Valor: CNModulo

Node Mappings

- Función: Define el tipo de nodo a agregar al entorno
- Valor: NModuloOdoo

A la operación “Begin” de la herramienta se agrega una nueva operación “CreateInstance”, esta indica la creación de una instancia nueva en el modelo, se define su comportamiento usando las siguientes características:

1. Reference Name

- Función: Indica el nombre de la referencia en la cual se va a agregar la instancia
- Valor: requerimiento_tiene_modulos

  Type Name

- Función: Indica el tipo de instancia a crear
- Valor: odoo8.ModuloOdoo

  Variable Name

- Función: Indica el nombre de la variable cuando se esté creando la instancia, esta variable puede ser usada usando extensiones de código Java
- Valor: instance

La estructura de la herramienta de creación de ModuloOdoo se muestra en la siguiente imagen:

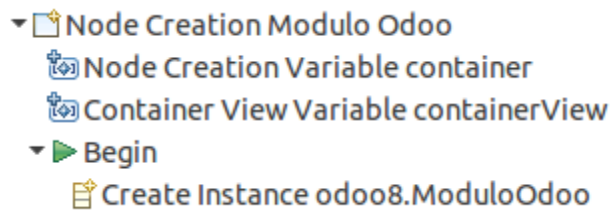


Ilustración 49 Estructura de la herramienta de creación de modulo. Fuente: Elaboración Propia

Creación de herramienta para ModeloOdoo

Se agrega a la sección de herramientas un nuevo elemento de creación del tipo “Node Creation”, las configuraciones para la herramienta son las siguientes:

1. ID

- Función: Define un identificador único para la herramienta
- Valor: CNModelo

Node Mappings

- Función: Define el tipo de nodo a agregar al entorno
- Valor: NModeloOdoo

A la operación “Begin” de la herramienta se agrega la operación “CreateInstance”, se define el comportamiento usando las siguientes características:

1. Reference Name

- Función: Indica el nombre de la referencia en la cual se va a agregar la instancia
- Valor: requerimiento_tiene_modelos

Type Name

- Función: Indica el tipo de instancia a crear
- Valor: odoo8.ModeloOdoo

Variable Name

- Función: Indica el nombre de la variable cuando se esté creando la instancia, esta variable puede ser usada usando extensiones de código Java
- Valor: instance

La estructura de la herramienta de creación de ModeloOdoo se muestra en la siguiente imagen:

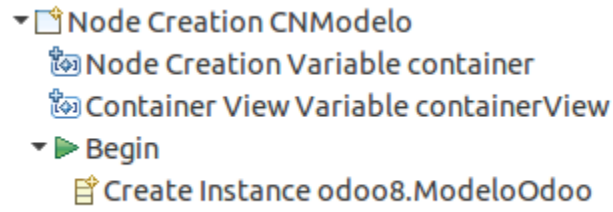


Ilustración 50 estructura de la herramienta de creación de Modelo. Fuente: Elaboración Propia

Creación de herramienta para AtributoOdoo

Se agrega a la sección de herramientas un nuevo elemento de creación del tipo "Node Creation", las configuraciones para la herramienta son las siguientes:

1. ID

- Función: Define un identificador único para la herramienta
- Valor: CNAtributo

2. Node Mappings

- Función: Define el tipo de nodo a agregar al entorno
- Valor: NAtributoOdoo

A la operación "Begin" de la herramienta se agrega la operación "CreateInstance", se define el comportamiento usando las siguientes características:

1. Reference Name

- Función: Indica el nombre de la referencia en la cual se va a agregar la instancia
- Valor: requerimiento_tiene_atributos

2. Type Name

- Función: Indica el tipo de instancia a crear
- Valor: odoo8.AtributoOdoo

3. Variable Name

- Función: Indica el nombre de la variable cuando se esté creando la instancia, esta variable puede ser usada usando extensiones de código Java
- Valor: instance

La estructura de la herramienta de creación de ModeloOdoo se muestra en la siguiente imagen:

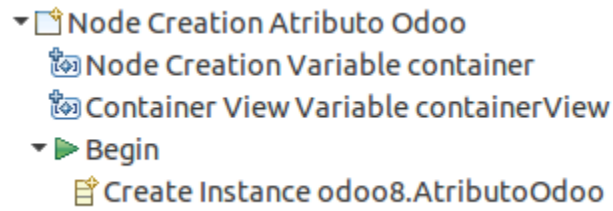


Ilustración 51 Estructura de la herramienta de creación de modelo. Fuente: Elaboración Propia

Creación de herramientas para las relaciones

Con la herramienta Sirius puede llegar a ser más compleja la creación de una relación que la de un nodo ya que para estas se debe especificar tanto el contexto desde el que parte como el contexto en el que termina dicha relación, a continuación se modificara el ya mencionado contexto.

