UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Máster Universitario en Ingeniería y Tecnología de Sistemas Software

CARACTERIZACIÓN ONTOLOGICA DE HERRAMIENTAS DE DESARROLLO DE SOFTWARE BASADAS EN MODELOS CONCEPTUALES

Trabajo Fin de Máster

Autor: Jenny Susana Lazo Cajilima

Tutor: Óscar Pastor López

2017-2018

Agradecimiento

A mis padres que han trabajado duro para ofrecerme lo mejor, gracias por confiar en mí y motivarme siempre a la superación.

A mi familia que es lo más valioso que tengo, gracias por sus concejos y su ayuda.

A mis maestros y en especial a mi director de tesis Oscar, gracias por ser un soporte y guía en el desarrollo de este trabajo.

A Felipe que ha estado a mi lado cada día, gracias por tu compañía y por brindarme tu apoyo incondicional en todo momento.

Resumen

En un contexto de generación automática de código a partir de modelos conceptuales, el proyecto determina cuáles son las primitivas de modelado conceptual que deben estar presentes en una herramienta de generación de software a partir de dichos modelos conceptuales.

En primer lugar se determina el soporte ontológico, escenario para caracterizar dicho conjunto de primitivas conceptuales esenciales. El conjunto de primitivas planteadas permiten representar los distintos componentes y características que tiene un sistema software, independiente de la arquitectura que se seleccione posteriormente.

Para determinar el conjunto de primitivas, el trabajo se basa en el estudio realizado al método OO. En el cuál se establece cuatro modelos para representar la información: modelo de objetos, modelo funcional, modelo dinámico, modelo de presentación. El estudio analiza cada uno de los modelos en donde se representa un aspecto en particular del esquema conceptual.

Una vez presentada esta ontología fundacional de partida, se determina como dos herramientas usadas en la industria de desarrollo de software a partir de modelos conceptuales -WebRatio e Integranova- dan soporte en sus modelos conceptuales a dicha ontología. Para esto se determina cómo permite cada herramienta representar los elementos incluidos en el marco ontológico en su respectivo entorno de programación. Para ello se ha propuesto el desarrollo de un caso práctico (gestión de una nota de gasto) que permite implementar las primitivas necesarias para el estudio.

Hecho esto, se comparan sus expresividades para determinar en qué grado son ontológicamente robustas.

Palabras clave: Modelado Conceptual, desarrollo dirigido por modelos conceptuales, comparación de Herramientas MDD

Abstract

In a context of automatic code generation from conceptual models, the project will determine which are the conceptual modeling primitives that must be present in a software generation tool from these conceptual models.

In first place the ontological support will be determined, a scenario to characterize this set of essential conceptual primitives. The set of raised primitives allow to represent the different components and characteristics that a software system has, independent of the architecture that is selected later.

To determine the set of primitives, this document is based on the study performed on the OO method. In which four models are established to represent the information: object model, functional model, dynamic model, presentation model. The study analyzes each of the models where a particular aspect of the conceptual scheme is represented.

Once this foundational starting ontology is presented, it will be determined how two tools used in the software development industry from conceptual models -WebRatio and Integranova- give support in their conceptual models to this ontology. For this, it will be determined how each tool allows to represent the elements included in the ontological framework in its respective programming environment. To this end, the development of a practical case (management of an expense note) has been proposed, which will allow the implementation of the necessary primitives for the study.

This done, the expressivities will be compared to determine to what degree they are ontologically robust.

Keywords: Conceptual Model, Model-Driven Development, MDD Comparison

Contenido

Índice de Figuras

Índice de Tablas

CAPÍTULO 1

Introducción

El desarrollo de los sistemas de información ha tenido una constante mejora a través de los años, sin embargo, aún existen algunos problemas relacionados con el diseño e implementación, el objetivo de todos los estudios tienen un fin en común, y es: producir software de calidad.

Es por ello que surge el desarrollo de software dirigido por modelos que consiste en representar los sistemas de información por medio de su esquema conceptual y a través de varias transformaciones de varios de modelos conceptuales, es posible obtener el código final de la aplicación.

Los modelos conceptuales capturan la información relevante de lo que se quiere representar mediante la especificación de clases y atributos, sin embargo, la implementación de estos modelos genera algunos inconvenientes, existen ciertas restricciones al momento de la trasformación lo que dificulta el adoptar estas herramientas para proyectos considerables.

