Universidad de las Ciencias Informáticas

Facultad 3

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Título: Desarrollo de una arquitectura para los módulos de configuración básica de Cedrux.

Autor: Andrés Asprón Hernández

Tutores:

Ing. Eliober Cleger Despaigne

Ing. Erich Mario Gómez Pérez

Junio, 2016

Frase:

"La Arquitectura no está basada en el hormigón y el acero y los elementos de la tierra. Está basada en el asombro"

Daniel Libeskind

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Andrés Asprón Hernández

_____

Firma del Autor

Ing. Eliober Cleger Despaigne

_____

Firma del Tutor

Ing. Erich Mario Gómez Pérez

_____

Firma del Tutor

Agradecimientos

Agradezco a:

En primer lugar a mi madre porque ha dedicado su vida para que yo me formara como persona y por darme su apoyo incondicional en todo momento, por su sacrificio en todos mis años de estudio desde la primaria hasta aquí, por su entrega y por el amor que me ha brindado siempre.

A mi abuela Amparo, aunque ya no esté conmigo, por ser aquella persona tan especial en mi vida que me daba más fuerzas para seguir adelante todos los días.

A mi abuelo Miguel que ha sido el padre que yo siempre desee, por su forma, su carácter, la fuerza con la que enfrentaba los problemas, por llenarme de consejos que el solo sabe darme.

A mi tutor Eliober por todo su apoyo y por tener paciencia conmigo, gracias por todo, sin tu ayuda no hubiese sido posible todo esto.

A mi pequeña hermana Lianet, que tantas veces me decía que estudiara para que saliera bien.

A toda mi familia porque es lo mejor que tengo, porque siempre me han apoyado en todo momento.

A mi hermano de toda la vida Maikel, por ser una persona que siempre estuvo, en las buenas y en las difíciles, sin importar lo que pudiera suceder.

A mi mejor amigo en la UCI, Disnel.

A mis compañeros de las grandes batallas, Yotsan, Carlos, Pedro, Keylier, Leansi, Manolo, Yoel, Daniel.

A mi queridos compañeros de aula.

A mis tíos Luis y Odelkis que tanto me han ayudado a lo largo de todos estos años.

A todas las personas que he conocido en estos cinco años y que siempre han estado ahí en todo momento.

Agradezco a todo el que de una forma u otra me ha ayudado a salir adelante.

Gracias a todos…

Dedicatoria

A la memoria de mi abuela

Y a la mejor de las madres.

RESUMEN

Con el desarrollo de la ciencia y la tecnología en el campo de la informática, con el fin de agilizar la gestión de los procesos administrativos de las empresas a nivel mundial, han surgido diferentes aplicaciones para controlar de manera eficientes los recursos de una empresa. No obstante, a pesar de la gran variedad de estos a nivel nacional e internacional, no se ha logrado obtener un sistema estándar para beneficiar la gestión contable en las empresas cubanas.

Con el objetivo de lograr una eficiente planificación de los recursos económicos, financieros y mejorar los resultados económicos del país, se crea el producto Xedro-ERP, un ERP (Enterprise Resource Planner) desarrollado en la Universidad de las Ciencias Informáticas (UCI) en el 2008. Este producto está integrado por un conjunto de módulos que responden a los diferentes procesos de negocios que se desarrollan en una empresa. El presente trabajo tiene como objetivo proponer una nueva arquitectura para el proyecto Cedrux, contribuyendo así a mejorar la escalabilidad y mantenibilidad del mismo.

Durante la investigación se realizan valoraciones y estudios sobre diferentes arquitecturas, sus posibles funcionalidades y las ventajas que proporcionan, definiéndose Django como la nueva arquitectura para el producto. Esta propuesta contribuirá a mejorar la gestión de los recursos de una organización.

Palabras claves: Cedrux, Django, mantenibilidad, escalabilidad, framework.

ÍNDICE

INTRODUCCIÓN

Cuba realiza profundas transformaciones en la esfera económica con el fin de sentar las bases para su futuro desarrollo productivo y social. Un paso de avance ha sido la puesta en marcha de un proceso de informatización, que prioriza los sectores más críticos en el desarrollo económico del país. La Universidad de las Ciencias Informáticas (UCI) ha jugado un papel destacado en esta tarea dirigida por el Ministerio de las Comunicaciones (MINCOM).

Debido a la necesidad que tiene el país de desarrollar un sistema de gestión integral para las entidades cubanas y sus particularidades, la UCI en el 2008 comenzó el desarrollo del “ERP cubano: Cedrux”. El proyecto involucró a más de 380 integrantes (200 estudiantes, 120 profesores y 60 especialistas funcionales), un número inédito en el país (Lazo Ochoa, 2007). Esta iniciativa contribuyó al desarrollo profesional de sus integrantes, sin embargo no constituyó un caso de éxito en la producción de software.

A finales del 2014, la dirección del centro CEIGE y los integrantes del proyecto Cedrux (17 en aquel entonces), realizaron un encuentro con el propósito de determinar las acciones que posibilitaran la rectificación de errores y continuidad de la iniciativa. Se tomó la decisión de realizar un estudio que reflejara la situación del proyecto y del producto hasta ese momento. Entre los resultados arrojados por el informe resultante (CEIGE, 2015) se enuncian a continuación un conjunto de deficiencias identificadas en las tecnologías utilizadas en el desarrollo del producto Xedro-ERP:

1. Desactualización de la base tecnológica (ExtJs, PHP, Zend, Doctrine, PostgreSQL).
2. Desactualización de los componentes desarrollados en la biblioteca UCID. En esta biblioteca se implementaron componentes de ExtJs con comportamiento y atributos personalizados para satisfacer necesidades propias del negocio del proyecto Cedrux y para facilitar la implementación de la capa de presentación.
3. La versión de PHP utilizada presenta problemas con la carga en memoria de objetos muy grandes. Por los procesos de negocio que maneja Cedrux es frecuente la carga de volúmenes de datos considerables.
4. Las validaciones, excepciones y los servicios están registrados de forma global por lo que se hace muy difícil trabajar con esos ficheros de configuración.
5. El mecanismo para el registro de las trazas afecta en gran medida el rendimiento de Cedrux.

6. El mecanismo de comunicación entre los procesos de configuración básica realizado a través de los módulos Configuración, Nomencladores y Multimoneda es deficiente y muy rígido, afectando.
7. La definición de los componentes no contó en todos los casos con el visto bueno de un especialista lo que provocó que en ocasiones se abusara de este aspecto y que aumentara el nivel de dependencias internas. Este elemento afecta directamente la modularidad de Cedrux.
8. Malas prácticas de programación, por ejemplo: código duplicado, sin el uso de estándares e insuficientemente documentado. Todo esto conlleva a la obtención de un código difícil de mantener e ilegible en ocasiones.
9. Diseño de la BD realizado de forma incorrecta. Tablas sin usar, datos inconsistentes, implementaciones correspondientes a los procesos de negocio en la base de datos.

En la actualidad el proyecto está compuesto por 10 integrantes (6 especialistas y 4 recién graduados en adiestramiento). Se tienen bien documentados los procesos de importantes áreas de una empresa como: la contabilidad, las finanzas y el capital humano, así como la referente a la estructura básica del sistema en función de los procesos de administración y configuración. Ha recibido varias certificaciones entre las que se pueden enunciar:

- La certificación realizada por Conavana S.A. de los procesos generales agrupados en los subsistemas Configuración, Estructura y Composición, Multimoneda, así como los procesos contables que posee el módulo de Contabilidad.
- La certificación realizada por Interaudit S.A. de los procesos del subsistema Capital Humano y de los procesos de finanzas, agrupados en los subsistemas Caja, Banco y Cobro y Pagos.

Desde enero del 2015 se encuentra implantado Xedro-ERP en la Empresa de Calzado COMBELL. Durante el despliegue se ha podido corroborar que las dificultades anteriormente mencionadas afectan el funcionamiento satisfactorio del producto. A partir de la necesidad de mantener los resultados obtenidos hasta la fecha y lograr un producto de calidad, que se ajuste a los cambios en el modelo económico cubano y basado en el principio de soberanía tecnológica la presente investigación plantea como **problema a resolver**: ¿Cómo mejorar la mantenibilidad y la escalabilidad en los módulos de configuración básica del producto Cedrux?

La presente investigación tiene como **objeto de estudio** la Arquitectura de software y como **campo de acción** los procesos de configuración básica en Cedrux.

Se plantea como **objetivo general** desarrollar una arquitectura para los módulos de configuración básica de Cedrux de manera que se mejore la mantenibilidad y escalabilidad de los mismos.

Para dar cumplimiento al objetivo propuesto se desglosan los siguientes **objetivos específicos:**

1. Elaborar un marco teórico relacionado con la selección de la arquitectura sobre la cual se implementará la solución.
2. Desarrollar los elementos de la arquitectura que permitan la ejecución de los módulos de configuración básica (Nomencladores, Configuración, y Multimoneda).
3. Validar la solución propuesta mediante la aplicación de métricas, pruebas de software y el estudio de casos.

Con el objetivo de orientar la investigación se precisa la siguiente **idea a defender** el desarrollo de una nueva arquitectura para los módulos de configuración básica de Cedrux contribuirá a la mejora de la mantenibilidad y escalabilidad de los mismos.

Como **posibles resultados** se espera la obtención de una base tecnológica para Cedrux que contribuya a eliminar los problemas actuales del sistema, y mantenga los logros alcanzados hasta la fecha.

Para cumplir satisfactoriamente las tareas investigativas se utilizaron los siguientes métodos de investigación:

Métodos Teóricos

Historio-Lógico: Permitió tener un mejor entendimiento de los antecedentes y tendencias actuales relacionadas con los arquitecturas más significativas para los sistemas informáticos y a partir de este un análisis realizado, proponer una solución coherente para el proyecto Cedrux.

Analítico-Sintético: Para realizar un análisis de la información empleada para la investigación, así como las especificaciones de los diferentes procesos centrándolos en la problemática, así como realizar una valoración de las diferentes características de los procesos y poder obtener la solución centrada en los objetivos de la investigación. Por último sintetizar en el informe los resultados obtenidos durante la investigación.

Métodos Empíricos

Entrevista en profundidad: Permitió corroborar la incidencia de cada uno de los elementos planteados en la problemática en el funcionamiento del producto Xedro-ERP. Para ello fueron necesarios varios encuentros con arquitectos y desarrolladores del proyecto Cedrux, que contribuyeron a identificar los elementos arquitectónicos de mayor incidencia en el desarrollo de la solución.

Capítulo 1: Fundamentación teórica, se abordan los elementos referentes a la arquitectura de software, así como los principales conceptos relacionados a la misma y se definen las herramientas y tecnologías empleadas para el desarrollo de la solución.

Capítulo 2: Análisis e implementación, se analizan las posibles alternativas para dar respuesta a los problemas de Cedrux, realizando un estudio de cada una de ellas. Por último se implementa la solución propuesta, a partir de la arquitectura seleccionada.

Capítulo 3: Validación y pruebas, se realiza un análisis a través de la aplicación de métricas, pruebas y la realización de un estudio de casos, con el objetivo de evidenciar la mejora en la mantenibilidad y escalabilidad a partir de la propuesta.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

En el presente capítulo se realiza un análisis de las tecnologías existentes teniendo en cuenta la mantenibilidad y la escalabilidad del nuevo framework para Cedrux. Luego se define la metodología de desarrollo de software a utilizar y finalmente se fundamenta el uso de las herramientas y tecnologías necesarias para la implementación de los módulos de configuración básica.

1.1 Conceptos esenciales

A continuación, se enuncian los conceptos utilizados durante el desarrollo de la investigación, dichos conceptos están relacionados con el objeto de estudio y campo de acción en la que la misma se desarrolla.

1.1.1 ERP (Enterprise Resource Planner)

Se denominan ERP a los sistemas de planificación de recursos empresariales que integran y manejan muchas de las prácticas de los negocios asociados con las operaciones de producción y los aspectos de distribución de una compañía comprometida en la producción de bienes o servicios, son parte del conjunto de sistemas de información gerencial que permiten tener un control de la empresa por sus directivos en tiempo (González Brito, 2007).

1.1.2 Calidad

Según la Real Academia de la Lengua Española (RAE), la calidad es:

“Propiedad o conjunto de propiedades inherentes a algo, que permiten juzgar su valor.”

En la metodología ágil Scrum se define la calidad como “la capacidad del producto o los productos entregables completados para cumplir con los criterios de aceptación y alcanzar el valor de negocio que espera el cliente”.

Según el Dr. Eduardo Arturo Rodríguez Tello, el software debe proveer la funcionalidad y desempeño requeridos por el usuario y debe ser mantenible, confiable y aceptable (Rodriguez Tello, 2012).

La norma ISO9126-1 presenta dos modelos de calidad, la primera referente a la calidad interna y externa y la segunda a la calidad de uso. La presente investigación se centrará en la primera de estas, principalmente en la calidad interna.

Dicha norma define seis categorías para la evaluar la calidad de software, las cuales son: funcionabilidad, confiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad.

1.1.3 Escalabilidad

La escalabilidad es la capacidad de mejorar recursos para ofrecer una mejora (idealmente) lineal en la capacidad de sus servicios, la manera de adaptarse y dar respuestas con respecto al rendimiento del sistema, a medida que aumentan de forma significativa el número de usuarios del mismo (IEEE, 2012).

1.1.4 Mantenimiento de software

Conjunto de actividades destinadas a proporcionar soporte económicamente rentable para un determinado producto software. Estas actividades se realizan tanto antes de la entrega del producto como después de la entrega del mismo. Las actividades previas a la entrega incluyen las actividades destinadas a planificar, anticipar y preparar actividades de mantenimiento posteriores. Las actividades posteriores a la entrega incluyen modificaciones del producto software, formación y asistencia al usuario (ISO/IEC/IEEE, 2006).

Pressman [1998] dice que “la fase mantenimiento se centra en el cambio que va asociado a la corrección de errores, a las adaptaciones requeridas a medida que evoluciona el entorno del software, y a cambios debidos a las mejoras producidas por los requisitos cambiantes del cliente”.

1.1.5 Mantenibilidad

El IEEE[1] define mantenibilidad como: “La facilidad con la que un sistema o componente software puede ser modificado para corregir fallos, mejorar su funcionamiento u otros atributos o adaptarse a cambios en el entorno” (IEEE, 2005).

[1] Institute of Electrical and Electronics Engineers. (1990) IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY.IEEE Std. 610.12 (1990) Standard Glossary of Software Engineering Terminology. IEEE Computer Society Press, Los Alamitos, CA.

1.1.6 Framework de desarrollo web

El término framework es empleado en diferentes ámbitos del desarrollo de software, no solo en el de aplicaciones Web. Según (Gutierrez, 2014) se puede considerar como una estructura de software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. Los objetivos principales que persigue un framework son: acelerar el proceso de desarrollo y reutilizar código ya existente. Un framework Web por tanto, según (Gutierrez, 2014) también, puede ser definido como un conjunto de componentes que conforman un diseño reutilizable que facilita y agiliza el desarrollo de sistemas Web. Supondría entonces una ventaja en cuanto a tiempo y buenas prácticas de desarrollo como son el uso de patrones arquitectónicos y de diseño, el empleo de un framework Web para la implementación de la solución de la presente investigación.

1.2 Posibles alternativas arquitectónicas

Durante la investigación, se tomó como referencia el Informe de estudio realizado al producto Xedro-ERP (CEIGE, 2015). En dicho documento se exponen diferentes alternativas para la obtención de la base tecnológica de Cedrux. A continuación, se expresan las conclusiones a las que se llegaron tras analizarlas.

1.2.1 Sauxe

Sauxe surge debido a la necesidad que tenía el país de desarrollar un sistema para la gestión integral de las entidades cubanas y sus particularidades. Es un framework que contiene un conjunto de componentes reutilizables, provee la estructura genérica para el desarrollo de aplicaciones web de gestión, logrando una mayor estandarización, flexibilidad, integración y agilidad en el proceso de desarrollo, siguiendo el paradigma de independencia tecnológica por el cual apuesta el país (Gómez Baryolo, 2010).

Está escrito en el lenguaje PHP, que recoge las mejores prácticas aplicadas en los framework estudiados, como es el caso de Synfony, Zend Framework, Code Igniter, Kumbia, entre otros. Cuenta con una arquitectura en capas que básicamente está compuesto por 4 capas o niveles.

Basado en un estudio que se realizó al centro (CEIGE, 2015), se identificaron varios factores subjetivos que influyen negativamente en el proceso de desarrollo de Sauxe, tales como la poca gestión de documentación relacionada con la arquitectura y funcionamiento, implementación con malas prácticas

de programación, poco control y seguimiento a la implementación de las soluciones, no se implementa un sistema de auditorías al código fuente o al cumplimiento de las definiciones arquitectónicas, mala organización del proceso de desarrollo. El equipo de proyecto no recibió suficiente capacitación sobre el uso de las tecnologías. Al no existir definiciones centrales, cada equipo ha resuelto los problemas de forma aislada, provocando la existencia de soluciones duplicadas.

La evolución y desarrollo de Sauxe quedó en un segundo plano. Para continuar utilizando Sauxe con las tecnologías actualizadas se debía contar con una experiencia de desarrollo en Zend Framework 2, la cual el equipo aún no posee. Estos elementos han dado lugar a que en la actualidad la base tecnológica de Cedrux sea calificada de obsoleta y a que su actualización sea sumamente compleja. Lo anterior descarta la posibilidad de continuar utilizando dicho framework.