Creación de herramienta para relación OdooModulo – OdooModelo

Se agrega a la sección de herramientas un nuevo elemento de creación del tipo "Edge Creation", las configuraciones para la herramienta son las siguientes:

1. ID

- Función: Define un identificador único para la herramienta
- Valor: CRModuloModelo

Edge Mapping

- Función: Representación de relación a mapear
- Valor: RModuloModelo

A la operación “Begin” de la herramienta se agrega la operación “ChangeContext”, se define el comportamiento usando las siguientes características:

1. Browse Expression

- Función: Consulta que retorna un tipo de instancia para navegar hacia la creación del elemento
- Valor: [source/] (Usando aql)

A la operación “Change Context” se agrega una nueva operación “Set” con la finalidad de cambiar el context a la entidad destino de la relación, se configura con las siguientes características:

1. Feature Name

- Función: nombre de la propiedad
- Valor: modulo_tiene_modelos

Value Expression

- Función: Define una consulta que retorna la entidad destino de la relación
- Valor: [target/] (Usando aql)

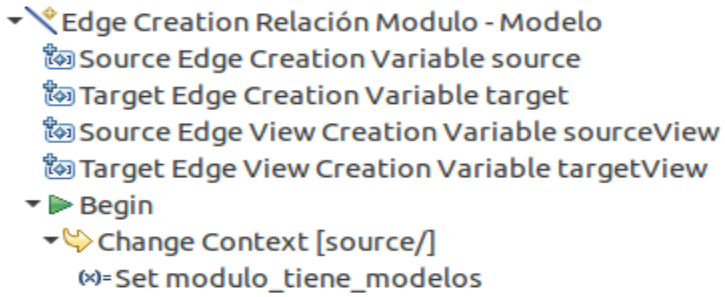


Ilustración 52 Estructura herramienta creación relación Modulo - Modelo. Fuente: Elaboración Propia

Creación de herramienta para relación ModeloOdoo - AtributoOdoo

Se agrega a la sección de herramientas un nuevo elemento de creación del tipo “Edge Creation”, las configuraciones para la herramienta son las siguientes:

1. ID

- Función: Define un identificador único para la herramienta
- Valor: CRModeloAtributo

Edge Mapping

- Función: Representación de relación a mapear
- Valor: RModeloAtributo

A la operación “Begin” de la herramienta se agrega la operación “Change Context”, se define el comportamiento usando las siguientes características:

1. Browse Expression

- Función: Consulta que retorna un tipo de instancia para navegar hacia la creación del elemento
- Valor: [source/] (Usando aql)

A la operación “Change Context” se agrega una nueva operación “Set” con la finalidad de cambiar el context a la entidad destino de la relación, se configura con las siguientes características:

1. Feature Name

- Función: nombre de la propiedad
- Valor: modelo_tiene_atributos

Value Expression

- Función: Define una consulta que retorna la entidad destino de la relación
- Valor: [target/] (Usando aql)

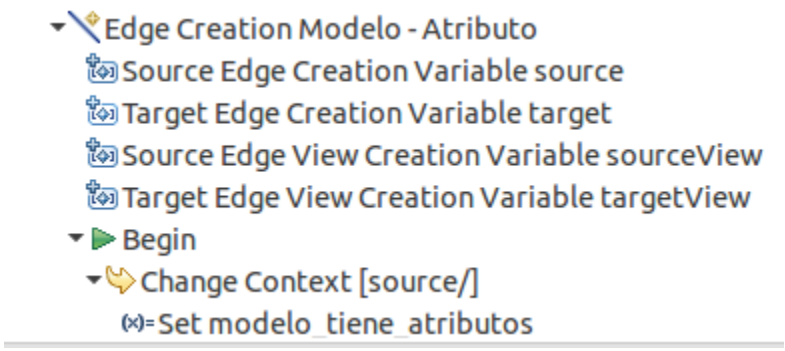


Ilustración 53 Estructura Herramienta creación Modelo - Atributo. Fuente: Elaboración Propia

Creación de la herramienta para relación ModeloOdoo - ModeloOdoo

Se agrega a la sección de herramientas un nuevo elemento de creación del tipo "Edge Creation", las configuraciones para la herramienta son las siguientes.

1. ID

- Función: Define un identificador único para la herramienta
- Valor: CRModeloModelo

2. Edge Mapping

- Función: Representación de relación a mapear
- Valor: RModeloModelo

A la operación "Begin" de la herramienta se agrega la operación "External Java Action", se define el comportamiento usando las siguientes características:

1. ID

- Función: Define un identificador único para la acción Java
- Valor: CrearRelacionOdoo

Action

- Función: Define la acción a llamar en el procedimiento Begin
- Valor: CrearRelacionOdoo

Se agregan los siguientes parámetros al “External Java Action”, estos parámetros serán inyectados a la función creada:

1. Parámetro Origen

- Función: Pasará como parámetro el ModeloOdoo desde el cual parte la relación
- Name: parametro_origen
- Value: [self.source/]

Parámetro Destino

- Función: Pasará como parámetro el ModeloODoo en el cual termina la relación
- Name: parametro_destino
- Value: [self.target/]

Contenedor

- Función: Pasará como parámetro el contenedor de la creación de la relación
- Name: parametro_contenedor
- Value: [self.container /]

Dentro del proyecto se agrega una nueva clase llamada “CrearRelacionOdoo”, esta clase implementa la interfaz “IExternalJavaAction”, esta interfaz debe ser implementada para que el “ExternalJavaAction” sea capaz de ejecutar el siguiente código:

```
@Override
public void execute(Collection<? extends EObject> arg0, Map<String, Object> arg1) {
      ModeloOdoo source = (ModeloOdoo) arg1.get("parametro_origen");
      ModeloOdoo target = (ModeloOdoo) arg1.get("parametro_destino");
      RequerimientoOdoo diagram = (RequerimientoOdoo) arg1.get("parametro_contenedor");

      RelacionOdoo relation = Odoo8Factory.eINSTANCE.createRelacionOdoo();

      relation.setModeloOrigen(source);
      relation.setModeloDestino(target);

      diagram.getRequerimiento_tiene_relaciones().add(relation);

}
```