Sin embargo existen herramientas que han mejorado en esos aspectos con el afán de avanzar en la implementación de proyectos entre ellas están Integranova y WebRatio, que ofrecen un desarrollo dirigido por modelos de manera ágil.

Existe una serie de estudios comparativos entre el desarrollo con herramientas tradicionales y el desarrollo dirigido por modelos conceptuales, analizando su facilidad de uso, su capacidad, etc. pero con el avance de este último paradigma han surgido diversas herramientas lo suficientemente estables que permiten realizar una comparativa entre sí, como es el caso de WebRatio e Integranova.

Este trabajo propone una comparativa entre dos herramientas de desarrollo dirigido por modelos, Integranova y WebRatio, determinando si dan soporte a las primitivas necesarias que una herramienta debe tener. Todo esto se define mediante la realización de un ejemplo determinado y en cada caso se realiza un análisis.

1.1 Motivación

La ingeniería de software tiene como meta brindar producto de software de calidad y de alta productividad, sin embargo a pesar de las numerosas metodologías y soluciones presentadas siempre existen inconvenientes y problemas.

A lo largo del tiempo se han planteado una infinidad de lenguajes de programación: lenguajes estructurados, orientados a objetos, etc. y por otro lado también se ha propuesto un gran número de métodos de producción de software: basada en componentes, orientada a agentes, desarrollo ágil, ingeniería de requisitos, etc.

Existe un sinfín de tecnologías que ofrecen un desarrollo de software de calidad, sin embargo, ninguna ha llegado a su objetivo propuesto, ya que el desarrollo de software es un proceso complejo y con el pasar del tiempo los requisitos y necesidades van en aumento y con un constante cambio, sumándole a ello la aparición continua de nuevas tecnologías a las cuales debe adaptarse el producto, llevando así al ingeniero de software a dedicar su esfuerzo en conocer la tecnología en lugar de centrarse en la comprensión del problema, es por esto que se plantea las siguientes preguntas: modelado vs. programación; comprender y representar el problema vs. implementar la solución.

Es necesario primero conocer exactamente el producto software que se va a hacer para luego definir como lo se va a construir, es decir, primero se debe representar el esquema conceptual del sistema de información para luego pasar a la fase de implementación.

Esta ideología es la mejor manera de generar un sistema, pero el inconveniente con el que se encuentra hasta ahora es que no existen entornos de desarrollo suficientemente avanzados para construir las aplicaciones a partir de los modelos conceptuales planteados. Con este objetivo han ido evolucionando durante años las herramientas y entornos de programación, acercándose al usuario de una manera natural, considerando la existencia de dos actores fundamentales:

1- Partes interesadas, son los usuarios finales.
2- Ingenieros de software, son los encargados de construir el modelo a partir del conocimiento brindado por las partes interesadas.

El modelado conceptual no es un término que surge actualmente, este concepto se viene manejando desde los primeros métodos de producción de software, planteado con el objetivo de facilitar la especificación de los sistemas de información en términos de los interesados y ayudar a la conversión de la solución de forma organizada y precisa mediante el uso de técnicas de compilación de modelos. Obteniendo con esto una de las tareas complejas de la ingeniería de software que es la especificación de requisitos. Esta fase se relaciona con la calidad final del producto software porque aquí se describen los objetivos del sistema, si se presentan errores en esta fase se producirán problemas de fiabilidad, costo y robustez del producto final.

Es por esto que los desarrolladores requieren herramientas para realizar construcciones de alto nivel que permitan especificar aplicaciones mediante primitivas conceptuales que serán transformadas en el producto de software a través de un proceso de traducción, asociando cada primitiva conceptual con su representación de software correspondiente. Con esto se alcanzaría a generar software partiendo de esquemas conceptuales.

1.2 Planteamiento del Problema

Es necesario determinar el nivel de expresividad que tienen las herramientas dirigidas por modelos, puesto que sería de gran utilidad al momento de iniciar con el desarrollo de un sistema, por lo tanto existen preguntas que se requiere responder:

Pregunta 1: ¿Cuáles son las primitivas fundamentales para una herramienta MDD?

Pregunta 2: ¿Cómo las herramientas MDD actuales (Integranova y WebRatio) dan soporte a dichas primitivas conceptuales fundamentales?

En los siguientes capítulos se trata de dar respuesta a estas preguntas.

1.3 Objetivos

En un contexto de generación automática de código a partir de modelos conceptuales, el objetivo principal de este trabajo es determinar cuáles son las primitivas de modelado conceptual que deben estar presentes en una herramienta de generación de software a partir de dichos modelos conceptuales.