1.2.2 Odoo 8.0

Odoo es un sistema ERP, producido por la empresa belga Odoo S.A, la cual declara que dicho producto es una alternativa de código abierto a SAP ERP y Microsoft Dynamics. Posee una arquitectura cliente-servidor, con módulos implementados en Python y XML para las vistas. El cliente se comunica con el servidor a través de interfaces XML–RPC. Utiliza PostgreSQL como gestor de base de datos.

Dada su variada gama de módulos ya implementados y su amplia comunidad de desarrollo, Odoo representa una atractiva variante para reutilizar dichas funcionalidades y comenzar un desarrollo sobre esa base, lo cual posibilitaría la consolidación de la idea de tener una solución ERP desarrollada parcialmente por la UCI y orientada a las necesidades de la empresa actual cubana.

A pesar de las facilidades que brinda este producto, es importante aclarar que dentro de la licencia bajo la cual está liberado, se expresa que se puede realizar módulos que utilicen Odoo como biblioteca (típicamente en función de ella, importando y haciendo uso de sus recursos), pero sin modificar código fuente o material del software. En cuanto a la arquitectura, es notable aclarar que todo desarrollo puesto en marcha sobre este producto, no sería mantenido por la compañía, por lo que, si ocurriesen cambios arquitectónicos en las próximas versiones de Odoo, podrían existir incompatibilidades de los módulos desarrollados, si se deseara migrar a versiones más actuales del sistema. Es decir, la licencia de Odoo expresa que no se debe cambiar su código, por lo que la UCI no tendría control arquitectónico sobre este, lo que obligaría en cierta medida a estar utilizando una única versión (CEIGE, 2015). Para resolver

este problema, se pueden desarrollar en componentes que utilicen los módulos del núcleo de Odoo, por tanto una actualización del sistema, no los afectaría.

1.2.3 Django 1.9

Django es un framework de desarrollo web escrito en Python, que permite la creación de aplicaciones web más rápido y con menos código. Posee énfasis en la reutilización, la conectividad y extensibilidad de componentes, el desarrollo rápido y el principio No te repitas (DRY, *Don't Repeat Yourself)* (Roldan Estebanez, 2015). Este utiliza la arquitectura MVC, donde el código para definir y acceder a los datos (modelo) está separado de la lógica de negocio (el controlador) que está separado de la interface de usuario (vista).

Pone énfasis en el re-uso de la conectividad y extensibilidad de componentes. Fue liberado bajo la licencia BSD[2] en el 2005

La arquitectura de un proyecto web utilizando Django ofrece una estructura flexible y bien organizada, la cual se basa en una filosofía que promueve el acoplamiento débil y la estricta separación entre los módulos de una aplicación web. Ello contribuye al aumento de la capacidad de escalabilidad, facilitando la implementación de sus complementos sin afectar el funcionamiento de los restantes módulos, así como la capacidad de manipular prácticas recomendables para el diseño de aplicaciones web, evidenciadas, en este caso, en el uso particular de patrones pertenecientes a la familia GRASP como son Alta Cohesión y Bajo Acoplamiento.

En este punto es válido aclarar dos conceptos básicos que deben conocerse al desarrollar aplicaciones con Django: proyecto y aplicación. Un proyecto es una instancia de un cierto conjunto de aplicaciones de Django, más las configuraciones de estas aplicaciones; mientras que una aplicación es un conjunto portable de una funcionalidad de Django, típicamente incluye modelos y vistas, que conviven en un solo paquete Python, estas son reusables en múltiples proyectos.

[2] BSD: Berkeley Software Distribution. Así se llamó a las distribuciones de código fuente desarrolladas en la Universidad de Berkeley en California y que en origen eran extensiones del sistema operativo UNIX® de AT&T Research. Varios proyectos de sistemas operativos de código abierto tienen su origen en una distribución de éste código conocida como 4.4BSD-Lite. Añaden además un buen número de paquetes de otros proyectos de Código Abierto, incluyendo de forma destacada al proyecto GNU.

Al estar desarrollado utilizando Python, aprovecha las bondades del lenguaje, las cuales se describen a continuación: es un lenguaje de alto nivel, conveniente para un desarrollo rápido de código, su filosofía de diseño enfatiza la simplicidad y legibilidad del código, posee un núcleo de lenguaje relativamente pequeño con el apoyo de magnificas bibliotecas, entre las que se encuentran: NumPy, SciPy, pandas, matplotlib etc. Es un lenguaje multiparadigma, al brindar la posibilidad de compilar varios estilos de programación (imperativo, orientado a objetos, funcional) y multiplataforma, gratuito y open source; posibilitando que el código fuente del producto sea fácil de mantener por otros desarrolladores (CAChemE, 2013).

Django es mejorado casi diariamente, los desarrolladores del framework tienen un alto grado de interés en asegurarse de que les ahorre tiempo a los desarrolladores, produzca aplicaciones que son fáciles de mantener y rindan bajo mucha carga.

1.2.4 Conclusiones del estudio realizado

Siguiendo el principio de elevar la soberanía tecnológica, expresado en el 6to Congreso del Partido Comunista de Cuba (VI Congreso del Partido Comunista de Cuba, 2011) y teniendo en cuenta las características analizadas, durante la presente investigación se hará uso del framework Django como base tecnológica para la implementación de la solución. Con la utilización del mismo es posible la realización de aplicaciones web con un alto grado de escalabilidad, evidenciadas en la utilización de patrones arquitectónicos, como es el caso de Bajo Acoplamiento y Alta cohesión, pues existe una estricta separación en los módulos de las aplicaciones web, posibilitando que sea posible la modificación de sus módulos sin que afecte el funcionamiento del resto de los módulos. Además de un alto grado de mantenibilidad que aportan las bibliotecas que posee, enfatizados en la simplicidad y legibilidad del código, posibilitando al desarrollador un producto fácil de mantener.

1.3 Metodología para el desarrollo de software

Actualmente el desarrollo de software dentro de la actividad productiva de la Universidad de las Ciencias Informáticas se caracteriza por el uso de diferentes metodologías de desarrollo entre robustas y ágiles, entre las que se encuentran: XP, RUP, SXP, Open UP, entre otras. La definición del uso de una metodología depende de las características del proyecto y el producto. En la presente investigación se requiere el desarrollo de tres módulos de configuración básica sobre la arquitectura que plantea del framework Django para la validación de la propuesta. Para ello se cuenta con un equipo de desarrollo e

investigación compuesto por el dúo tutor-estudiante, así como un período corto para el cumplimiento de los objetivos. Es necesario destacar además la necesidad de obtener resultados de implementación a corto plazo. Por último existe información suficiente sobre el framework y los módulos de configuración básica, por lo que no es necesaria la generación exhaustiva de documentos. Todo ello fue considerado para la selección de la metodología Variación AUP-UCI.

Esta Metodología se apoya en el Modelo CMMI-DEV v1.3, el cual constituye una guía para aplicar las mejores prácticas en una entidad desarrolladora. Estas prácticas se centran en el desarrollo de productos y servicios de calidad (Sánchez, 2015).

1.3.1 Descripción de las fases

De las 4 fases que propone AUP (Inicio, Elaboración, Construcción, Transición), Variación AUP-UCI mantiene la fase de Inicio, pero modificando el objetivo de la misma, y unifica las restantes 3 fases de AUP en una sola, a la que nombra Ejecución y agrega la fase de Cierre, como se muestra a continuación para una mejor comprensión (Sánchez, 2015):

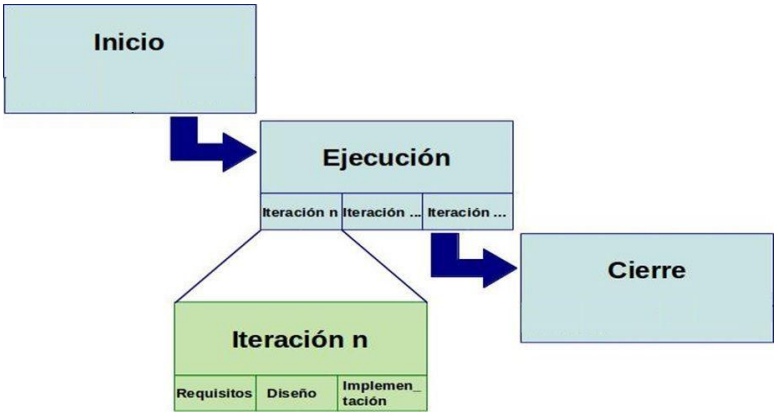


Figura 1: Metodología Variación AUP-UCI.

Fuente: Metodología Variación AUP-UCI

A continuación, se describe cada una de las fases de la metodología:

Inicio: Durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.

Ejecución:

En esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto, considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto.

Cierre:

En esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

A partir de que el Modelado de negocio propone tres variantes a utilizar en los proyectos (CUN, DPN o MC) y existen tres formas de encapsular los requisitos (CUS, HU, DRP), surgen cuatro escenarios para modelar el sistema en los proyectos, de los cuales, se escogió el cuarto debido a que el cliente estará siempre acompañando al equipo de desarrollo para convenir los detalles de los requisitos y así poder implementarlos, probarlos y validarlos. Se recomienda en proyectos no muy extensos, ya que una HU no debe poseer demasiada información (Sánchez, 2015).

Escenario No 4: Proyectos que no modelen negocio solo pueden modelar el sistema con HU.

Figura 2: Escenario No 4. Historias de Usuarios.
Fuente: Metodología Variación AUP-UCI

1.4 Tecnologías y herramientas a utilizar

Una vez definida la metodología que guiará el desarrollo de los módulos de prueba en el acápite anterior, se deben definir las tecnologías afines a Django necesarias para el desarrollo de aplicaciones web. De igual forma debe fundamentarse el uso de herramientas de apoyo al desarrollo de software.

1.4.1 PostgreSQL 9.4

PostgreSQL es un potente sistema de base de datos objeto-relacional de código abierto. Cuenta con más de 15 años de desarrollo activo y una arquitectura probada que se ha ganado una sólida reputación de fiabilidad e integridad de datos.

Esta versión agrega muchas nuevas características que mejoran la mantenibilidad de manera que provee una nueva API para leer, filtrar y manipular el flujo de replicación de PostgreSQL, además de proporcionar una mejora considerable en cuanto a la escalabilidad del mismo, así como el rendimiento de PostgreSQL para diferentes tipos de usuarios de bases de datos, incluyendo mejoras al soporte para archivos JSON, replicación y rendimiento de los índices (Acosta, 2016).

1.4.2 Redis 3.0

Redis es un motor de base de datos en memoria, basado en el almacenamiento en tablas de hashes (clave/valor) pero que opcionalmente puede ser usada como una base de datos durable o persistente. En el contexto de la investigación se utilizará para almacenar solo datos en memoria (BD Redis, 1012).

1.4.3 Módulos de terceros

Además de Django se hará uso de otros módulos desarrollados por terceros, los cuales se describe a continuación.

Django CORS Headers

Aplicación de Django que añade cabeceras CORS (de origen cruzado de intercambio de recursos) a las respuestas. Aunque JSON-P es útil, se limita estrictamente a las peticiones GET. Las cabeceras CORS forman parte de una petición XmlHttpRequest y permiten a los desarrolladores hacer peticiones entre dominios, similar a las solicitudes del mismo dominio (Python, 2016).

Django Rest Framework (DRF)

Una aplicación de Django que facilita la creación de aplicaciones Restful con Django. A partir de la definición de los recursos (serializers), las vistas (viewsets) y las rutas (urls) es posible brindar recursos mediante una API (Python, 2016).

Auth Token

Un conjunto de bibliotecas que utiliza DRF para la autenticación por token (Python, 2016).

Djoser

Una aplicación de Django que provee una API para la gestión básica de usuario (Python, 2016).

Django Websockets Redis 0.4.4

Este módulo implementa la interfaz Websockets en el framework Django sin requerir ningún marco adicional. Para la mensajería que utiliza el almacén de datos Redis (Python, 2016).

1.4.4 Lenguaje Unificado de Modelado (UML) 2.0

Es un lenguaje para la visualización, especificación, construcción, y documentación de los artefactos que se crean durante el proceso de desarrollo.

UML es un lenguaje que proporciona un vocabulario y unas reglas para permitir una comunicación, en este caso este lenguaje se centra en la representación gráfica de un sistema. Permite representar a mayor o menor medida todas las fases de un proyecto informático: desde el análisis de los casos de usos, el diseño con los diagramas de clases, la implementación con los diagramas de componentes, hasta la configuración con los diagramas de despliegue, entre otros.

UML brinda el lenguaje gráfico para modelar, por lo que se necesita de una herramienta que como mínimo facilite la creación de diagramas y la relación entre los componentes de estos (Pedrosa, y otros, 2011).

1.4.5 Visual Paradigm 8.0

Visual Paradigm para UML es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software. Cuenta con un editor de detalles de casos de uso. Permite representar todos los tipos de diagramas de clases, generar código y documentación desde diagramas. Soporta notación UML 2.x, tiene capacidad de ingeniería inversa y directa, así como importar y exportar ficheros XMI. Proporciona abundante información sobre UML. Es fácil de instalar y actualizar, es compatible entre ediciones. Además, admite otras herramientas y plug-in de modelado UML para varias plataformas. Facilita licencias especiales para fines académicos. Entre sus características más significativas se puede resaltar que, posee varios idiomas, sus ediciones son compatibles y es de fácil uso para la creación de aplicaciones web (Lerman, 2003).

1.4.6 Entorno de desarrollo

Un ambiente de desarrollo integrado, en inglés Integrated Development Environment (IDE), es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software.

Es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un depurador de interfaz gráfica (Lau, 2011).

PyCharm 4.5

Es un IDE o entorno de desarrollo integrado multiplataforma utilizado para desarrollar en el lenguaje de programación Python. Proporciona análisis de código, depuración gráfica, integración y soporte para el desarrollo web con Django, entre otras bondades. PyCharm es desarrollado por la empresa JetBrains y debido a la naturaleza de sus licencias tiene dos versiones, la Community, que es gratuita y orientada a la educación y al desarrollo puro en Python y la Professional, aportando a la implementación de las soluciones un alto grado de mantenibilidad en el código, así como la facilidad de producir aplicaciones que sean fácil de escalar, pues permite una separación por componentes, de la lógica del sistema, sin que exista una relación fuerte entre ellos (Python, 2016).

1.4.7 Lenguaje de programación

Un Lenguaje de Programación es un conjunto de reglas, notaciones, símbolos y/o caracteres que permiten a un programador poder expresar el procesamiento de datos y sus estructuras en la computadora. Cada lenguaje posee sus propias sintaxis. También se puede decir que un programa es un conjunto de órdenes o instrucciones que resuelven un problema específico basado en un Lenguaje de Programación (Rodriguez Tello, 2012).

Un lenguaje de programación es un idioma artificial diseñado para que las computadoras puedan interpretar las órdenes dadas por el hombre. Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana. Está formado por un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones.

Python 2.7

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma. Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License, que es compatible con la Licencia pública general de GNU a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores (Python, 2016).

1.5 Conclusiones del capítulo

En el presente capítulo se realizó un estudio de las diferentes opciones para desarrollar una arquitectura capaz de resolver los problemas de mantenibilidad y escalabilidad evidenciados a lo largo de toda la investigación, en los módulos de configuración básica de Cedrux. Se determinaron además, las herramientas y tecnologías que se utilizarán en el desarrollo de la solución en correspondencia con la arquitectura propuesta, además se seleccionó la metodología AUP-UCI como guía para el desarrollo de los módulos de prueba. A partir de lo anterior se derivan las siguientes conclusiones:

- La utilización Django como alternativa arquitectónica contribuye a la realización de aplicaciones web con un alto grado de escalabilidad, evidenciadas en la utilización de patrones arquitectónicos para la separación de sus módulos; así como un alto grado de mantenibilidad a partir de las bibliotecas que posee para proveer simplicidad y legibilidad al código.
- Es necesario el uso de módulos de terceros para lograr una solución robusta, haciendo énfasis en el uso de la biblioteca Django Rest Framework (DRF).
- El uso de PyCharm 4.5 aporta a la implementación de las soluciones un alto grado de mantenibilidad en el código, así como la facilidad de producir aplicaciones que sean fácil de escalar, pues permite una separación por componentes, de la lógica del sistema, sin que exista una relación fuerte entre ellos.

CAPÍTULO 2: Desarrollo de la Arquitectura

Introducción

En el presente capítulo se enuncian las características principales de la aplicación. Se plantea una nueva arquitectura para el proyecto Xedro-ERP y se explica la dinámica del proyecto a través de las historias de usuario y los artefactos generados durante el desarrollo del mismo.

2.1 Requisitos funcionales

Entre las técnicas para la captura de estos requisitos utilizadas se encuentra la entrevista en profundidad, aplicada a los clientes para los cuales va a ser desarrollada la aplicación, también se desarrollaron tormentas de ideas con el fin de posibilitar debates y talleres donde el cliente explicó de forma concreta sus necesidades, definiéndose así los siguientes requisitos:

RF 1. Gestionar bancos.
RF 2. Gestionar sucursales.
RF 3. Gestionar unidad de medida.
RF 4. Gestionar conversión entre unidad de medida.
RF 5. Gestionar especialidad.
RF 6. Gestionar país a la entidad.
RF 7. Gestionar párrafo fiscal.
RF 8. Gestionar cliente o proveedor.
RF 9. Gestionar formato.
RF 10. Gestionar nivel de formato.
RF 11. Gestionar cliente o proveedor a la entidad.
RF 12. Gestionar documento primario.
RF 13. Gestionar operación.
RF 14. Gestionar regla contable.
RF 15. Gestionar moneda.
RF 16. Gestionar moneda contable y financiera.
RF 17. Gestionar tasa.
RF 18. Gestionar moneda por entidad.
RF 19. Gestionar desglose de moneda.