Ilustración 54 Código creación de relaciones. Fuente: Elaboración Propia

Los Java Paremeters agregados en el Viewpoint, llegan a la clase por medio de un objeto MAP de Java, el cual permite extraerlos usando código, cada línea del código de la ilustración 54 se explica a continuación:

- Lineas 2,3,4: Extraen los parámetros parametro_origen, parametro_destino, parametro_contenedor, y los convierte a objetos para posteriormente realizar operaciones con ellos
- Linea 6: Crea una RelacionOdoo usando el patrón Fábrica
- Lineas 8,9: Cambia los modelos de la relación por los correspondientes origen, destino
- Linea 10: Agrega la RelacionOdoo recién creada al RequerimientoOdoo

Dentro del archivo Plugin.xml del proyecto Odoo8Dsl.design, en la sección de extensiones se agrega la siguiente extensión org.eclipse.sirius.externalJavaAction, se configura la action class con el nombre de la clase recién creada y su paquete "Odoo8Dsl.design.CrearRelacionOdoo", se configura el id como "CrearRelacionOdoo"

Para este punto, la sección de herramientas se debe ver así:

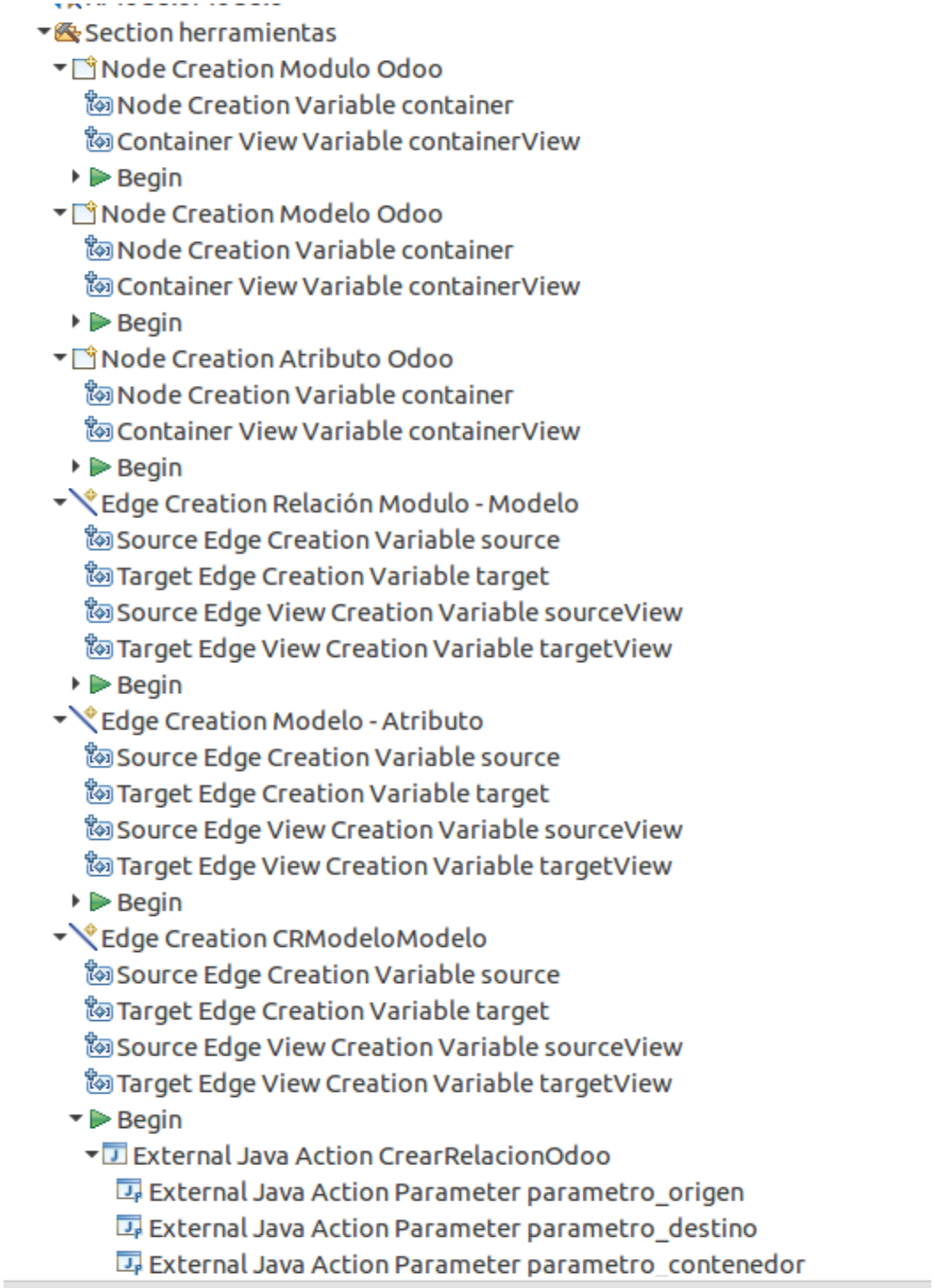


Ilustración 55 Seccion de herramientas del DSL. Fuente: Elaboración Propia

Una vez se ejecuta el proyecto, así debe verse la visualización y la barra de herramientas:

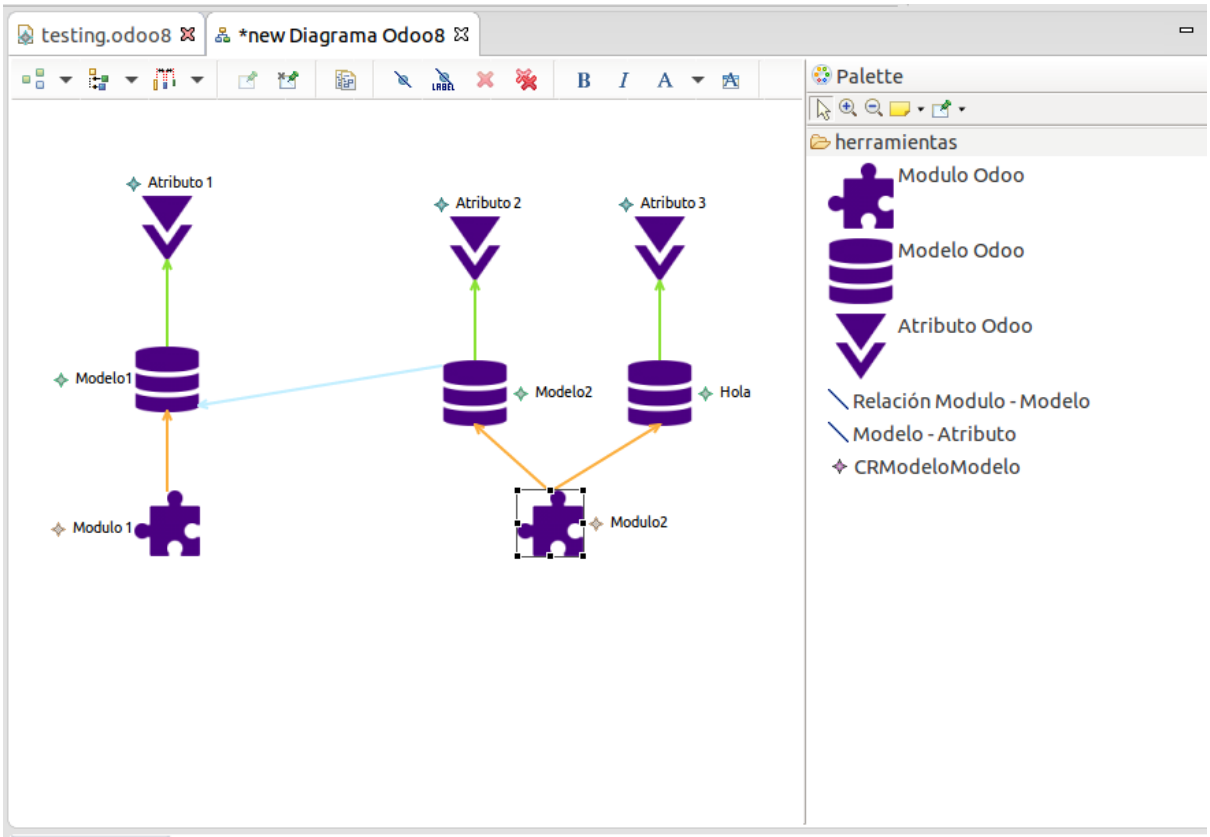


Ilustración 56 Visualización y barra de Herramientas del proyecto. Fuente: Elaboración Propia

Transformación del modelo en módulos para el sistema Odoo

Hasta este punto se tiene el modelado del requerimiento odoo, sin embargo la representación de esto es un archivo XML, este archivo contiene todos los elementos modelados y su descripción, para transformar esto en código utilizable por el sistema Odoo, es necesario hacer un proceso mediante el cual se codifican las descripciones.

El proceso de transformación tiene lugar fuera de las herramientas provistas por Sirius, la razón de esto, es que debido a la complejidad de la estructura de los módulos, es necesario un lenguaje que permita la creación de archivos y no limite las posibilidades de codificación, para este propósito se ha escogido Python.

A continuación se puede observar un flujo de cómo debería funcionar el script a desarrollar en Python.

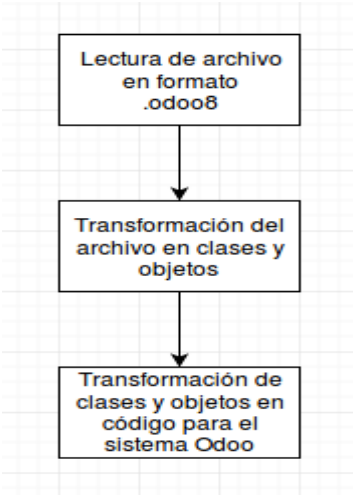


Ilustración 57 flujo de funcionamiento del script. Fuente: Elaboración propia

El script es un archivo que debe ejecutarse por consola con el nombre odooxml2odoo8.py, como se puede ver en la siguiente ilustración:

```
vagrant@vagrant: /vagrant/odoo
→ xmi2openerp git:(master) ✗ python2.7 odooxml2odoo8.py testing.odoo8
[INFO] Starting to parse Odoo8 XML file testing.odoo8
[INFO] Parsing modules, relations and classes ...
[INFO] Parsing classes attributes ...
[INFO] Finished
→ xmi2openerp git:(master) ✗
```

Ilustración 58 Ejecución del Script. Fuente: Elaboración propia

Una vez ejecutado el script la estructura de datos debe ser una carpeta por módulo, la orden usada para mostrar esta estructura es “ls . -R”

```
→ xmi2openerp git:(master) ✗ cd generated_code
→ generated_code git:(master) ✗ ls . -R
.:
clientes  ventas

./clientes:
clientes.py  clientes_view.xml  __init__.py  __openerp__.py

./ventas:
__init__.py  __openerp__.py  ventas.py  ventas_view.xml
→ generated_code git:(master) ✗
```

Ilustración 59 Estructura de datos generada por el script

Una vez finalizado todo el proceso descrito anteriormente el siguiente paso es probar la herramienta y hacer las validaciones correspondientes, esto se verá en el siguiente capítulo.

5 Diseño de un módulo Odoo con OdooDSL

A continuación se hará un diseño de un módulo Odoo con el DSL grafico generado a lo largo de este proyecto, esto con el fin de probar que la herramienta funcione de manera adecuada.

Descripción del requerimiento

Se requiere un sistema en odoo que mediante dos módulos distintos sea capaz de manejar tanto clientes como ventas. El sistema de clientes solo contendrá el manejo de un cliente regular, el cual tiene los siguientes atributos: nombre, identificación y un indicador de si está activo en el sistema o no, el sistema de ventas por su parte como su nombre lo indica debe encargarse de las ventas, estas constan básicamente de los siguientes atributos: monto y descripción, a su vez las ventas deben ser atribuidas a un vendedor que tiene atributos propios como :identificación, nombre y un indicador de si está activo en el sistema o no, finalmente los clientes deben tener en su registro todas las compras que han hecho.

Abstracción de elementos requeridos

Después de hacer la debida abstracción del requerimiento descrito previamente se llegó a la conclusión de que la estructura más apropiada sería la siguiente:

Modulo Clientes

Este módulo posee una entidad llamada cliente regular con los siguientes atributos:

- nombre: Texto

- Identificación: Numérico
- activo: Booleano
- ventas: Relación uno a mucho a venta

Modulo Ventas

Este módulo posee dos entidades, la entidad venta y la entidad vendedor, cada uno con las siguientes características

- Venta
    - valor: Flotante
    - descripción: Texto
- Vendedor
    - nombre: Texto
    - Identificación: Numérico
    - activo: Booleano
    - Ventas: Relación uno a muchos a venta

Modelado dentro de la herramienta OdooDsl

Después del proceso de modelado la abstracción obtenida anteriormente el resultado es el siguiente:

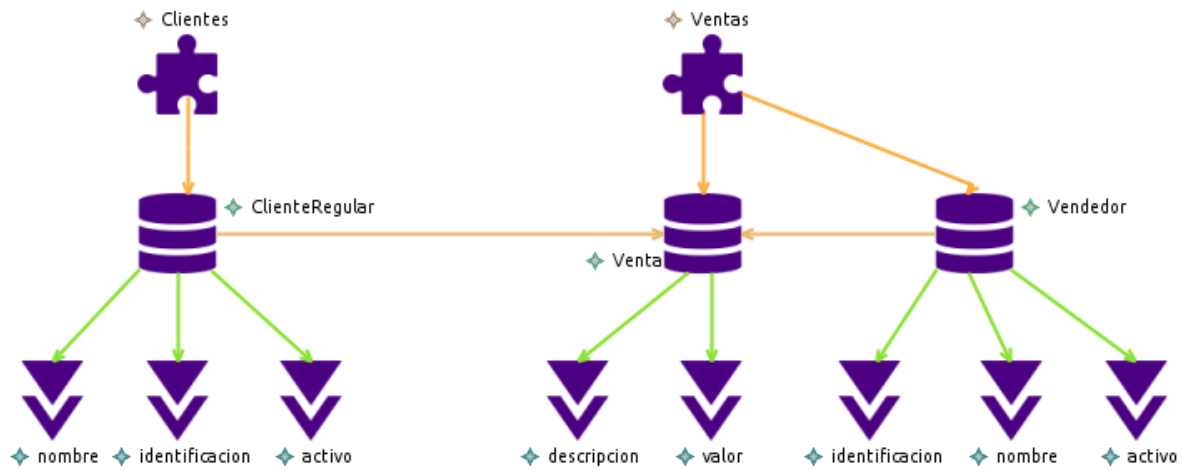


Ilustración 60 Modelo representante del requerimiento. Fuente: Elaboración Propia

Una vez se toma el archivo .odoo8 y se transforma con el script de transformación, se obtiene el siguiente resultado en la estructura

```
→ generated_code ls . -R
.:
clientes  ventas

./clientes:
controllers.py  controllers.py~  __init__.py  models.py  models.py~  __openerp__.py  __openerp__.py~  templates.xml

./ventas:
controllers.py  controllers.py~  __init__.py  models.py  models.py~  __openerp__.py  __openerp__.py~  templates.xml
→ generated_code
```

Ilustración 61 Resultado de la transformación. Fuente: Elaboración Propia

6 Prueba de eficiencia del DSL gráfico

En este capítulo se describe una prueba hecha con varios desarrolladores de odoo que actualmente trabajan en diferentes proyectos, dicha prueba se hace con el fin de probar la eficiencia del DSL gráfico generado en este proyecto.

Metodología de prueba

Se dará al desarrollador el requerimiento que debe programar, una vez el desarrollador lo haya comprendido se dará inicio y se tomará el tiempo de desarrollo del módulo usando OdooDsl, posteriormente se pedirá al desarrollador que desarrolle de nuevo el modulo pero esta vez sin usar la herramienta, a este proceso también se le tomará el tiempo para finalmente comparar los tiempos de ambos procesos, se espera que los tiempos de desarrollo del módulo mejoren con el uso de la herramienta OdooDsl.