RF 20. Gestionar revaluación.
RF 21. Gestionar escala salarial.
RF 22. Gestionar agrupación lógica.
RF 23. Gestionar calificador de cargo.
RF 24. Gestionar categoría ocupacional.
RF 25. Gestionar cargo civil.
RF 26. Gestionar medio.
RF 27. Gestionar órgano.
RF 28. Gestionar grupo de complejidad.
RF 29. Gestionar nivel de utilización de cargos.
RF 30. Gestionar nivel jerárquico.
RF 31. Gestionar responsabilidad.
RF 32. Gestionar subcategoría.
RF 33. Gestionar tipo de cifra.
RF 34. Gestionar tipo de calificador.
RF 35. Gestionar tipo de escala salarial.
RF 36. Gestionar tipo de plantilla.

2.2 Requisitos no funcionales

Los requisitos no funcionales son los encargados de definir las restricciones adicionales con que debe contar el sistema, indicando prohibiciones, limitaciones y propiedades emergentes. Constituyen rasgos fundamentales que hacen que el sistema sea confiable, rápido, usable y agradable para los usuarios (IEEE, 2005).

RNF 1. Software:

Se requiere un servidor con las características de software siguientes:

- SO: Ubuntu Server >= 14.04.
- PostgreSQL >= 9.4
- Django >= 1.9
- Los módulos de terceros ya especificados en la sección 1.4.3 del capítulo 1.

RNF 2. Hardware:

- Para el despliegue de la arquitectura se requiere una estación con microprocesador doble núcleo Intel Dual Core o Core i3 a 2.0GHz, 2 o 4 GB de RAM y una capacidad de 100 GB en disco duro.

RNF 3. Escalabilidad:

- El sistema debe ser construido sobre la base de un desarrollo evolutivo e incremental, de manera tal que nuevas funcionalidades y requerimientos relacionados puedan ser incorporados afectando el código existente de la menor manera posible; para ello deben incorporarse aspectos de reutilización de componentes.
- El sistema debe estar en capacidad de permitir en el futuro el desarrollo de nuevas funcionalidades, modificar o eliminar funcionalidades después de su construcción y puesta en marcha inicial.

RNF 4. Mantenibilidad:

- El sistema debe contar con una interfaz de administración que incluya: Administración de usuarios y Administración de módulos. En cada una de éstas secciones deberá ofrecer todas las opciones de administración disponibles para cada uno.
- El sistema debe estar en capacidad de permitir en el futuro su fácil mantenimiento con respecto a los posibles errores que se puedan presentar durante la operación del sistema.

2.3 Historias de usuarios

Con el objetivo de definir cada una de las actividades que describen las HU se le asignan a estas una o más tareas de ingeniería. Dichas tareas permiten conocer el tiempo que dura cada una y quien debe desarrollarla, pudiéndose realizar entonces una correcta estimación del tiempo de implementación.

A continuación, se muestra la siguiente HU referente al requisito Adicionar Banco, que se encuentra en el módulo configuración.

Tabla 1: HU_1 Adicionar banco.

Historia de Usuario	
Número:HU_1	**Nombre Historia de Usuario:** Adicionar Banco
Modificación de Historia de Usuario Número: ninguna	
Usuario del sistema: Andrés Asprón Hernández	**Iteración Asignada:** 2

Prioridad en Negocio: Muy Alta	**Tiempo Estimado:** 1 hora
Riesgo en Desarrollo: Alto	**Tiempo Real:** minutos
Descripción: La presente historia de usuario tiene como objetivo adicionar un banco en el sistema.	
Observaciones: El usuario debe estar autentificado en el sistema y poseer los permisos necesarios para acceder al mismo. Esta HU hace referencia a la agrupación de requisitos RF1.	

2.4 Arquitectura de software

2.4.1 Modelo – Plantilla –Vista (MTV)

Django es conocido como un Framework MTV, del inglés “Model-Template-View” (Modelo Plantilla Vista), que no es más que una variación del patrón Modelo-Vista-Controlador, e intenta ser lo más funcional posible. En comparación con el patrón MVC, el modelo en Django sigue siendo modelo; a la vista se le denomina plantilla y en el caso del controlador se le denomina vista (Nieto Muñoz, 2015).

En el patrón de arquitectura MTV:

- **M** significa "**Model**" (Modelo), la capa de acceso a la base de datos. Esta capa contiene toda la información sobre los datos: cómo acceder a estos, cómo validarlos, cuál es el comportamiento que tiene, y las relaciones entre los datos.
  El modelo define los datos almacenados, se encuentra en forma de clase de Python, cada tipo de dato que debe ser almacenado se encuentra en una variable con ciertos parámetros; también posee métodos. Todo esto nos permite indicar y controlar el comportamiento de los datos.
- **T** significa "**Template**" (Plantilla), la capa de presentación. Esta capa contiene las decisiones relacionadas a la presentación: como algunas cosas son mostradas sobre una página web u otro tipo de documento.
  La plantilla es básicamente una página HTML con algunas etiquetas extras propias de Django, en sí no solo crea contenido en HTML sino también XML, CSS, JavaScript. La plantilla recibe los datos de la vista y luego los organiza para la presentación al navegador web.
- **V** significa "**View**" (Vista), la capa de la lógica de negocios. Esta capa contiene la lógica que accede al modelo y la delega a la plantilla apropiada: se puede pensar en esto como un puente

entre el modelo y las plantillas.

La vista se presenta en modo de funciones en Python y su propósito es determinar qué datos serán visualizados. El ORM de Django permite escribir código Python en lugar de SQL para realizar las consultas que necesita la vista. La autentificación de servicios y la validación de datos a través de formularios son también tareas de las que se encarga la vista (Infante Montero, 2012).

Figura 3: Arquitectura Modelo Vista Plantilla.

Fuente: (Infante Montero, 2012)

A continuación se muestra como se evidencia la arquitectura MVT en la nueva arquitectura propuesta para los módulos de configuración básica de Cedrux.

La url: **/Home/Configuración/Banco/Add_banco** *(que se comporta como el navegador web), realiza una petición a la view (vista) "***create***", la cual es renderizada por el template (plantilla) "***create_base.html***" y este le envía a la url como respuesta de la petición realizada, a su vez hace uso del model (modelo) "***Banco***" que se comunica con la tabla "***Banco***" de la base de datos para almacenar el acción deseada.*

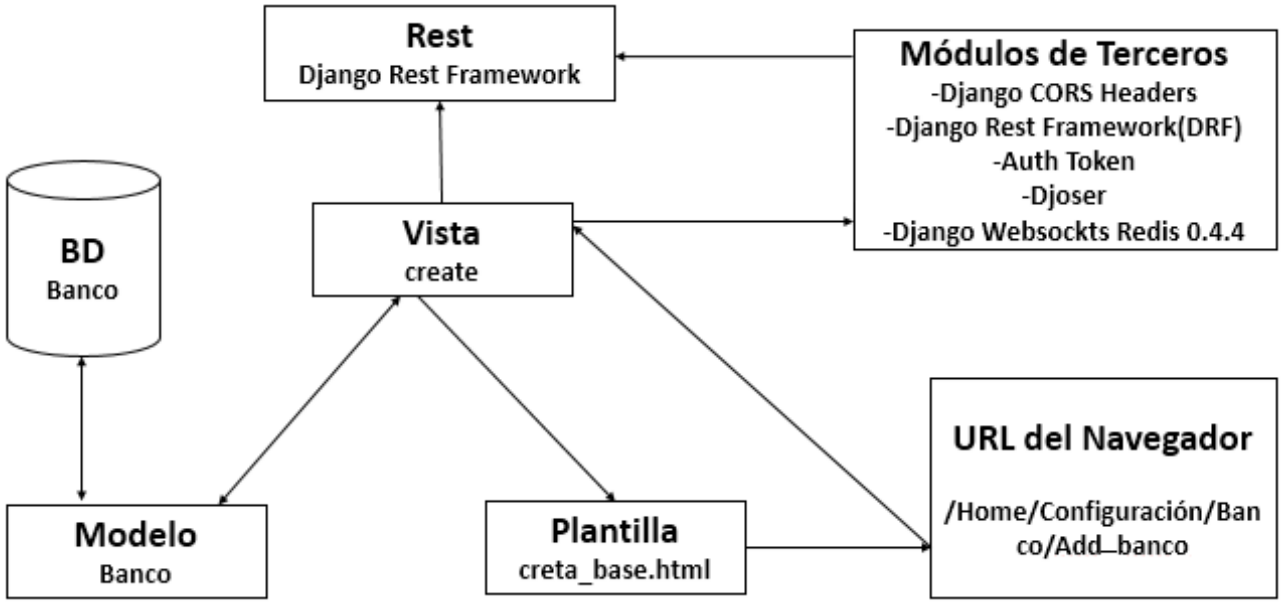


Figura 4: Arquitectura Modelo-Plantilla-Vista (MTV) evidenciado en la funcionalidad Adicionar Banco del módulo Configuración. Fuente: Elaboración propia.

2.4.2 REST

La Transferencia de Estado Representacional (Representational State Transfer) o REST es una arquitectura y no un protocolo, se basa en acciones sobre recursos, para permitir la interacción de un cliente con un servidor. REST define un set de principios arquitectónicos por los cuales se diseñan servicios web haciendo foco en los recursos del sistema, incluyendo cómo se accede al estado de dichos recursos y cómo se transfieren por HTTP hacia clientes escritos en diversos lenguajes (Dos Ideas, 2008).

A continuación se mencionan los principios que sigue la arquitectura:

- Utiliza los métodos HTTP de manera explícita.
- No mantiene estado.
- Expone URIs con forma de directorios.
- Transfiere XML, JavaScript Object Notation (JSON), o ambos.

Los servicios REST surgieron como una alternativa para diseñar servicios web con menos dependencia que su contraparte SOAP. De algún modo, REST es la vuelta a la Web antes de la aparición de los

grandes servidores de aplicaciones, ya que hace énfasis en los primeros estándares de Internet, URI y HTTP. Resulta muy flexible el poder exponer los recursos del sistema con un API REST, de manera que brinde datos a distintas aplicaciones, formateados en distintas maneras. *REST*[3] ayuda a cumplir con los requerimientos de integración que son críticos para construir sistemas en donde los datos tienen que poder combinarse fácilmente y extenderse. Desde este punto de vista, los servicios REST se convierten en algo mucho más grande (Dos Ideas, 2012).

2.5 Caso de estudio

Basada en las tendencias actuales y con el objetivo de lograr que Cedrux sea un producto de mayor escalabilidad y mantenibilidad, se decide proponer una nueva arquitectura tecnológica para el mismo, por una que ofrezca dichas características. Por ello, se utilizó como objeto de estudio los módulos de configuración básica de Cedrux. A continuación se describen cada uno de estos módulos que ofrecerán un mayor entendimiento del sistema.

- Configuración.

El subsistema de Configuración brinda una serie de opciones que deben ser ejecutadas previamente antes de comenzar a utilizar el resto de los subsistemas. Las funcionalidades que en él se encuentran son: Configurar parámetros, Documentos, Reglas contables, Operaciones, Nomencladores generales y otros nomencladores.

- Multimoneda.

El módulo comprende los procesos relacionados con la gestión y configuración de la moneda base y monedas alternativas que se utilizaran en las operaciones contables registradas en el sistema en cada proceso.

- Nomencladores.

En el siguiente subsistema se describe la relación que existe entre los diferentes nomencladores, tales como: País, Escala Salarial, Calificador Cargo, Agrupación Lógica, Cargo Civil, Medio, Órgano, entre otros, así como las funcionalidades que se emplean en el mismo.

[3] REST: Representational State Transfer.

2.6 Patrones

Un patrón es un par de problema/solución con nombre que se puede aplicar en nuevos contextos, con consejos sobre cómo aplicarlos en nuevas situaciones, o sea, un patrón es una descripción de un problema bien conocido que suele incluir: descripción, escenario de uso, solución concreta, consecuencias de utilizar el patrón, ejemplos de implementación y lista de patrones relacionados (Lerman, 2003).

Existen diversos tipos de patrones, entre ellos se encuentran los de arquitectura y de diseño. A continuación, se explica brevemente en que consiste cada uno de ellos y muestra varios ejemplos de los mismos.

2.6.1 Patrones de Diseño

Son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. Los patrones de diseño no son fáciles de entender, pero una vez entendido su funcionamiento, los diseños serán mucho más flexibles, modulares y reutilizables (Gracia, 2016).

2.6.2 Patrones GRASP

Los Patrones de Software para la Asignación General de Responsabilidad (GRASP), representan los principios básicos de la asignación de responsabilidad esos objetos, expresados en forma de patrones. GRASP es el acrónimo para General Responsibility Assignment Software Patterns (Patrones Generales de Software para Asignar Responsabilidades).

GRASP destaca 5 patrones principales: Bajo acoplamiento, Experto, Alta cohesión, Creador y Controlador (Grosso, 2011). A continuación, se explica brevemente en qué consiste cada uno de ellos y como se evidencian en la implementación de la nueva arquitectura desarrollada:

Bajo Acoplamiento

Este patrón garantiza que las clases estén lo menos ligadas posible entre sí, de tal forma que, en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de las clases, potenciando la reutilización y disminuyendo la dependencia entre las clases (Grosso, 2011). Mediante el uso de este patrón se proporciona un bajo acoplamiento en el diseño debido a que las

clases existentes tienen asignadas responsabilidades de tal forma que estas no dependan en gran medida de otras, permitiendo de esta forma tener un sistema más robusto y de fácil mantenimiento.

Alta Cohesión

Cada elemento del diseño debe realizar una labor única dentro del sistema, no desempeñada por el resto de los elementos y auto-identificable.

Ejemplos de una baja cohesión son clases que hacen demasiadas funciones. En todas las metodologías se considera la refactorización. Uno de los elementos a refactorizar son las clases saturadas de métodos. Ejemplos de buen diseño se producen cuando se crean los denominados “paquetes de servicio” o clases agrupadas por funcionalidades que son fácilmente reutilizables (bien por uso directo o por herencia) (Lerman, 2003).

A continuación, se muestra un ejemplo de diagrama de clases donde se evidencia el uso de los patrones de Alta Cohesión y Bajo Acoplamiento:

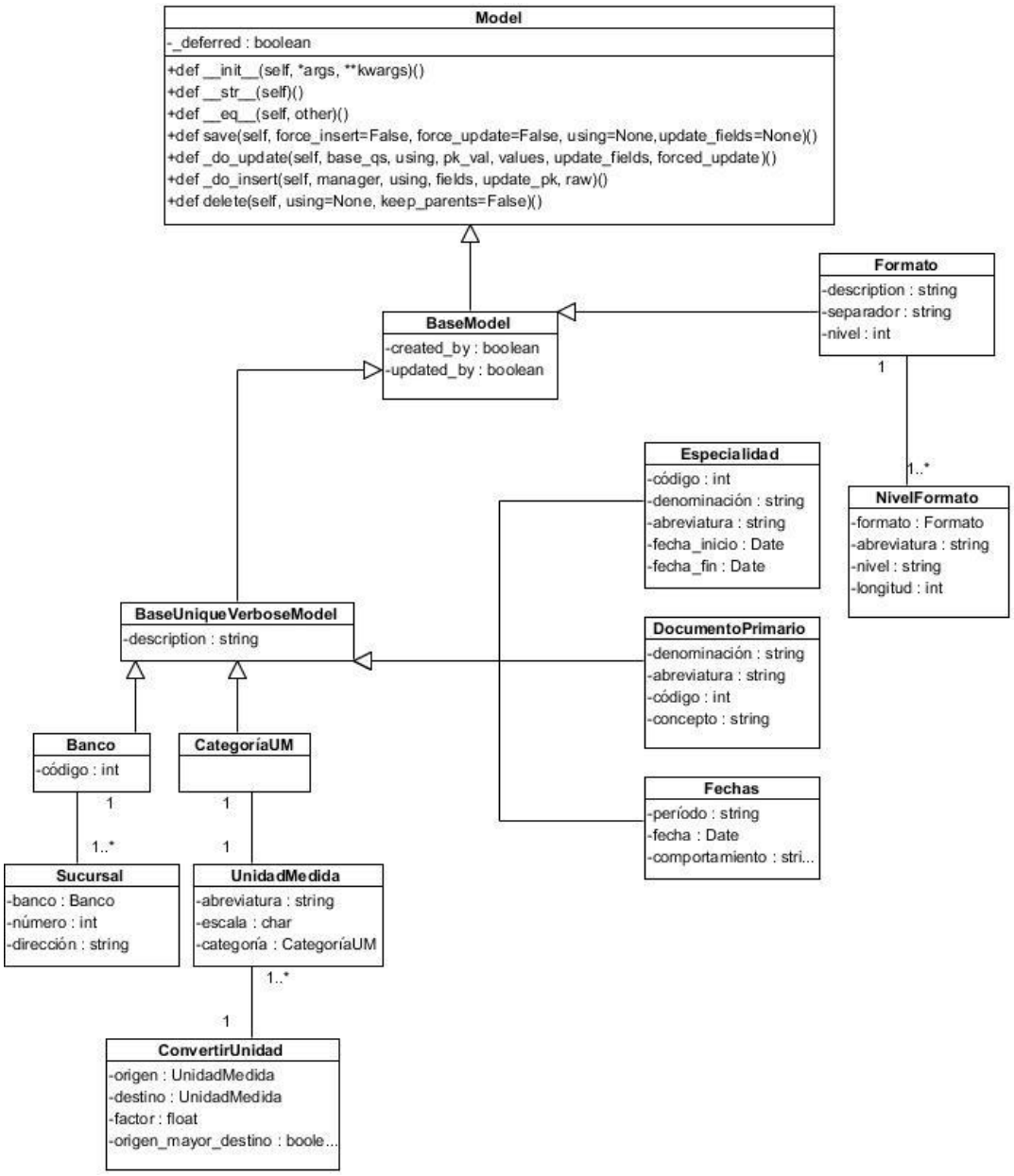


Figura 5: Patrón Bajo Acoplamiento y Alta Cohesión empleado en la solución, modelado a través de un diagrama de clases. Fuente: Elaboración propia.