Prueba

A continuación se describirán los pormenores del proceso de prueba

Desarrolladores

Para la ejecución de esta prueba se solicitó apoyo a los siguientes desarrolladores:

- José Javier Vargas, desarrollador en el Instituto de desarrollo urbano IDU de Bogotá.
- Cinxgler Mariaca Minda, desarrollador en el Instituto de desarrollo urbano IDU de Bogotá.
- Cristian Alejandro Rojas Quintero, desarrollador Senior Python freelance.
- Steven Sierra Forero, Desarrollador Senior Ruby Globant.

- Christian Chaparro Africano, Desarrollador Junior freelance.

Enunciado del requerimiento

Se requiere un sistema en odoo que mediante dos módulos distintos sea capaz de manejar tanto clientes como ventas. El sistema de clientes solo contendrá el manejo de un cliente regular, el cual tiene los siguientes atributos: nombre, identificación y un indicador de si está activo en el sistema o no, el sistema de ventas por su parte como su nombre lo indica debe encargarse de las ventas, estas constan básicamente de los siguientes atributos: monto y descripción, a su vez las ventas deben ser atribuidas a un vendedor que tiene atributos propios como :identificación, nombre y un indicador de si está activo en el sistema o no, finalmente los clientes deben tener en su registro todas las compras que han hecho.

Tiempos esperados

Con el fin de comprobar que el DSL gráfico funcionaba correctamente se hicieron pruebas que consistían básicamente en desarrollar varios módulos con la herramienta, encontrar posibles bugs y corregirlos.

Gracias a dichas pruebas se pudieron establecer unos tiempos esperados para el desarrollo de un módulo de Odoo con OdooDSL.

Se diseñaron 6 módulos de prueba, 3 fueron diseñados por Andrea Stefania sicachá Arenas y tres por Andrés Hamir Cobos Prada, los tiempos se pueden ver en la siguiente tabla:

Tabla 3

Tiempos de desarrollo de los módulos de prueba con OdooDSL, sacada de: Elaboración propia.

Desarrollador	Modulo	Tiempo
Andrea Sicachá	1	35 minutos
Andrés Cobos	1	30 minutos
Andrea Sicachá	2	40 minutos
Andrés Cobos	2	25 minutos
Andrea Sicachá	3	25 minutos
Andrés Cobos	3	25 minutos

Haciendo un promedio de los tiempos de cada desarrollador se obtuvieron los siguientes resultados:

- Promedio Andrea Sicachá: 33 minutos por módulo.
- Promedio Andrés Cobos: 27 minutos por módulo.

Con base a los promedios mencionados anteriormente se espera un tiempo de desarrollo de 30 minutos por módulo (promedio desarrollo por módulo de ambos desarrolladores.), sin embargo es necesario tener en cuenta que ninguno de los desarrolladores trabaja normalmente con OdooERP por lo cual los tiempos pueden variar de manera drástica.

Resultados

Como se mencionó en la descripción de la prueba, se le solicito a cinco desarrolladores que hicieran la prueba tanto con OdooDSL como de manera manual, y se tomaron los tiempos de ambos procesos, los resultados de cada desarrollador se encuentran a continuación.

José Javier Vargas

Tabla 3

Resultados en tiempo de la prueba aplicada al desarrollador José Javier Vargas sacada de: Elaboración propia.

Método de desarrollo	Tiempo requerido
OdooDsl	10 Minutos
Manual	15 Minutos

Comentarios

Como actividad extra para obtener un poco de retroalimentación de la herramienta se pidió al desarrollador hacer comentarios sobre la misma, dichos comentarios se encuentran a continuación:

- Es complejo el cambio de uso de eclipse a ejecutar un script el cambio, debería ser automático
- Es una herramienta útil para hacer el prototipo inicial de los módulos, ya que los cambios son fáciles de hacer.

Cinxgler Mariaca Minda

Tabla 4

Resultados en tiempo de la prueba aplicada al desarrollador Cinxgler Mariaca Minda sacada de: Elaboración propia.

Método de desarrollo	Tiempo requerido
OdooDsl	15 Minutos
Manual	18 Minutos

Comentarios

Como actividad extra para obtener un poco de retroalimentación de la herramienta se pidió al desarrollador hacer comentarios sobre la misma, dichos comentarios se encuentran a continuación:

- El modulo que se genera presenta unas mínimas inconsistencias, sin embargo son errores que deben revisarse y corregirse.
- Automatizar el proceso puede que tome cada vez menos tiempo si se conoce bien la herramienta

Cristian Alejandro Rojas Quintero

Tabla 5

Resultados en tiempo de la prueba aplicada al desarrollador Cristian Alejandro Rojas Quintero sacada de: Elaboración propia.

Método de desarrollo	Tiempo requerido

OdooDsl	20 Minutos
Manual	15 Minutos

Comentarios

Como actividad extra para obtener un poco de retroalimentación de la herramienta se pidió al desarrollador hacer comentarios sobre la misma, dichos comentarios se encuentran a continuación:

- Aunque nunca había trabajado con Odoo fue más sencillo hacer un módulo sin la herramienta, probablemente porque ya había trabajado con código Python y en cambio de la herramienta no tenía ningún conocimiento previo.