Este patrón se utiliza debido a que a cada una de las clases se le asignaron responsabilidades de tal forma que estén estrechamente relacionadas entre sí y no realicen un trabajo excesivo.

Experto

La responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos). Una clase, contiene toda la información necesaria para realizar la labor que tiene encomendada. Hay que tener en cuenta que esto es aplicable mientras se estén considerando los mismos aspectos del sistema:

- ✓ Lógica de negocio.
- ✓ Persistencia a la base de datos.
- ✓ Interfaz de usuario.

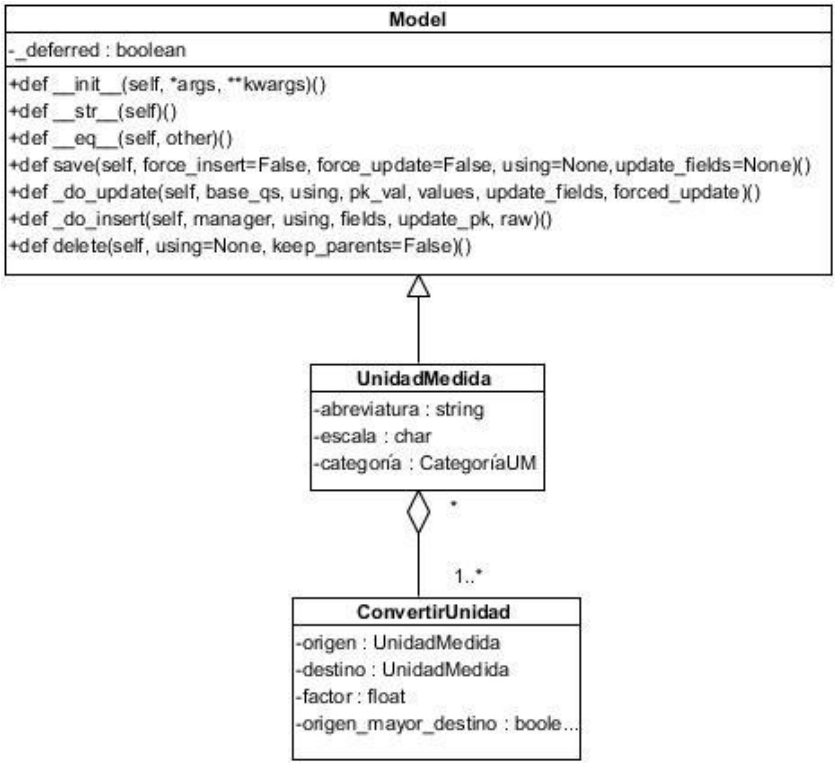


Figura 6: Patrón Experto empleado en la solución modelado a través de un diagrama UML. Fuente: Elaboración propia.

Este patrón es aplicado en todas las clases debido a que cada una de ellas es experta en brindar información que solo ella contiene; facilitando así el entendimiento, extensión y mantenimiento del sistema. A continuación, se muestra un diagrama UML donde se evidencia su uso a través de la clase CategoriaUM que interactúa con las clases UnidadMedida y ConvertirUnidad.

Creador

Se asigna la responsabilidad de que una clase B cree un objeto de la clase A solamente si:

- ✓ B contiene a A.
- ✓ B es una agregación (o composición) de A.
- ✓ B almacena A.
- ✓ B tiene los datos de inicialización de A (datos que requiere su constructor).
- ✓ B usa a A.

La creación de instancias es una de las actividades más comunes en un sistema orientado a objetos. En consecuencia, es útil contar con un principio general para la asignación de las responsabilidades de creación. Si se asignan bien el diseño puede soportar un bajo acoplamiento, mayor claridad, encapsulamiento y reutilización (Lerman, 2003).

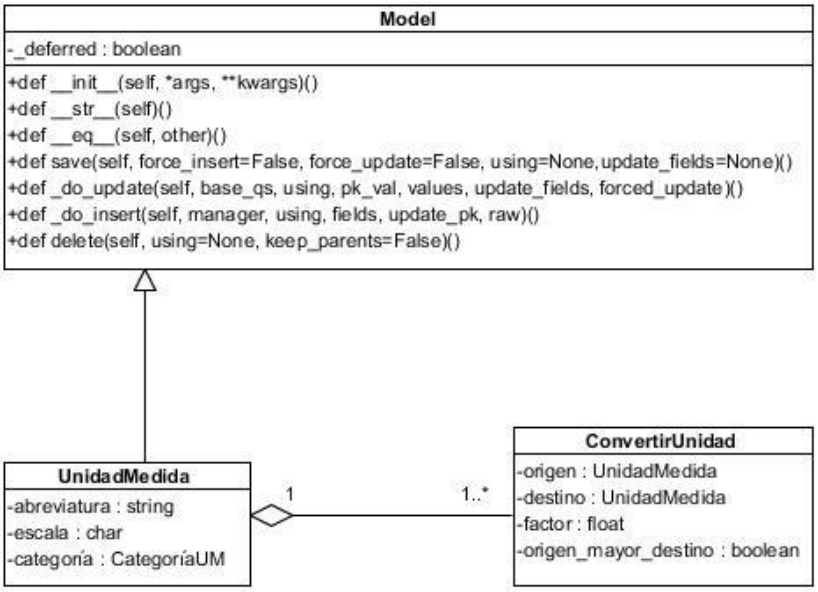


Figura 7: Patrón Experto empleado en la solución modelado a través de un diagrama UML. Fuente: Elaboración propia.

El patrón se evidencia en las clases controladoras que, para cada uno de los módulos de configuración básica, son las encargadas de crear las instancias de los objetos que manejan, favoreciendo así la reutilización y el bajo acoplamiento. Un ejemplo de ello se evidencia en la adición de una nueva unidad de medida.

Controlador

Asignar la responsabilidad de controlar el flujo de eventos del sistema a clases específicas. Esto facilita la centralización de actividades (validaciones, seguridad). El controlador no realiza estas actividades, las delega en otras clases con las que mantiene un modelo de alta cohesión. Un error muy común es asignarle demasiada responsabilidad y alto nivel de acoplamiento con el resto de los componentes del sistema (Lerman, 2003).

Este patrón se evidencia en la arquitectura propuesta, en la clase Model, pues es la clase donde de la cual dependen el resto de las clases, evidenciándose de esta forma el patrón controlador.

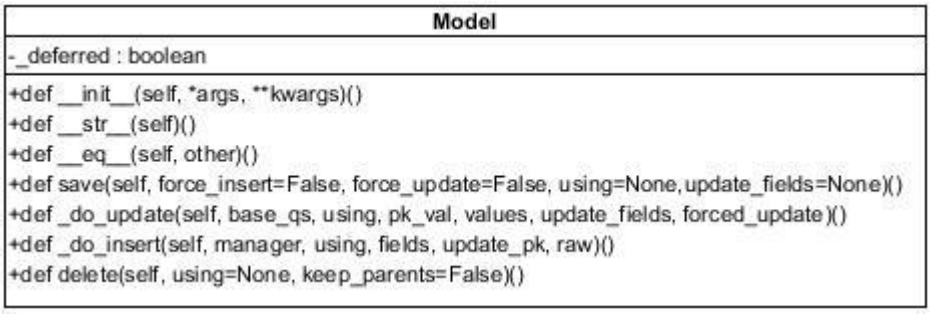


Figura 8: Patrón Controlador empleado en la solución modelado a través de un diagrama UML. Fuente: Elaboración propia.

Esta clase es la que contiene toda la implementación del sistema y actúa como controladora, brindando todo un conjunto de funcionalidades al resto de las clases existente en el caso de estudio.

2.7 Modelo de dominio

La construcción del modelo conceptual permite una mejor comprensión del dominio del problema; no es más que una representación visual de los conceptos y objetos del mundo real que son significativos para el problema; representando las clases conceptuales y los componentes de software.

En este caso como dicho negocio no está bien definido entre los clientes y los ejecutores del proyecto es generado el llamado Modelo de Dominio según el módulo que le corresponda:

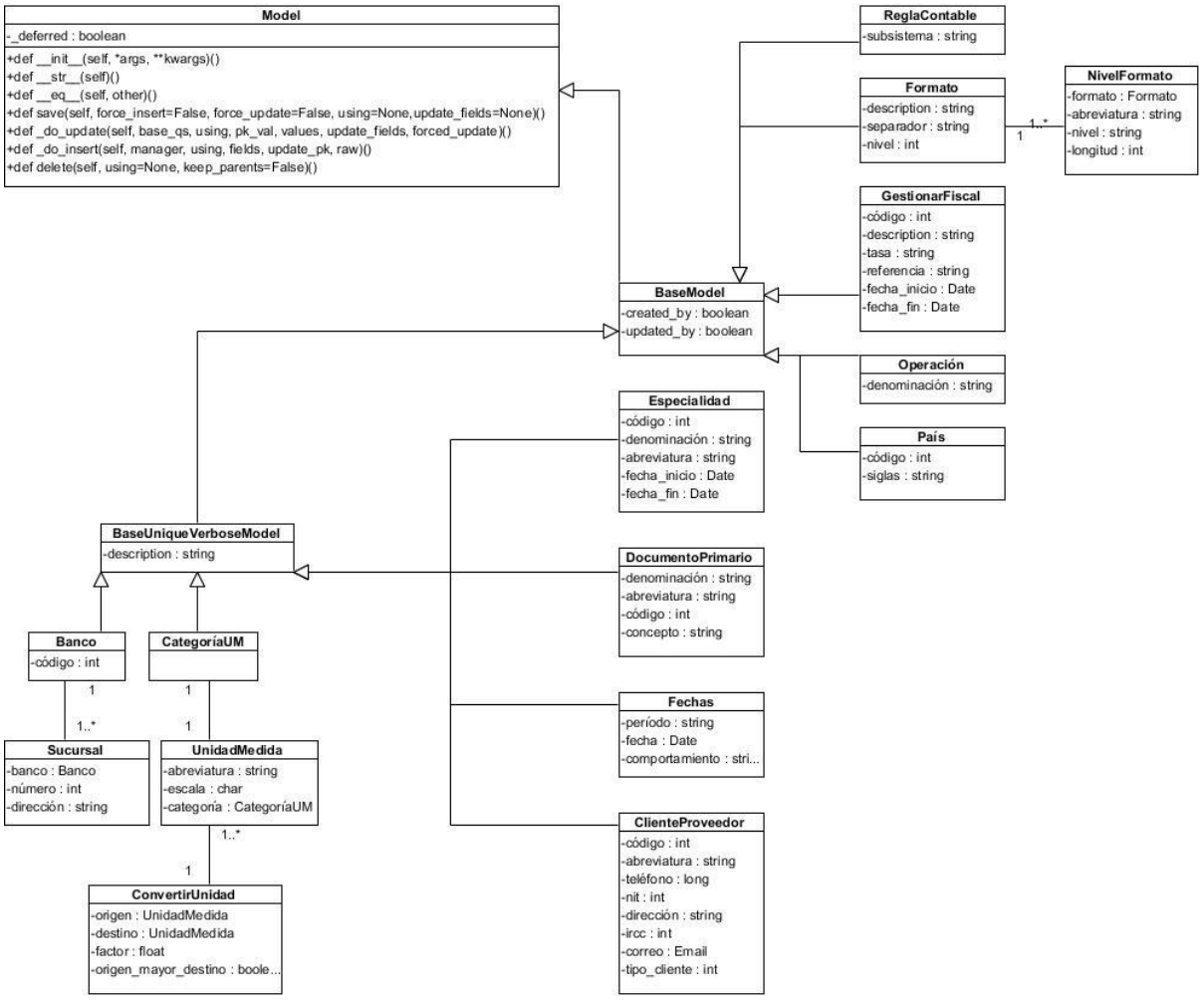


Figura 9: Modelo de dominio del módulo Configuración.

Fuente: Elaboración propia.

El diagrama contiene todas las clases y muestra la relación que existe entre ellas en el módulo Configuración, por ser uno de los que más información maneja. Se evidencia que la clase Model es la que maneja toda la información del sistema y actúa como clase controladora del sistema. El resto de las clases mostradas tienen una relación de herencia con la clase Model para obtener las funcionalidades que esta posee.

2.8 Diagrama de componentes

Un diagrama de componentes representa la separación de un sistema de software en componentes físicos (por ejemplo, archivos, módulos, paquetes, base de datos, código fuente, binario o ejecutable, y muestra las dependencias y organización existente entre estos componentes (Arizaca, 2011).

En el diagrama de componentes de la Figura 10, se evidencian las interacciones entre los componentes del sistema, está compuesto por cuatro paquetes fundamentales; Sistema, Módulos, Plantillas y Archivos Estáticos, además de que se anexa un paquete de bibliotecas que apoyarán la utilización de los componentes de forma acoplada y ágil. Estos paquetes con sus correspondientes componentes permiten un adecuado manejo de la seguridad, las configuraciones, las excepciones y errores del subsistema, así como de la información que se maneja.

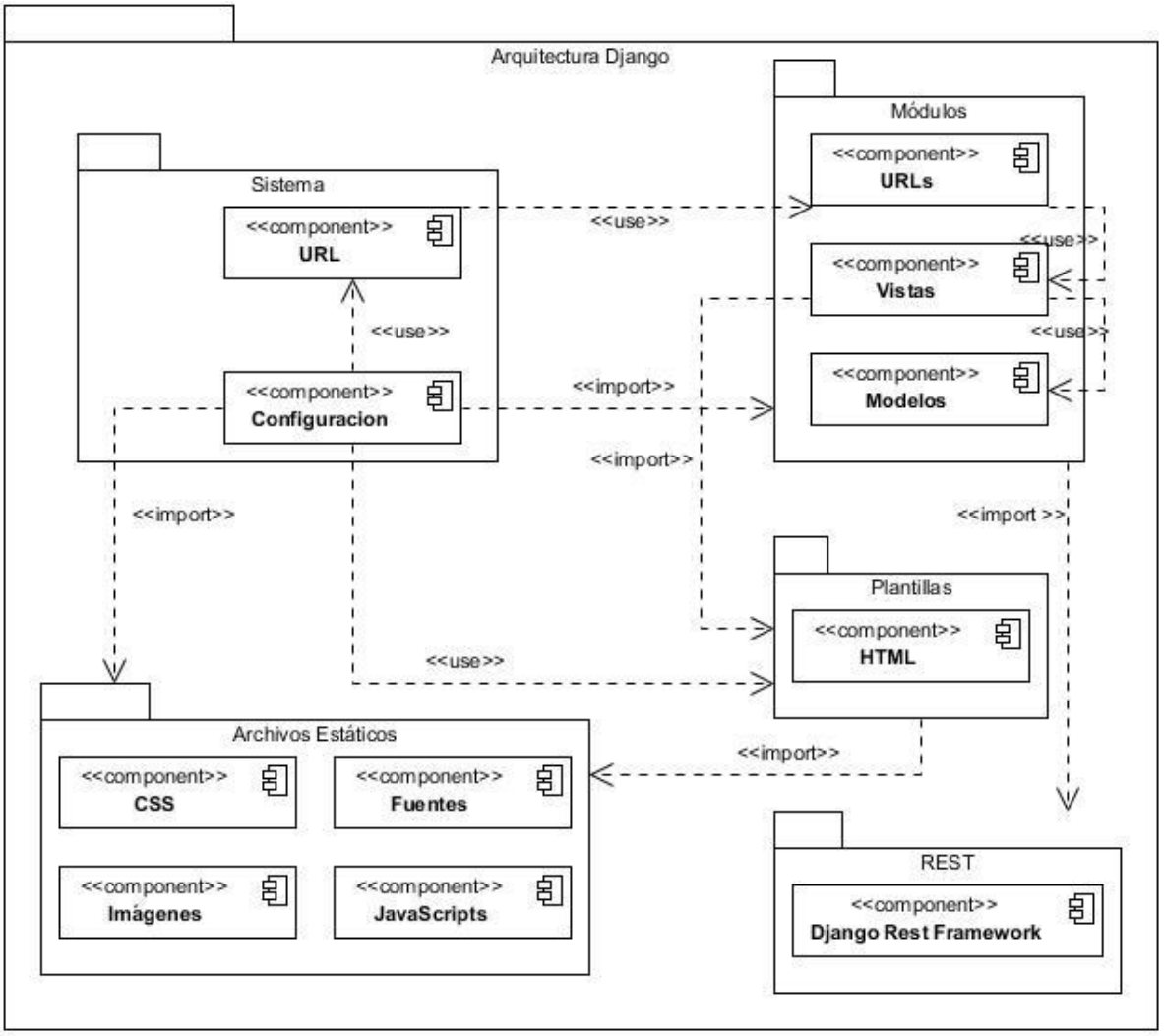


Figura 10: Diagrama de componentes.

Fuente: Elaboración propia.

El paquete Sistema maneja todas las configuraciones de las herramientas y contiene el componente Configuración; encargado de registrar todas las configuraciones de los demás paquetes, y URL; contiene y manipula todas las urls de todo el sistema. El paquete Módulo está compuesto por los componentes Vistas, Modelos y URLs, y representa físicamente cada uno de los módulos por los que está compuesta la solución, donde URL realiza peticiones a la vista (lógico del negocio) y estas a su vez se comunican con el modelo, que renderiza las peticiones al paquete Plantillas, que está compuesto por el componente HTML, que es donde se almacena todo el código fuente del sistema. Además el paquete

Plantillas se comunica con el paquete Archivos Estáticos, que es donde se encuentran todos los archivos JavaScript, los estilos CCS, las fuentes y las imágenes de la solución. Se puede evidenciar un paquete llamado REST, que es donde se encuentran un conjunto de bibliotecas que presenta la arquitectura, que apoyan toda la lógica del negocio, además que aporta facilidades, evidenciadas en la forma que se muestran los servicios del sistema.

2.9 Modelo de datos

Un modelo de datos es la definición lógica, independiente y abstracta de los objetos, operadores y demás que en conjunto contribuyen a la máquina abstracta con la que interactúan los usuarios (Compuesto en Evolution., 2000).

A continuación se muestra el modelo de datos del módulo Multimoneda, el cual permite realizar la representación lógica de los datos procesados por el mismo:

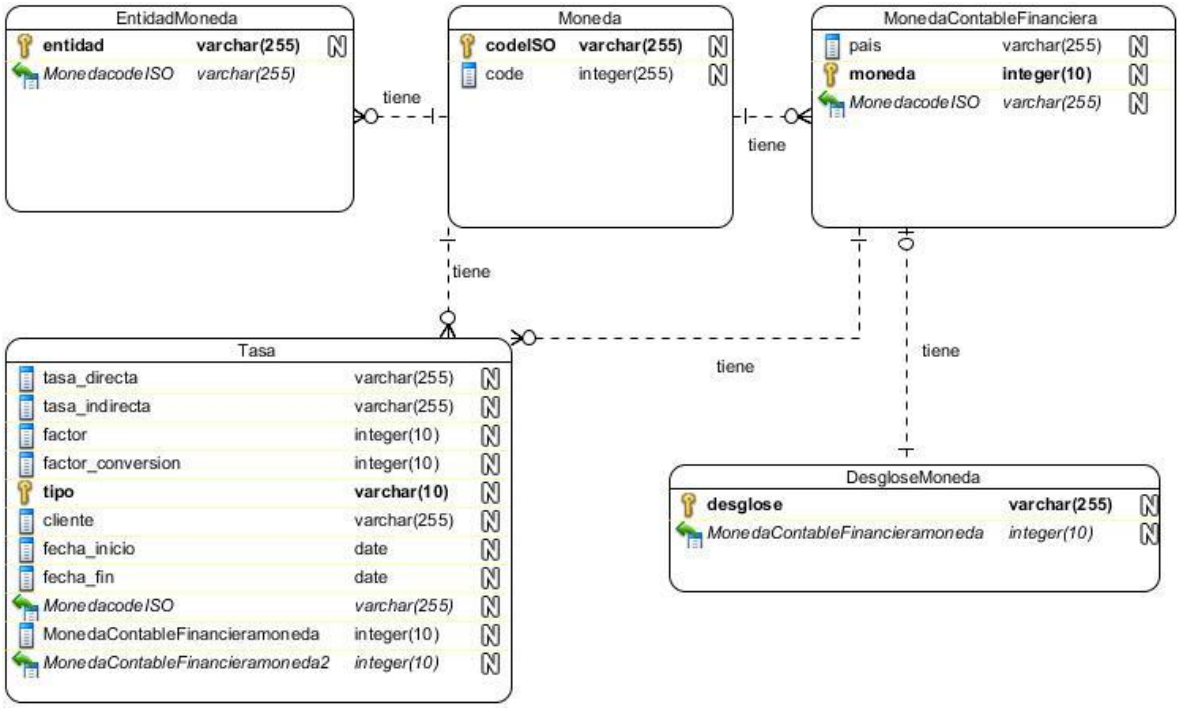


Figura 11: Modelo de dominio del módulo Multimoneda.

Fuente: Elaboración propia.

El modelo está compuesto por 5 entidades fundamentales que contienen toda la información del módulo, estas son las siguientes: EntidadMoneda, Moneda, MonedaContableFinanciera, Tasa y DesgloseMoneda, las cuales están relacionadas entre sí para darle una secuencia de flujos a los datos mostrados. La tabla Moneda es la que contiene todas las funcionalidades fundamentales de dicho modulo y las restantes realizan funciones específicas.

2.10 Diseño con estructuras

Al realizar la migración de los módulos de configuración básica de Cedrux a la nueva arquitectura, se realizaron cambios en la estructura interna de los módulos.

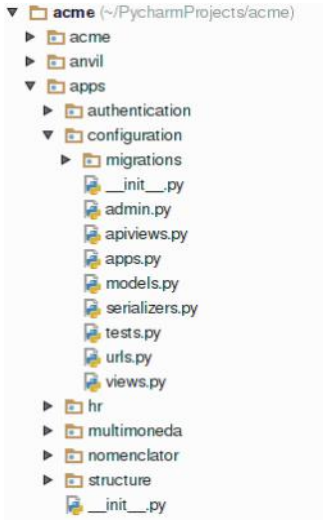


Figura 12: Estructura del proyecto. Fuente: Elaboración propia.

Los módulos del proyecto poseen la misma estructura de ficheros, los cuales se describen a continuación.

- ***models.py:*** Contiene las clases necesarias para interactuar con la base de datos y hace que los modelos solo sean responsables de definir sus campos en una sintaxis compacta y agradable.

```python
from __future__ import unicode_literals

from django.db import models

from anvil.models import BaseUniqueVerboseModel


class Bank(BaseUniqueVerboseModel):
    code = models.CharField(max_length=255)
```

Fig. 13: Ejemplo de definición de un modelo. Fuente: Elaboración propia.

- ***admin.py:*** Registra cada uno de los modelos en la interfaz administrativa, para que Django ofrezca una interfaz para cada uno de los modelos registrados, bajo el nombre de la aplicación y se pueda introducir datos directamente en ellos.

```python
from django.contrib import admin

from .models import Bank

admin.site.register(Bank)
```

Fig. 14: Ejemplo de registro de un modelo en la administración de Django. Fuente: Elaboración propia.

- ***serializers.py:*** Contendrá la definición de los modelos del módulo como recursos para la API REST.

```python
from rest_framework import serializers

from .models import Bank


class BankSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Bank
        fields = ('id', 'code')
```

Fig. 15: Ejemplo de definición de un modelo como recurso. Fuente: Elaboración propia.

- ***views.py:*** Este módulo de Python contendrá las vistas (controladores) del módulo en el que se encuentre.
- ***apiviews.py:*** Su función es similar a la del fichero ***views.py***, pero este se encargará de definir los controladores para la API REST.

```python
from anvil.apiviews import WSViewSet

from .models import Bank
from .serializers import BankSerializer


class BankViewSet(WSViewSet):
    queryset = Bank.objects.all()
    serializer_class = BankSerializer
    wsrest_uri = 'bank-list'
```

Fig. 16: Ejemplo de definición de controlador para la API REST. Fuente: Elaboración propia.

- ***urls.py:*** Contendrá los patrones de direcciones *urls* definidas para el módulo.

```python
from anvil.routers import router

from .apiviews import BankViewSet

router.register(r'configuracion/bank', BankViewSet)
```

Fig. 17: Ejemplo de definición de url. Fuente: Elaboración propia.

Django contiene en la estructura básica de un módulo otros ficheros, tales como: apps.py y tests.py, en dicha investigación no se hará uso de ellos.

2.11 Estándares de codificación

Los estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código. Los estándares de codificación en la nueva arquitectura van a permitir una mejor integración entre las líneas de producción y se establecerán las pautas que conlleven a lograr un código más legible y reutilizable, de tal forma que se pueda aumentar su capacidad de mantenimiento a lo largo del tiempo. La legibilidad del código fuente repercute directamente en lo bien que un programador comprende un sistema de software. La mantenibilidad del código es la facilidad con que el sistema de software puede modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores, o mejorar el rendimiento. Usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un código de alta calidad es de gran importancia para la calidad del software y para

obtener un buen rendimiento. El desarrollo sobre esta arquitectura se regirá por el estándar de código definido por la Python Software Foundation PEP8 (van Rossum, 2013) que a continuación se describe:

Tabla 1: Estándar de codificación. Fuente: Elaboración propia.

REGLA	DESCRIPCIÓN
Diseño del código.	Puedes continuar usando indentaciones de 8 (ocho) espacios.
Tabulaciones o espacios.	El método de indentación más popular en Python es con espacios. El segundo más popular es con tabulaciones, sin mezclar unos con otros.
Longitud de las líneas.	Limita todas las líneas a un máximo de 79 caracteres.
Líneas en blanco.	Separa funciones de alto nivel y definiciones de clase con dos líneas en blanco. Definiciones de métodos dentro de una clase son separadas por una línea en blanco.
Codificaciones.	El código en el núcleo de la distribución de Python siempre debería utilizar la codificación ASCII o Latin-1 (alias ISO-8859-1).
Importaciones.	Las importaciones deben estar en líneas separadas. Las importaciones siempre se colocan al comienzo del archivo, simplemente luego de cualquier comentario o documentación del módulo, y antes de globales y constantes y deben seguir el siguiente orden: Importaciones de la biblioteca estándar. Importaciones terceras relacionadas. Importaciones locales de la aplicación / biblioteca.
Nombres de paquetes y módulos.	Los módulos deben tener un nombre corto y en minúscula. Guiones bajos pueden utilizarse si mejora la legibilidad.

Nombres de clases.	Casi sin excepción, los nombres de clases deben utilizar la convención “CapWords” (palabras que comienzan con mayúsculas). Clases para uso interno tienen un guión bajo como prefijo.
Nombres de funciones.	Las funciones deben ser en minúscula, con las palabras separadas por un guión bajo, aplicándose éstos tanto como sea necesario para mejorar la legibilidad, seguidas de la palabra *def*.

2.12 Conclusiones del capítulo

En este capítulo se presentó una propuesta de arquitectura que permite mejorar la escalabilidad en la implementación de la arquitectura en los módulos de configuración básica de Cedrux, lo que contribuirá al desarrollo de los nuevos subsistemas del proyecto de forma ágil y a la mitigación de las dificultades identificadas en el proyecto.

Se realizó un estudio de los patrones arquitectónicos propuestos por dicha arquitectura y se fundamentaron los utilizados en el caso de estudio. Además se definió el estándar de codificación propuesto por Django basado en el estándar PEP8. Lo anterior contribuirá a la obtención de un código legible, con el uso de buenas prácticas de programación, mejorando de igual forma la mantenibilidad de la solución.

CAPÍTULO 3: VALIDACIÓN Y PRUEBAS

Introducción

En el presente capítulo se realizará la validación de la solución propuesta, para lo que se aplican diferentes métricas de diseño con el objetivo de valorar en qué medida se garantiza la calidad y completitud de la solución. Por último, se describirán las pruebas de caja negra y caja blanca realizadas al software y los resultados obtenidos de estas.

La calidad de un producto de software es el indicador que permite determinar si los procesos de construcción de software fueron apropiados. Pero, aunque a este renglón son dedicados grandes cantidades de esfuerzos, el software en sí nunca es perfecto. Es por esto que debe investigarse sobre los métodos y técnicas que garantizan calidad en los productos, con miras a generar propuestas concretas para aplicaciones con características específicas. Estos deben aplicarse desde las más tempranas etapas del desarrollo, en aras de detectar y corregir a tiempo una mayor cantidad de errores.

3.1 Evaluación de la solución a la problemática planteada

En siguiente epígrafe se reflejan los problemas evidenciados en la problemática, así como la solución de los mismos mediante la implementación de la nueva arquitectura.

Problemas evidenciados y la solución propuesta

1- Desactualización de la base tecnológica (ExtJs, PHP, Zend, Doctrine, PostgreSQL).

- La implementación de la arquitectura permite la utilización de la más reciente versión de Python en su versión 1.9 como lenguaje de programación, definido por el equipo de desarrollo, mitigando los problemas actuales que presenta Sauxe referente a la base tecnológica.

2- Las validaciones, excepciones y los servicios están registrados de forma global por lo que se hace muy difícil trabajar con esos ficheros de configuración. El mecanismo para el registro de las trazas afecta en gran medida el rendimiento de Cedrux.

- Actualmente el proyecto presenta un sistema de registro para las trazas que presenta dificultades en la forma de almacenamiento, lo que provoca que se almacenen todas las trazas del sistema en un periodo de tiempo relativamente corto, lo que provoca que el sistema afecte su rendimiento. Con la implementación de Django como arquitectura esta deficiencia queda suprimida, pues este

posee un conjunto de bibliotecas que realizan las validaciones, el tratamiento de excepciones y los servicios, ya sea de seguridad o de autenticación de manera automática. Para ello no necesita la implementación de código en nuevos componentes, haciendo uso de las buenas prácticas de programación, como la reutilización y la no incorporación de nuevos componentes que afecten la modularidad del sistema.

3- El mecanismo de comunicación entre los procesos de configuración básica realizado a través de los módulos Configuración, Nomencladores y Multimoneda es deficiente y muy rígido.

- La arquitectura que presenta Sauxe está basada en componentes, pero las malas prácticas de programación utilizadas provocaron la implementación de nuevos componentes para pocas funcionalidades. Esto provoca la existencia de un gran número de dependencia entre componentes que afectan directamente la estructura del proyecto y así su modularidad. Por su parte Django presenta una arquitectura que MVT que separa el modelo de la lógica del negocio, propiciando un código organizado y fácil de mantener por otros programadores.

4- La definición de los componentes no contó en todos los casos con el visto bueno de un especialista lo que provocó que en ocasiones se abusara de este aspecto y que aumentara el nivel de dependencias internas. Este elemento afecta directamente la modularidad de Cedrux.

5- Malas prácticas de programación, por ejemplo: código duplicado, sin el uso de estándares e insuficientemente documentado. Todo esto conlleva a la obtención de un código difícil de mantener e ilegible en ocasiones.

- Al no existir un estándar de codificación definido por el centro y todos los especialistas funcionales necesarios para auditar el trabajo realizado por los desarrolladores, estos realizaban sus propuestas sobre la base de argumentos teóricos insuficientes. Lo anterior trajo consigo la existencia de código duplicado por la mala reutilización de las funcionalidades, por su parte Django propone un estándar de código basado en el informe PEP8, que facilita al programador un código entendible y fácil de mantener, evitando así una implementación cargada de malas prácticas y errores que afecten la escalabilidad y mantenibilidad del sistema. En el caso de los especialistas funcionales, en la actualidad es política del centro consultar cada cambio con los mismos para evitar la introducción de errores conceptuales y de procesos de negocio.

6- Diseño de la BD realizado de forma incorrecta. Tablas sin usar, datos inconsistentes, implementaciones correspondientes a los procesos de negocio en la base de datos.

- Para mitigar esta deficiencia se diseñó la BD de forma que la información se encuentra en el archivo de configuración de Django setting.py, el cual brinda las siguientes opciones: database-name, indica a Django el nombre de la BD, database-user, le indica a Django cual es el nombre del usuario a usar cuando se conecte a la base de datos, database-password, le indica a Django cual es la contraseña a usar cuando se conecte a la BD y por ultimo database-host, le indica a Django cual es el host a utilizar cuando se conecte a la BD. El equipo optó por la utilización de SQlite en su versión 3, que reduce la latencia de acceso la base de datos, debido a que las llamadas a funciones son más eficientes que la comunicación entre procesos. Permite el almacenamiento de hasta 2 Terabytes de información, y permite la inclusión de nuevos campos.

3.2 Validación del diseño propuesto

Para la evaluación de la calidad del diseño propuesto para el sistema se realizó un estudio de las métricas básicas inspiradas en la escalabilidad y mantenibilidad que pudiera aportar la nueva arquitectura a la solución. Según Jaime Ramírez, “Las métricas de diseño a nivel de componentes se concentran en las características internas de los componentes del software e incluyen entre otras medidas la cohesión, acoplamiento y complejidad del módulo, medidas que pueden ayudar al desarrollador de software a juzgar la calidad de un diseño a nivel de los componentes” (Pressman, 2000).

Posteriormente, se describen los atributos que permiten medir la calidad del diseño propuesto.

- **Complejidad de implementación**: consiste en el grado de dificultad que tiene implementar un diseño de clases determinado.
- **Reutilización**: consiste en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.
- **Acoplamiento**: consiste en el grado de dependencia o interconexión de una clase o estructura de clase con otras, está muy ligada a la característica de reutilización.

Complejidad del mantenimiento: consiste en el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta, pero fuertemente en los costos y la planificación del proyecto.

3.3 Métricas Orientadas a Objetos

En este epígrafe se hace referencia a un conjunto de métricas orientadas a objetos (OO) propuestas por Chidamber y Kemerer (Chidamber, y otros, 1994). Estas métricas fueron empleadas para medir la calidad del diseño OO y están ajustadas a las características de la programación orientada a objetos (POO).

A continuación se establecen 6 métricas basadas en clases para medir atributos básicos en el diseño orientado a objetos: escalabilidad, mantenibilidad, acoplamiento, complejidad de una clase, reutilización, cohesión y herencia.