Steven Sierra Forero

Tabla 6

Resultados en tiempo de la prueba aplicada al desarrollador Steven Sierra Forero sacada de: Elaboración propia.

Método de desarrollo	Tiempo requerido
OdooDsl	15 Minutos
Manual	20 Minutos

Comentarios

Como actividad extra para obtener un poco de retroalimentación de la herramienta se pidió al desarrollador hacer comentarios sobre la misma, dichos comentarios se encuentran a continuación:

- Es una herramienta útil y ayuda a mejorar los tiempos de desarrollo.

Cristian Chaparro Africano

Tabla 7

Resultados en tiempo de la prueba aplicada al desarrollador Cristian Chaparro Africano sacada de: Elaboración propia.

Método de desarrollo	Tiempo requerido
OdooDsl	20 Minutos
Manual	25 Minutos

Comentarios

Como actividad extra para obtener un poco de retroalimentación de la herramienta se pidió al desarrollador hacer comentarios sobre la misma, dichos comentarios se encuentran a continuación:

- Gracias a que se tiene una interfaz gráfica el proceso es más rápido ya que si hubiera que codificar para generar el modulo daría lo mismo hacerlo de manera manual.

Análisis Estadístico

Con base a los resultados presentados en el sub numeral anterior (Prueba->Resultados) fue posible realizar un análisis estadístico que se presentará a continuación:

Datos

Tabla 8

Resumen tiempos de desarrollo con y sin la herramienta OdooDSL

Desarrollador	Tiempo con OdooDSL(min)	Tiempo manual(min)
José Javier Vargas	10	15

Cinxgler Mariaca Minda	15	18
Cristian Alejandro Rojas	20	15
Steven Sierra Forero	15	20
Cristian Chaparro	20	25

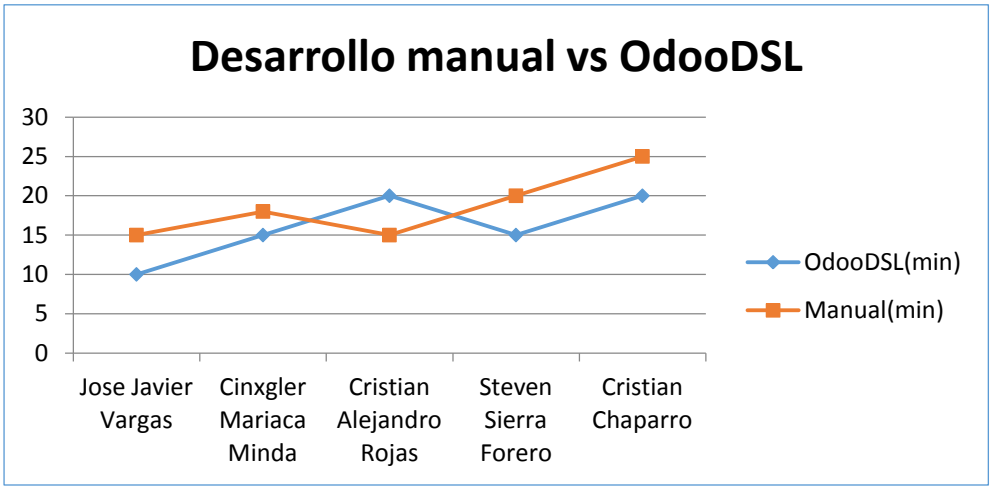


Ilustración 62 gráfica desarrollo manual vs OdooDSL

Análisis

Promedio de tiempo desarrollo manual: 18.6 Minutos

Promedio de tiempo desarrollo con OdooDSL: 16 Minutos

Porcentaje de optimización: Para obtener el porcentaje de optimización es necesario tomar el tiempo de desarrollo manual como el 100% ya que este es el tiempo promedio esperado sin aplicar ninguna optimización (en este caso el uso de la herramienta OdooDSL). Partiendo de este punto el siguiente paso consiste en calcular por medio de una regla de 3 a qué porcentaje del tiempo de desarrollo manual equivale el tiempo de desarrollo con la herramienta OdooDSL y finalmente al 100% se debe restar el resultado de la regla de 3, este proceso se puede ver a continuación:

18,6 min (tiempo de desarrollo manual) → 100%

16 min (tiempo de desarrollo con OdooDSL) → ?

Ilustración 63 representación de la regla de 3 para calcular el porcentaje de tiempo de desarrollo con OdooDSL

$$16\ min\ x\ \frac{100\ \%}{18.6\ min} =\ \ 86.021\ \%$$

Ilustración 64 Regla de 3

Porcentaje de optimización = 100% - 86.021%

Porcentaje de optimización = 13.9784

El porcentaje de optimización como se acabó de ver es de 13.978 %, lo que quiere decir que gracias a la herramienta OdooDSL los tiempos en el desarrollo de módulos Odoo se pueden reducir aproximadamente en un 14%.

Conclusiones y Trabajo Futuro

- La implementación del DSL gráfico propuesto en este proyecto funciona adecuadamente en la etapa de modelamiento y desarrollo inicial de un módulo Odoo, sin embargo no es posible utilizar el DSL en etapas más avanzadas debido a que la edición de módulos ya generados no es soportada por OdooDsl.

- El tiempo que toma la fase inicial de desarrollo de un módulo Odoo se optimizó aproximadamente un 14% usando OdooDsl con respecto al desarrollo manual de los módulos (se puede ver en el capítulo 6 en el numeral Análisis estadístico) gracias a la implementación de la herramienta OdooDSL desarrollada en este proyecto.