- Métodos ponderados por clase (Weighted Methods per Class –WMC)
- Profundidad del árbol de herencia (Depth of Inheritance Tree -DIT)
- Número de hijos (Number of children-NOC)
- Acoplamiento entre objetos (Coupling Between Object classes-CBO)
- Respuesta de una clase (Response for a class-RFC)
- Falta de cohesión en los métodos (Lack of cohesion in methods-LCOM)

3.3.1 Aplicación de las Métricas OO

A partir de un estudio realizado a las métricas OO antes mencionadas se hace una selección de aquellas que evidencian que el producto software es escalable y mantenible, demostrado a través de la aplicación de las mismas.

Con el objetivo de determinar el grado de escalabilidad y mantenibilidad de la del diseño propuesto en el capítulo anterior, se aplicaron las siguientes métricas:

Resultado de la métrica Acoplamiento entre objetos (CBO)

Para aplicar esta métrica es necesario conocer la cantidad de clases con las que se relaciona una clase, excluyendo las relaciones por herencia.

La siguiente tabla muestra los resultados arrojados de la métrica realizada al módulo Configuración.

Tabla 2: Colaboraciones por clase. Fuente: Elaboración propia.

Nombre de la clase	No. Colaboraciones
BaseModel	0
BaseUniqueNameModel	1
BaseUniqueVerboseModel	1
Bank	2
Sucursal	1
CategoriaUM	2
UnidadMedida	1
ConvertirUnidad	2
Especialidad	0
EntidadPais	2
GestionarFiscal	0
ClienteProveedor	2
Formato	1
NivelFormato	2
DocumentoPrimario	0
Fechas	0
País	0
Operación	0
ReglaContable	0

Chidamber y Kemerer sugieren que en cuanto más acoplamiento se evidencia en una clase, más difícil será reutilizarla. Además, las clases con excesivo acoplamiento dificultan la comprensibilidad y hacen más difícil el mantenimiento por lo que será necesario un mayor esfuerzo. Las clases deberían de ser lo más independientes posible, y aunque siempre es necesaria una dependencia entre clases, cuando ésta es grande, la reutilización puede ser más cara que la reescritura. Al reducir el acoplamiento se reduce la complejidad, se mejora la modularidad y se promueve la encapsulación.

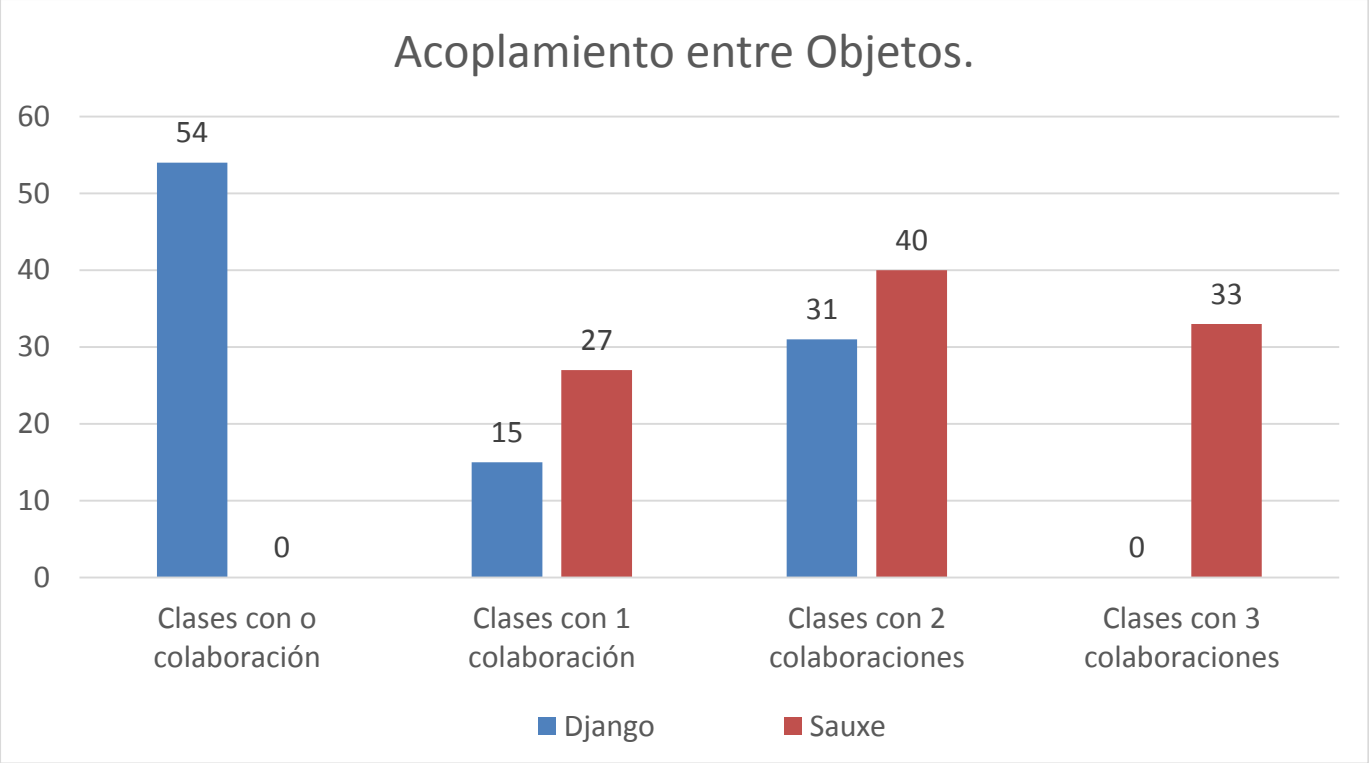


Figura 18: Acoplamiento entre objetos. Fuente: Elaboración propia.

La Figura 18 muestra las colaboraciones existentes entre las clases del módulo Configuración una vez desarrollada la arquitectura nueva y el mismo procedimiento aplicado al sistema arquitectura Sauxe. Se puede evidenciar que el mayor por ciento de las clase con 0 colaboraciones lo presenta la arquitectura Django, demostrándose que las clases presentan un acoplamiento relativamente bajo, permitiendo que aumente el grado de reutilización de las clases, de igual forma sucede con las clase con colaboración 1 y 2 respectivamente. Este resultado también ayuda a que las pruebas o modificaciones que sean necesarias, resulten fáciles de ejecutar, lo que demuestra que existe un alto grado de mantenibilidad del sistema. Por su parte las clases que presentan colaboración 3, muestran un mayor por ciento en las clases de la arquitectura Sauxe, evidenciándose un alto acoplamiento, por lo que serían de mayor complejidad y difícil de mantener (Chidamber & Kemerer, 1994).

Resultado de la métrica Profundidad del árbol de herencia (DIT)

Para aplicar esta métrica es preciso conocer la cantidad de relaciones de herencia que tiene una clase. Seguidamente se muestran los datos recopilados y su representación gráfica.

La siguiente tabla muestra los resultados arrojados de la métrica realizada al módulo configuración una vez implementado el sistema con la nueva arquitectura.

Tabla 3: Profundidad del árbol de herencia. Fuente: Elaboración propia.

Nombre de la clase	Profundidad árbol herencia
Models	0
BaseModel	1
BaseUniqueNameModel	2
BaseUniqueVerboseModel	2
Bank	3
Sucursal	3
CategoriaUM	3
UnidadMedida	3
ConvertirUnidad	3
Especialidad	3
EntidadPais	3
GestionarFiscal	3
ClienteProveedor	3
Formato	3
NivelFormato	3
DocumentoPrimario	3
Fechas	3
País	3
Operación	3

La Figura 19 muestra el nivel que tiene cada una de las clases en el árbol de herencia en la arquitectura Django. Las que tienen valor cero indican que no tienen relaciones de herencia o que están en el nivel cero del árbol. Las restantes poseen valor de 1 y 3. Estos resultados indican que las clases serán fáciles de desarrollar y de mantener debido que cuanto más profunda se encuentra una clase en la jerarquía, mayor será la probabilidad de heredar un mayor número de métodos.

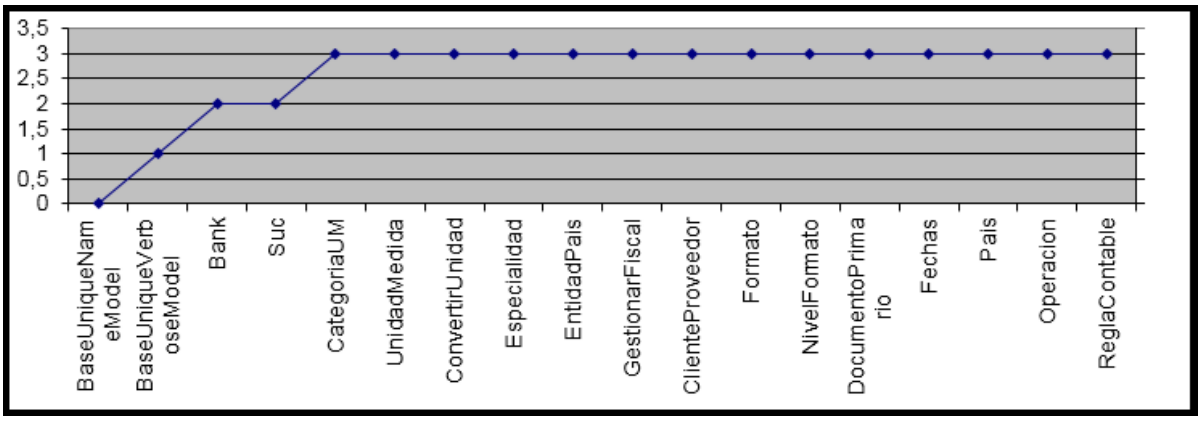


Figura 19 Profundidad árbol de herencia aplicada a Django. Fuente: Elaboración propia.

La Figura 20 muestra el nivel que tiene cada una de las clases en el árbol de herencia en la arquitectura Sauxe. Las que tienen valor 1 y 2 indican que son clase fáciles de desarrollar y mantener. Las restantes poseen oscilatorios, algunos de estos superan el valor 5, evidenciándose que las clases presentan un nivel alto en el árbol de herencia. Estos resultados indican que las clases serán difíciles de desarrollar y de mantener debido q que cuanto más profunda se encuentra una clase en la jerarquía, mayor será la probabilidad de heredar un mayor número de métodos.

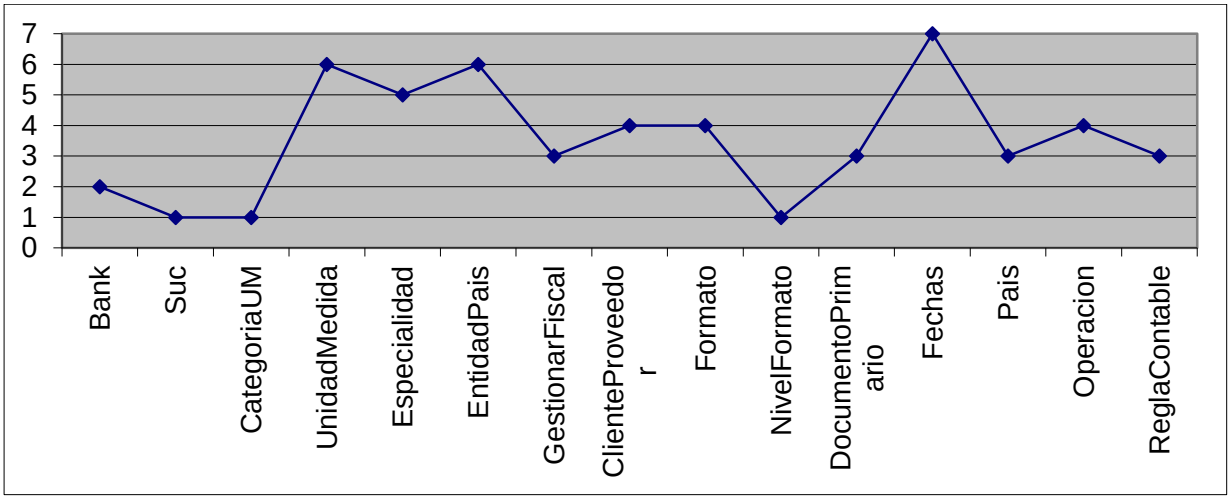


Figura 20: Profundidad árbol de herencia aplicada a Sauxe. Fuente: Elaboración propia.

Es necesario plantear que los valores mostrados constituyen estimaciones conservadoras debido a que generalmente durante la fase de implementación se hace necesario incluir nuevas funcionalidades y atributos. Esta métrica define una escala del 1 al 5 que es lo sugerido por la bibliografía consultada, donde 5 es el nivel superior. Se analizaron los resultados obtenidos por algunas métricas para el módulo Configuración por ser uno de los de mayor responsabilidad en el sistema.

3.3.2 Resultados generales arrojados por las métricas

Una vez aplicadas algunas de las métricas propuestas por Chidamber y Kemerer (Chidamber & Kemerer, 1994) se demostró que los resultados arrojados demuestran que las clases implementadas sobre la nueva arquitectura serán fáciles de desarrollar y mantener en un nivel superior comparado con la arquitectura Sauxe. A través de tablas se demuestra que los resultados arrojados por las métricas favorecen un bajo acoplamiento entre las clases de Django, posibilitando que no ocurran grandes cambios en el resto de las clases de cualquier módulo a la hora de modificar una.

3.4 Pruebas

Las pruebas realizadas a un software determinan el estado de la calidad del producto. En este proceso se ejecutan pruebas dirigidas a componentes del software o al sistema de software en su totalidad, con el objetivo de medir el grado en que el software cumple con los requerimientos. El proceso de pruebas, sus objetivos y los métodos y técnicas usados se describen en el plan de prueba (IEEE, 2012).

Las pruebas son un elemento crítico para la garantía del correcto funcionamiento del software, por lo que entre sus objetivos se encuentran los siguientes:

1. Detectar defectos en el software.
2. Verificar la integración adecuada de los componentes.
3. Verificar que todos los requisitos se han implementado correctamente.
4. Identificar y asegurar que los defectos encontrados se han corregido antes de entregar el software al cliente.
5. Diseñar casos de prueba que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo (IEEE, 2012).

Seguidamente se describe como se aplicaron cada una de las pruebas en el sistema.

3.3.1 Pruebas de aceptación

El uso de cualquier producto de software tiene que estar justificado por las ventajas que ofrece. Sin embargo, antes de empezar a usarlo es muy difícil determinar si sus ventajas realmente justifican su uso. Para lograr esta determinación es utilizada la llamada “prueba de aceptación”.

En esta prueba se evalúa el grado de calidad del software con relación a todos los aspectos relevantes para que el uso del producto se justifique.

Para eliminar la influencia de conflictos de intereses y para que sea lo más objetiva posible, la prueba de aceptación nunca debería ser responsabilidad de los ingenieros de software que han desarrollado el producto. Para la preparación, la ejecución y la evaluación de la prueba de aceptación ni siquiera hacen falta conocimientos informáticos. El cliente es el mayor responsable de verificar cada una de las pruebas y de priorizar la corrección de las pruebas que fallan (IEEE, 2012).

En el siguiente epígrafe se describe como se realizó esta prueba a cada uno de los requisitos funcionales propuestos:

Prueba asociada al requisito adicionar banco a la base datos

Esta prueba evaluará el proceso de adición de un banco en el sistema, cuando el usuario omite el llenado de algunos campos obligatorios, el sistema no recibe los datos que espera y los toma como un error, notificando al usuario que faltan campos por llenar. Se realizaron 2 iteraciones, encontrando algunas no conformidades en la primera y 0 en la última, teniendo éxito en las pruebas.

Tabla 4: Adicionar banco. Fuente: Elaboración propia.

Número de caso de prueba : 1	**Requisito Adicionar Banco**
Nombre de la prueba: Adicionar banco con campos incorrectos.	
Descripción de la prueba: El usuario accede al formulario de adición de un nuevo banco en el cual llena los datos requeridos para dicho proceso, estos datos son incorrectos o existen campos obligatorios vacíos.	
Condiciones de ejecución: El servidor de base de datos de la aplicación en ejecución se encuentre en funcionamiento.	
Entradas:	

-nombre del banco. -siglas. -código.
Resultado esperado: El sistema alerta al usuario de los errores cometidos en el momento de completar el formulario de creación de una nueva conexión, se muestra un mensaje de error.
Evaluación: Satisfactoria.

3.3.2 Pruebas de caja blanca

La Prueba de Caja Blanca también se conoce como Prueba de Caja Transparente o de Cristal (Pressman, 2000). En ese sentido el criterio de selección de casos de prueba buscará cierta cobertura no solo para la determinación de caminos independientes, sino también de valores para las condiciones de bucles dentro y fuera de sus límites operacionales basados en el contenido de los módulos.

Esta prueba consiste específicamente en como diseñar los casos de prueba atendiendo al comportamiento interno y la estructura del programa, examinándose la lógica interna sin considerar los aspectos de rendimiento.

Dentro de la prueba de caja blanca se incluyen las Técnicas de Pruebas. A continuación se describe la técnica del camino básico la cual se utilizará en la presente investigación:

Prueba del Camino Básico: Permite obtener una medida de la complejidad lógica de un diseño y usar la misma como guía para la definición de un conjunto de caminos básicos. Los casos de prueba obtenidos garantizan que durante la prueba se ejecute al menos una vez cada sentencia del programa.

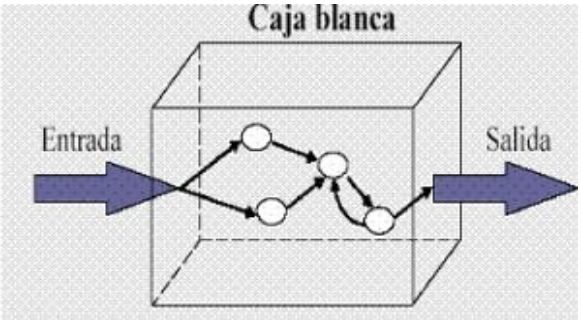


Figura 21: Caja Blanca.