- La Ingeniería Dirigida por Modelos da una solución al problema de la creación de un DSL gráfico como OdooDSL(DSL gráfico desarrollado en este proyecto) ya que el proceso de abstracción que esta describe para cada una de sus capas facilita y organiza el desarrollo de la herramienta, además de que permite dividir el proceso en fases.

- El metamodelo diseñado con el meta-metamodelo ECORE para este proyecto cumple con las condiciones necesarias para la construcción de un DSL gráfico en un editor gráfico como Sirius o GMF, esto se da debido a que el metamodelo contiene características parametrizables tanto en nodos como aristas. Esto se puede evidenciar en el capítulo 4 creación del DSL gráfico en el que se observa que se uso el metamodelo creado en Ecore.

- El DSL gráfico creado a lo largo de este proyecto cumple cabalmente con la función esperada, es decir la construcción de un modulo en la herramienta ERP Odooo en su etapa inicial, esto se puede evidenciar en el capítulo 6 Pruebas de Eficiencia del DSL gráfico, en el cual desarrolladores ajenos al proyecto crearon un modulo utilizando la herramienta OdooDSL.

- Como trabajo futuro se puede hablar de la creación de un DSL gráfico para otros sistemas ERP teniendo en cuento que estos deben ser a su vez modulares y que sus módulos deben contener relaciones y modelos, o de lo contrario sería necesario realizar cambios un poco extremos en el metamodelo propuesto, sin embargo una característica que hace a un sistema ERP es que este sea modular por tanto ya hay una generalidad que podría ser explotada.

Bibliografia

[1] Siswanto, Joko and Anggi Maulida. "ERP Module Requirements For Micro, Small And Medium Enterprise Fashion Industry In Bandung". International Conference on Information Technology Systems and Innovation (ICITSI) (2014): 6. Print.

[2] Lenn **Bass, Paul Clements, Rick Kazman. 2012.** *software architecture in practice.* Westford, Massachusetts : Addison-Wesley, 2012.

[3] **Reis, Daniel. 2015.** *Odoo Development Essentials.* Birmingham : Packt Publishing Ltd, 2015.

[4] Ågren, Herman. "Using Formal Verification To Eliminate Software Errors". (2007): 6. Print.

[5] Chang, LIU, ZHENG Jun, and REN Zhanyong. "Software Architecture Of Integrated Modular Avionics Software Testing Environment". (2009): 5. Print.

[6] Gaelic D, Djuric D, Devedzic , "Model Driven Engineering and Ontology Development." s.l. : Springer, 2006.

[7] D, Schmit, "Model Driven Engineering". s.l. : IEEE Computer Society, 2006.

[8]Object management Group. "MDA specifications"; Available: http://www.omg.org/mda/specs.htm

[9] D, Frankel. " Model driven Architecture, applying MDA to enterprise computing". Indianapolis : Wiley publishing, Inc.

[10] Deursen Arie, Klint Paul. “Domain-Specific Language Design Requires Feature Descriptions”, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, 2002

[11] Montenegro Carlos, Cueva Juan Manuel, SanJuan Oscar, Gaona Paulo. “Desarrollo de un lenguaje de dominio específico para sistemas de gestión de aprendizaje y su herramienta de implementación “KiwiDSM” mediante ingeniería dirigida por modelos”. España. Universidad de Oviedo. 2011

[12] Alvarez rosío, Urla Jacqueline, “Tell me a good story: using narrative analysis to examine information requirements interviews during an ERP implementation”. University of Massachusetts. 2002

[13] Chiesa, Florencia. “METODOLOGÍA PARA SELECCIÓN DE SISTEMAS ERP.” Buenos Aires -Argentina : s.n., 2004.

[14] Rodriguez Andrés, Pineda José, Sanchez Ricardo. “sistemas de planificación de recursos empresariales: un caso real” México D.F, México, 2002 .

[15] Fabien Pinckaers, Geoff Gardiner, Els Van Vossel. “The OpenERP Solution” Odoo, 2012.

[16] Pinckaers, F. and Gardiner, G. (2009). *Open ERP for retail and industrial management*. Grand-Rosière: Open Object Press.

[17] Eclipse ORG, “Eclipse Modeling Framework”; available:

https://eclipse.org/modeling/emf/

[18]Eclipse ORG, “Package org.eclipse.emf.ecore”; Available:

Http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html#details

[19] Object Managment Group. (2015). *OMG Meta Object Facility(MOF) core specification*

. Object Managment Group.

[20] Object Managment Group. (2015). *OMG UNIFIED MODELING LANGUAGE (UML) specification* . Object Managment Group.

[21] Pascual Queralt, L. H. (2008). N MOTOR DE TRANSFORMACIÓN DE MODELOS CON SOPORTE PARA EL LENGUAJE QVT RELATIONS, Universidad Politecnica de Valencia

[22] Odoo S.A; "Odoo Documentation"; Available: https://www.odoo.com/documentation/8.0/

[23] Dimitrios S. Kolovos, L. M. (Mayo de 2009). Raising the Level of Abstraction in the Development of GMF-based Graphical Model Editors, Vancouver, Canada.

[24] Eclipse ORG, " Sirius Overview"; Available: https://eclipse.org/sirius/overview.html

[25] Vladimir Vujović*, M. M. (Julio de 2014). Sirius: A Rapid Development of DSM Graphical Editor . Tihany, Hungary.