La prueba de caja blanca empleada en la solución desarrollada fue la Prueba del Camino Básico, la cual permite obtener una medida de la complejidad lógica del diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución, garantizando con estos que durante la prueba se ejecute por lo menos una vez cada sentencia del programa.

Para la realización de este tipo de prueba se hace necesario calcular la complejidad ciclomática del fragmento de código a analizar. Seguidamente se enumeran las sentencias de código del procedimiento realizado sobre el método save() que se encuentra en la clase Model() y el grafo de flujo correspondiente.

```
def save(self, *args, **kwargs):  # /1
    if self.parent == self or str(self.parent) in self.all_subordinates:  # /2
        return# /3
    super(BaseTreeModel, self).save(*args, **kwargs)  # /4
    self._order = 0  # /4
    if self.parent:  # /5
        self._order = self.parent._order + 1  # /6
    super(BaseTreeModel, self).save(*args, **kwargs)  # /6
```

Figura 22: Método save().Fuente: Elaboración propia.

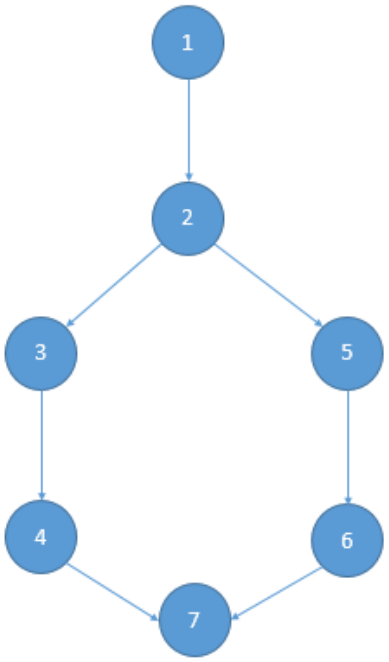


Figura 23: Gráfico del método save().Fuente: Elaboración propia.

El grafo representado (Figura 23) permite observar fácilmente los caminos que se pueden recorrer al ejecutar el código al cual pertenece este grafo. La representación gráfica del código a través del grafo ayuda a determinar la cantidad de aristas, regiones, nodos y nodos predicados del grafo.

Fórmulas para calcular la complejidad ciclomática:

1. V (G) = R

Donde “R” representa la cantidad de regiones en el grafo.

V (G) = 2

2. V (G) = (A - N) + 2

Donde “A” es la cantidad de aristas y “N” la cantidad de nodos.

V (G) = (7 - 7) + 2

V (G) = 2

3. V (G) = P + 1

Siendo “**P**” la cantidad de nodos predicados (son los nodos de los cuales parten dos o más aristas).

V (G) = 1 + 1

V (G) = 2

Luego de haber realizado el cálculo de la complejidad ciclomática aplicando las tres fórmulas se puede concluir diciendo que dicha complejidad es igual a 2 ya que es el mismo valor en cada uno de los resultados obtenidos. Dicha complejidad representa el límite mínimo del número total de casos de pruebas para el procedimiento tratado.

Tabla 5: Caminos básicos del flujo. Fuente: Elaboración propia.

Números	Caminos básicos
1	1-2-3-4-7
2	1-2-5-6-7

Una vez definidos los caminos básicos del flujo se pasa a ejecutar los casos de prueba para cada uno de estos. Para definir los casos de prueba es necesario tener en cuenta:

- **Descripción:** se describe el caso de prueba y de forma general se tratan los aspectos fundamentales de los datos de entrada.
- **Condición de ejecución:** se especifica cada parámetro para que cumpla una condición deseada y así ver el funcionamiento del procedimiento.
- **Entrada:** se muestran los parámetros que serán la entrada al procedimiento.
- **Resultados Esperados:** se expone el resultado esperado que debe devolver el procedimiento después de efectuado el caso de prueba.

Caso de prueba para el camino básico # 1.

Descripción: Los datos de entrada son atributos de bancos.

Condición de ejecución: no existan bancos en el sistema.

Entrada: Created_by, updated_by, name, description, code.

Resultados Esperados: Se adiciona el banco satisfactoriamente.

Resultados: No se adiciona el banco.

Salida: -1

Caso de prueba para el camino básico # 2.

Descripción: Los datos de entrada son atributos de bancos.

Condición de ejecución: existan bancos en el sistema.

Entrada: Created_by, updated_by, name, description, code.

Resultados Esperados: Se adiciona el banco satisfactoriamente.

Resultados: No se adiciona el banco.

Salida: 1

3.3.3 Pruebas de caja negra

Se refiere a las pruebas que se llevan a cabo sobre la interfaz del software (Pressman, 2000), por lo que los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce una salida correcta, así como que la integridad de la información externa se mantiene. Esta prueba examina algunos aspectos del modelo fundamentalmente del sistema sin tener mucho en cuenta la estructura interna del software.

Pressman plantea en 1998 que estas pruebas permiten encontrar:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a las Bases de Datos externas.
- Errores de rendimiento.
- Errores de inicialización y terminación.

En la preparación de los casos de prueba son necesarios un conjunto de datos que ayuden en la ejecución de estos y que permitan que el sistema se ejecute en todas sus variantes, los datos utilizados pueden ser válidos o inválidos siempre en consecuencia de si lo que se desea hallar es un error o probar una funcionalidad. Los datos se escogen atendiendo a las especificaciones del problema, sin importar los detalles internos del programa, a fin de verificar que el programa corra bien.

Según Pressman público en el año 2000, para desarrollar la prueba de caja negra existen varias técnicas, entre ellas están:

1. Técnica de la Partición de Equivalencia: esta técnica divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.

2. Técnica del Análisis de Valores Límites: esta técnica prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.

3. Técnica de Grafos de Causa-Efecto: es una técnica que permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

En este caso se utilizará la Partición de Equivalencia que es una de las más efectivas pues permite examinar los valores válidos e inválidos de las entradas existentes en el software, descubre de forma

inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. La partición equivalente se dirige a la definición de casos de pruebas que descubran clases de errores, reduciendo así el número de casos de prueba a desarrollar.

Diseño de Casos de Prueba

Condiciones de ejecución:

- Se debe identificar y autenticar ante el sistema y además debe tener los permisos para ejecutar esta acción.
- Se debe seleccionar el subsistema de Configuración.
- Se debe seleccionar la opción Configuración /Banco /Adicionar banco.
- Se deben llevar los campos esperados por el sistema.
- Se debe haber adicionado el banco.

Requisito a probar: Eliminar banco.

Tabla 6: Diseño de caso de prueba. Eliminar banco.

Fuente: Elaboración propia.

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Eliminar banco	Se eliminan los bancos seleccionados del sistema.	El sistema procesa la petición y muestra el mensaje: “The bank we was deleted succesfull”.	Se insertan los parámetros en los campos mostrados. -Se presiona el botón Save. -Se validan los datos. -el sistema muestra un mensaje indicando el éxito de la operación.
EC 1.2 Eliminar Banco sin seleccionar un Banco.	No se selecciona ningún banco para eliminar.	No se activa el botón DELETE.	No se activa el botón DELETE.

Descripción de las variables.

Tabla 7: Descripción de las variables. Eliminar banco. Fuente: Elaboración propia.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	name	ComboBox	No	Caracteres alfanuméricos

3.5 Conclusiones del capítulo.

Durante el desarrollo de este capítulo se evidenció que las dificultades enunciadas en la problemática de la investigación fueron corregidas con el desarrollo de la nueva arquitectura. Además, se aplicaron las métricas para evaluar el diseño y se realizaron pruebas de aceptación a los principales requisitos propuestos, así como las pruebas de caja blanca y caja negra al sistema, analizándose los resultados obtenidos. Se realizó una comparación a partir de los resultados arrojados por las métricas, concluyendo que la implementación utilizando la arquitectura Django contribuirá a la mejora del producto Xedro-ERP. Por su parte, los resultados satisfactorios arrojados por las pruebas demostraron que el sistema cuenta con las características y funcionalidades para las que fue creado.

Conclusiones generales

Una vez concluido el trabajo de investigación para el desarrollo de una propuesta de solución para las dificultades identificadas en los módulos de configuración básica de Cedrux, se evidenció el cumplimiento de los objetivos trazados, razón por la cual se arriba a las siguientes conclusiones generales:

- La utilización Django como alternativa arquitectónica contribuyó a la realización de aplicaciones web con un alto grado de escalabilidad, evidenciadas en la utilización de patrones arquitectónicos para la separación de sus módulos; así como un alto grado de mantenibilidad a partir de las bibliotecas que posee para proveer simplicidad y legibilidad al código.
- La inclusión de bibliotecas a los módulos de Django como parte de su arquitectura, como es el caso de Django CORS Headers, Django Rest Framework (DRF), Auth Token, Djoser y Django Websockets, permitieron un mejor acoplamiento entre los componentes.
- La validación de la solución a través de la aplicación de métricas, evidenció que la escalabilidad y mantenibilidad que brinda la arquitectura Django una vez desarrollada para los módulos de configuración básica de Cedrux es superior a la propuesta por Sauxe.

Bibliografía

1. Acosta, N. (2016). *Herramienta de desarrollo para web gratis*. Obtenido de https://neftaliacosta.com/herramientas-desarrollo-web-gratis/
2. Arizaca, R. E. (2011). *Artefacto: Diagrama de Componentes.* La Paz, Bolivia.
3. BD Redis. (1012). *Redis. Base Datos.*
4. CAChemE. (2013). *Ventajas de Python*. Obtenido de http://cacheme.org/ventajas-python-vs-fortran-c/
5. CEIGE. (2015). *Informe del estudio realizado a Cedrux.* Universidad de las Ciencias Informáticas, Centro de Informatización de Entidades, La Habana.
6. Chidamber, S. R., & Kemerer, C. F. (1994). *A Metrics Suite for Object Oriented Design. s.l.: IEEE Transactions on Software Engineering. 1994. vol. 20,.*
7. Compuesto en Evolution. (2000). *Diseño de base de datos relacionales. .* España.
8. Dos Ideas. (2008). *Introducción a los servicios web RESTful*. Obtenido de http://www.dosideas.com/noticias/java/314-introduccion-a-los-servicios-web-restful.html
9. Dos Ideas. (2012). *Introduccion a los servicios web restful.* Obtenido de http://www.dosideas.com/noticias/java/314-introduccion-a-los-servicios-web-restful.html
10. Gómez Baryolo, O. (2010). *Solución informática de autorización en entornos multientidad y.* Habana.
11. González Brito, H. R. (2007). *ERP Cubano, Un paso estratégico para la consolidación del Software Libre en Cuba.* Ciudad de laHabana.
12. Gracia, J. (2016). *Ingeniero de Software*. Obtenido de http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php
13. Grosso, A. (2011). *Patrones-Grasp*. Recuperado el 10 de Diciembre de 2015, de http://www.practicasdesoftware.com.ar/2011/03/patrones-grasp/
14. Gutierrez, J. (2014). *Framework de desarrollo web.*

15. IEEE. (2005). *Calidad de Software*. Obtenido de www.calidadysoftware.com/testing/pruebas_unitarias1.php.
16. IEEE. (2012). *Pruebas de software*. Obtenido de pruebasdesoftware.com/pruebadeaceptacion.htm.
17. Infante Montero, S. (2012). *LibrosWeb*. Recuperado el 18 de Junio de 2015, de librosweb.es: http://librosweb.es/5_2_ El patrón de diseño MTV (El libro de Django 1_0).htm
18. ISO/IEC/IEEE. (2006). *Software engineering – software life cycle processes – maintenance.*
19. Lau, R. (2011). *Entorno de Desarrollo Integrado (IDE).* Recuperado el 14 de Enero de 2016, de Vigo:Unión Webmaster Libres: https://sites.google.com/site/vigomiciudad/otras-cosa-intersantes/ide-entornos-de-desarrollo-integrado
20. Lazo Ochoa, R. (2007). *Modelo de referencia para el desarrollo arquitectónico de sistemas de software en dominios de gestión.* Habana.
21. Lerman, G. (2003). *UML y Patrones, 2da Edición.*
22. Nieto Muñoz, M. (2015). Diseño y Arquitectura de Django. Habana.
23. Pedrosa, M., & otros. (2011).
24. Pressman, R. (2000). *Pruebas de Software.*
25. Python. (2016). *Python Software Corporation*. Obtenido de https://www.python.org/
26. Rodriguez Tello, E. A. (2012). *Conceptos básicos de Ingeniería de Software*. Obtenido de http://queaprendemoshoy.com/¿que-es-la-calidad-i-¿por-que-este-concepto-“suena-tan-bien”/
27. Roldan Estebanez, D. (2015). *Sistema multiplataforma de gestión integral de huertos urbanos.*
28. Sánchez, T. R. (2015). *Metodología de desarrollo para la Actividad productiva de la UC.* UCI.
29. van Rossum, G. (2013). *Guía de estilo para el código Python – PEP 8 en Español.*
30. VI Congreso del Partido Comunista de Cuba. (2011). *Lineamientos de la Política Económica y Social del Partido y la Revolución.*

Anexos

Anexo 1.

Diagrama de clases del módulo Multimoneda.

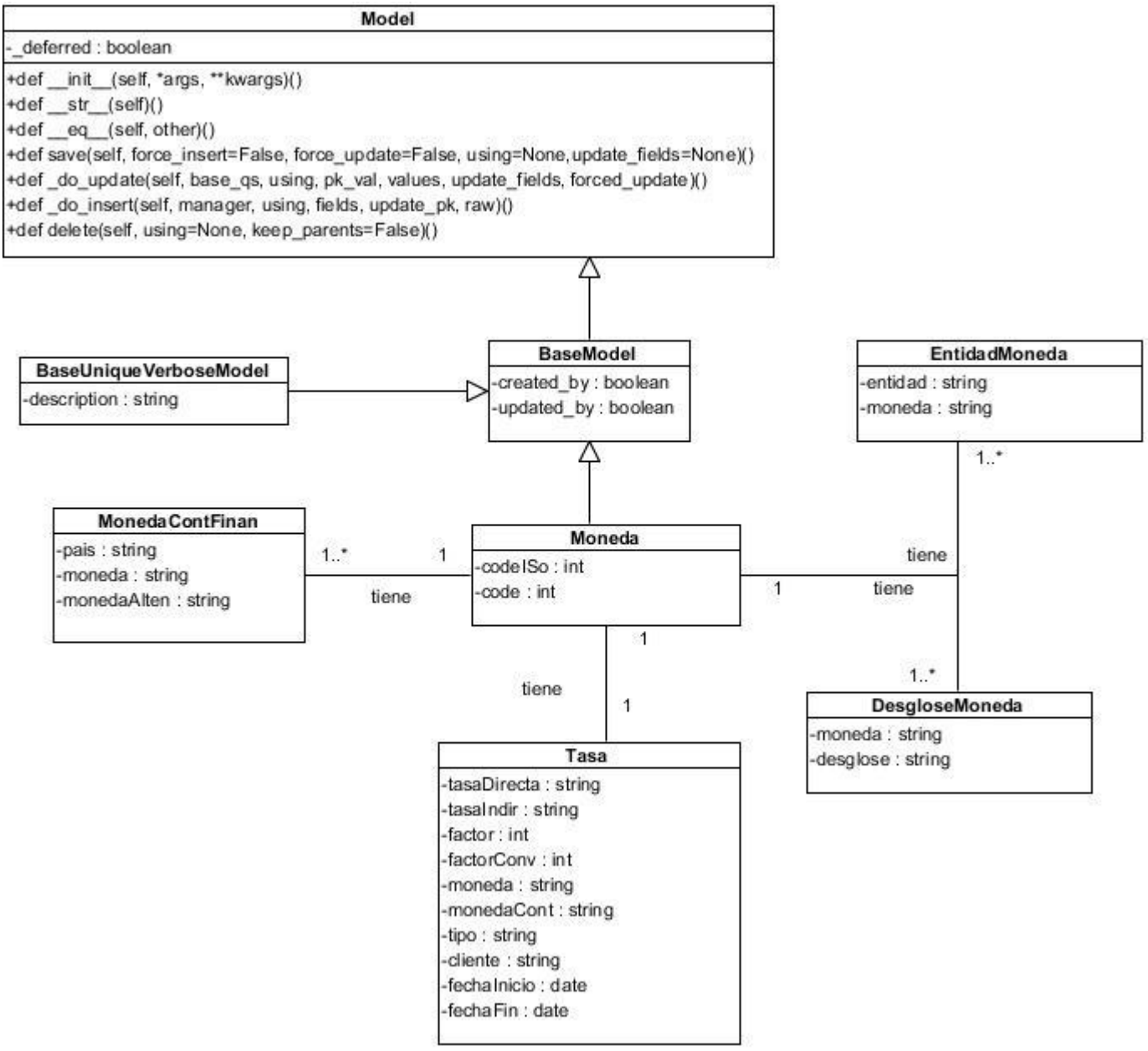


Diagrama de clases. Fuente: Elaboración propia.

Anexo 2.

Diagrama de clases del módulo Nomencladores.

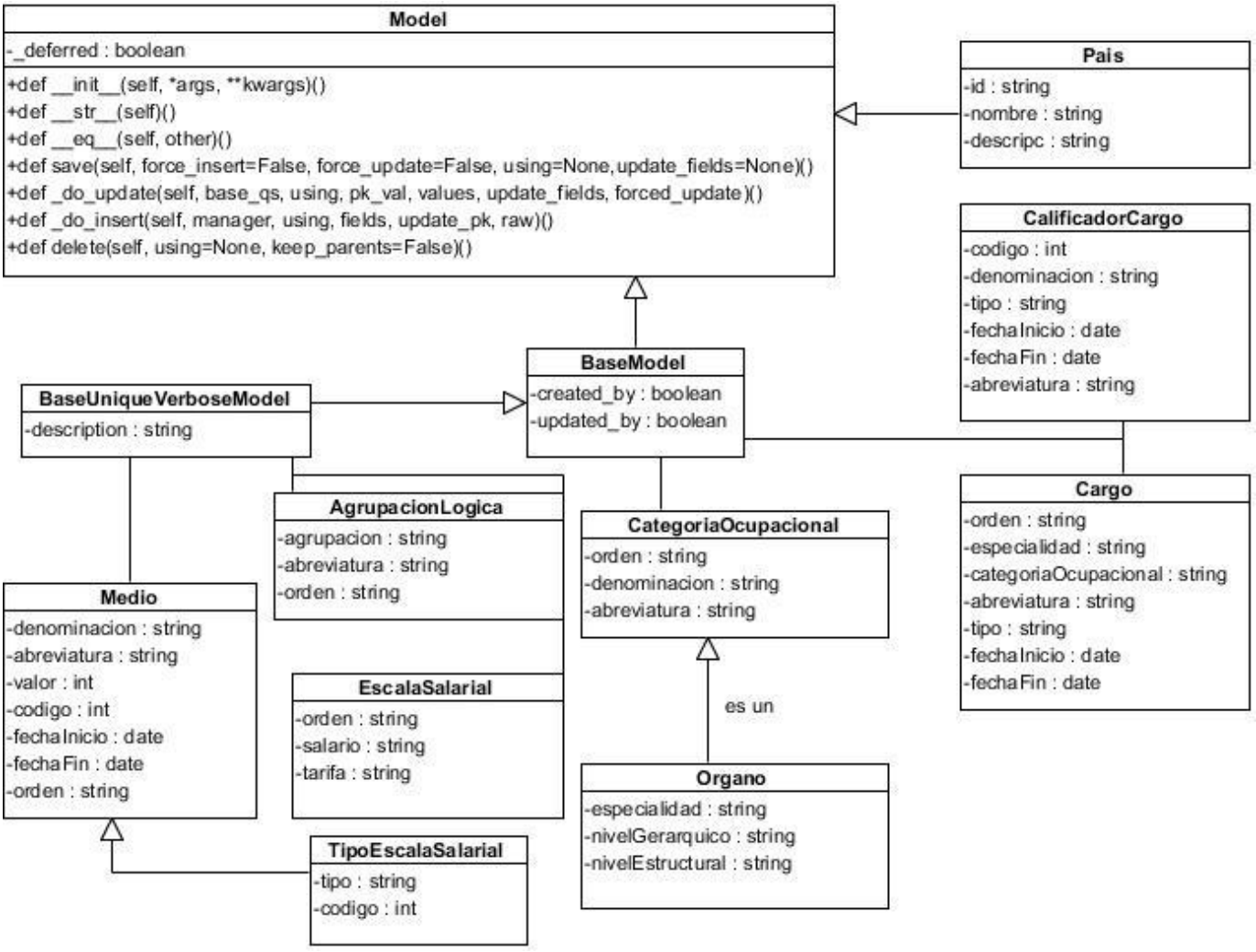


Figura 24. Diagrama de clases. Fuente: Elaboración propia.

Anexo 3.

Diagrama Entidad-Relación del módulo Configuración.

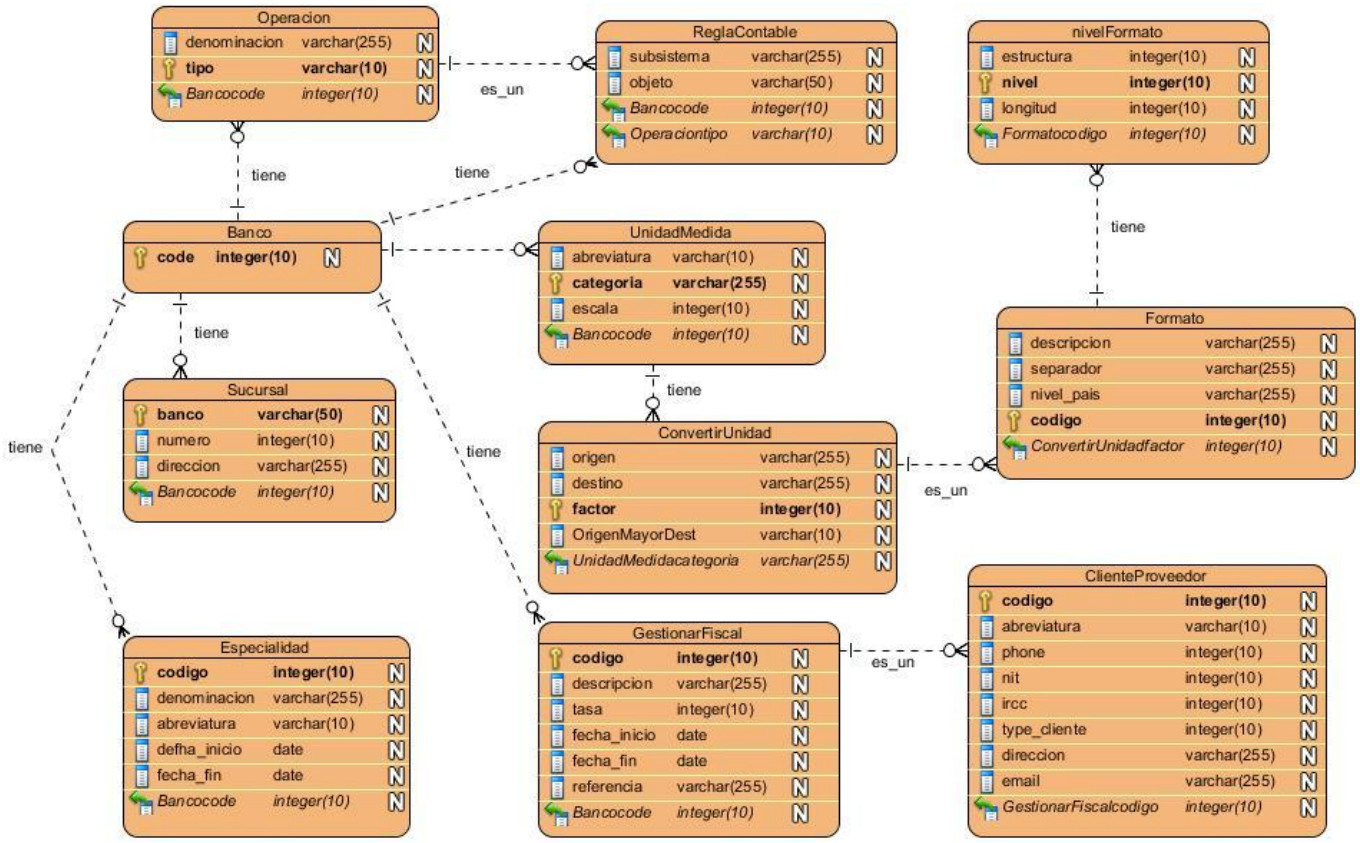


Figura 25.. Diagrama Entidad-Relación. Fuente: Elaboración propia.

Anexo 4,

Diagrama Entidad-Relación del módulo Nomencladores.

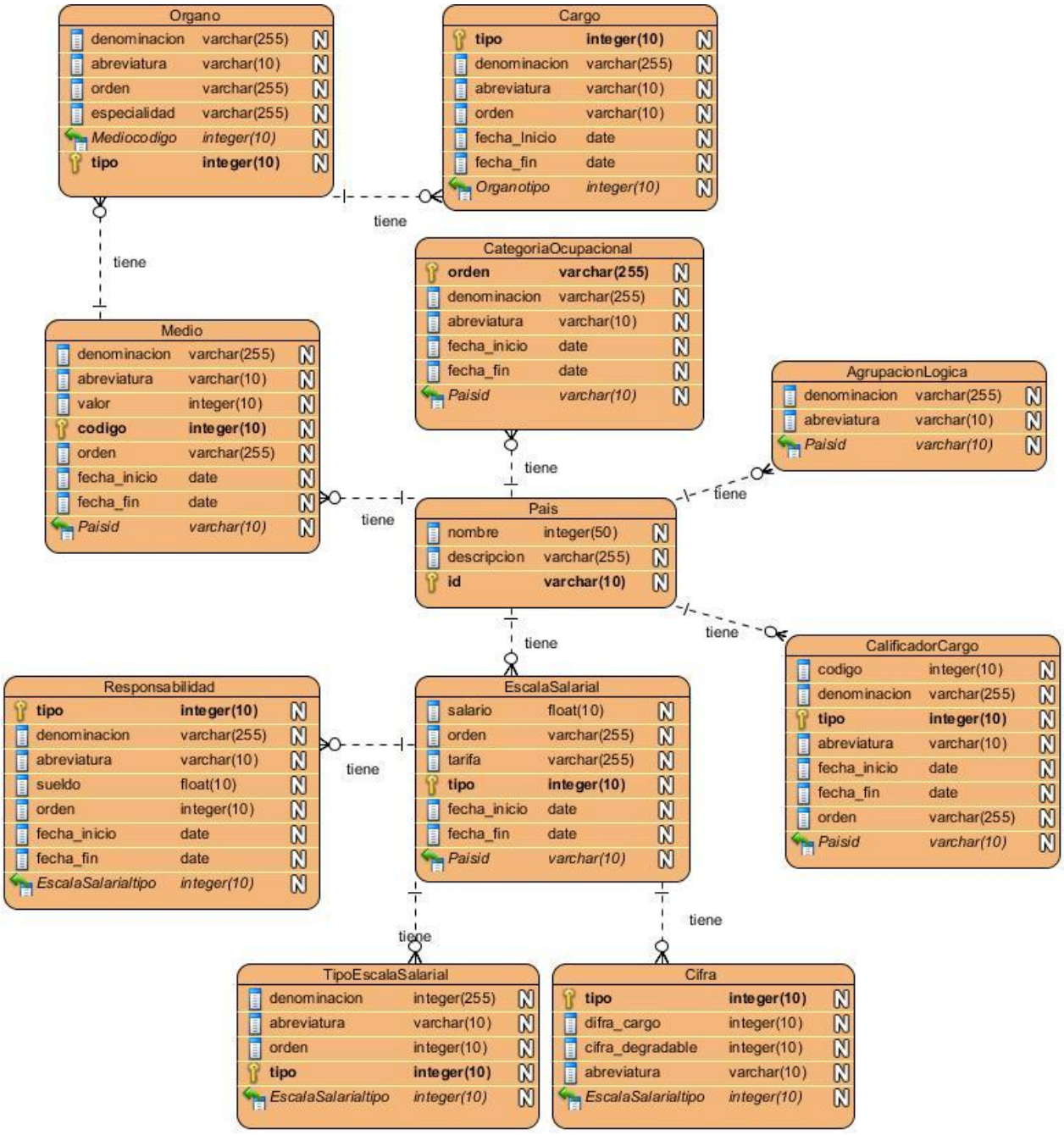
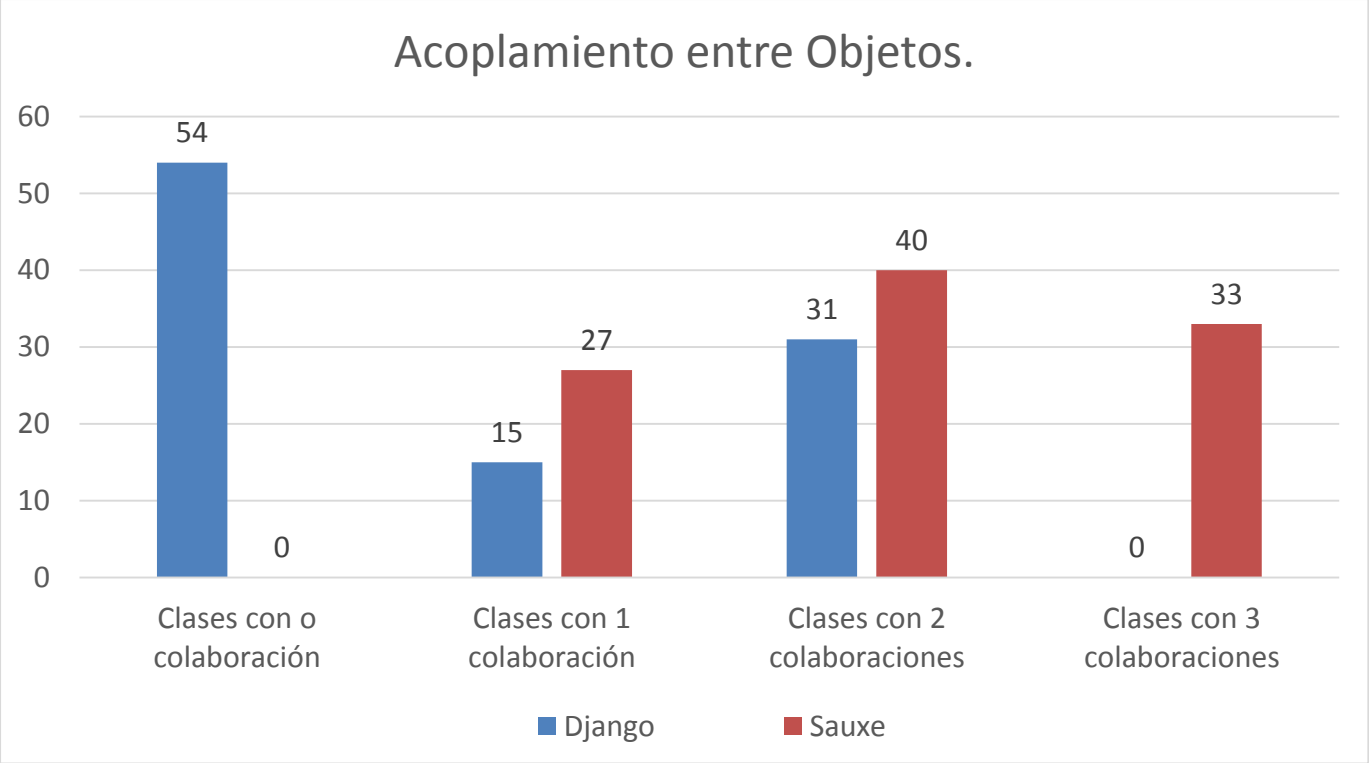
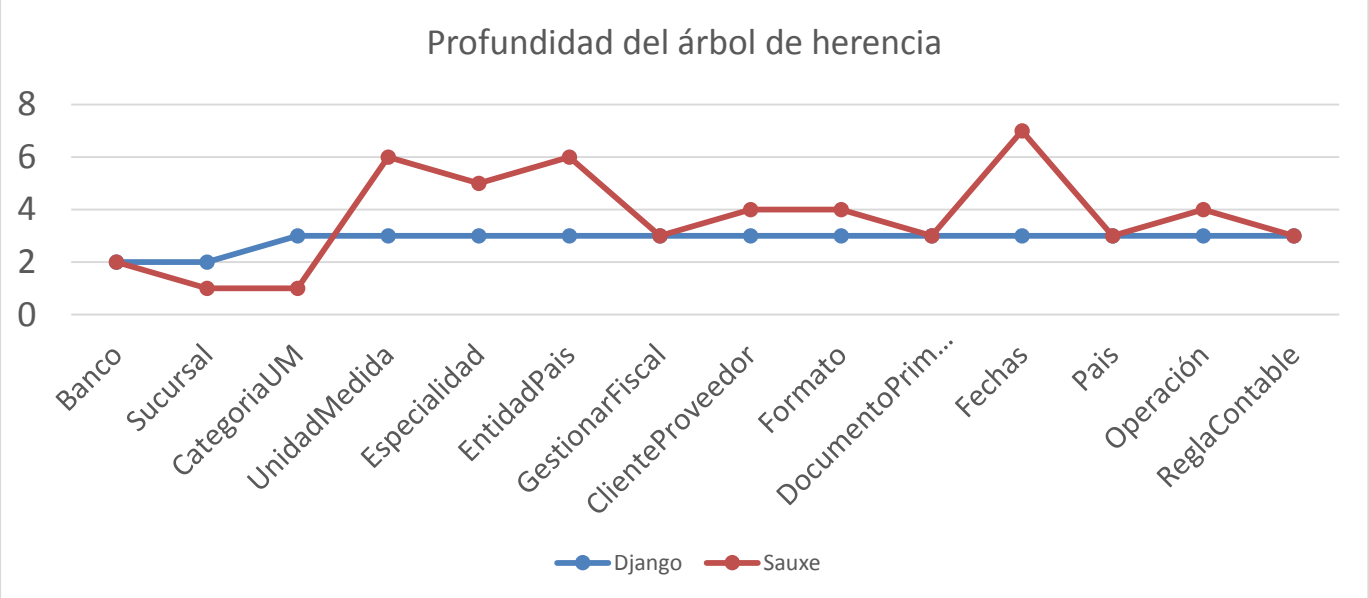


Figura 26.. Diagrama Entidad-Relación. Fuente: Elaboración propia.

Anexo 5. Gráfico de la aplicación de la métrica Acoplamiento entre Objetos evidenciado en las dos arquitecturas.



Anexo 6. Gráfico de la aplicación de la métrica Profundidad de árbol de herencia evidenciado en las dos arquitecturas.



Anexo 7. Informe # 1 de Cedrux.