Para lograr el objetivo general, es necesario cumplir los siguientes objetivos específicos:

- Determinar el soporte ontológico, escenario para caracterizar dicho conjunto de primitivas conceptuales esenciales.

- Analizar como dos herramientas de desarrollo de software a partir de modelos conceptuales usadas en la industria (Integranova y WebRatio) dan soporte en sus modelos conceptuales a dicha ontología.

- Comparar sus expresividades

- Determinar en qué grado son ontológicamente robustas.

1.4 Estructura del documento

Este documento está estructurado en siete capítulos descritos brevemente a continuación:

- Capítulo 1, Introducción: El presente capitulo da una introducción del desarrollo del presente trabajo, la motivación que tiene, expone la problemática encontrada junto con sus respectivas preguntas de investigación, se indican los objetivos que se pretenden alcanzar y finalmente muestra la estructura del documento.

- Capítulo 2, fundamentos teóricos: En este capítulo se describe el desarrollo Dirigido por modelos, sus principales características, la metodología OO-Method y presenta una revisión sistemática de las propuestas para modelado conceptual.

- Capítulo 3, Primitivas del modelado conceptual: En este capítulo se plantean una serie de primitivas conceptuales que deben estar presentes en una herramienta de modelado conceptual.

- Capítulo 4, Caso Práctico 1: En este capítulo se utiliza la herramienta WebRatio para la realización de una aplicación ejemplo (gestión de una nota de gasto) y se evalúa como permite WebRatio representar los conceptos incluidos en el marco ontológico propuesto

- Capítulo 5, Caso Práctico 2: En este capítulo se utiliza la herramienta Integranova para la realización del mismo ejemplo (gestión de una nota de gasto) y se evalúa como permite Integranova representar los conceptos incluidos en el marco ontológico propuesto

- Capítulo 6, Comparativa: En este capítulo se determina el nivel de soporte de las primitivas planteada en cada una de las herramientas mediante una comparativa.

- Capítulo 7, Conclusiones: Finalmente se indican las conclusiones obtenidas en la realización del presente trabajo

CAPITULO 2

Fundamentos teóricos

Este capítulo presenta diversos conceptos relacionados con el desarrollo de software, centrándose en el desarrollo dirigido por modelos, con el fin de brindar una perspectiva sobre el desarrollo de software dirigido por modelos (MDD).

2.1 Introducción al desarrollo de software

A través de la historia del desarrollo de software el nivel de abstracción con el que se representan los sistemas de información ha ido en aumento, se inició con el lenguaje ensamblador y luego se atravesó por el lenguaje orientado a objetos, que facilitaba el proceso de elaboración de software. Sin embargo a pesar de todo este enfoque dirigido a la programación de los sistemas de información, no se ha llegado todavía a un resultado exitoso, es por esto, que en los últimos años surge una visión diferente para la producción de software, que consiste en el desarrollo basado en modelos conceptuales. Hoy en día las empresas están orientándose hacia este desarrollo para la producción de software.

Siguiendo las definiciones presentadas por Oscar Pastor (Pastor & Molina, Conceptual Modelling Primitives, 2007) que establece que para obtener un producto de software de calidad, la clave es el modelo, considerando que “el modelo es el código” en lugar de “el código es el modelo” en donde todo parte de una abstracción del dominio de un sistema que se transforma en un esquema conceptual, el cual, finalmente se convierte en el software desarrollado. Esto pretende que los desarrolladores de software ya no se preocupen en las características de los lenguajes de programación sino que su atención esté dirigida al análisis y diseño.

Como esta tendencia para el desarrollo de software se dirige a transformar modelos del mundo real en modelos que puedan ser interpretados y utilizados por las computadoras, el termino abstracción es el que cobra mayor fuerza en este estudio, puesto que se parte de una abstracción de conceptos claves para poder representarlos y transformarlos correctamente.

Es importante que los esquemas conceptuales que se construye a partir de los modelos mentales representen con precisión aquellos universales que participan en el sistema analizado, es decir, el esquema siempre debe ser la representación de los universales relevantes junto con sus relaciones, solo así se obtiene un esquema conceptual útil. Para ello es necesario la identificación de una ontología que defina los conceptos relevantes para especificar dichos universales.

Se requiere identificar las primitivas que son necesarias para la representación de los modelos mentales en un modelo generado por una herramienta MDD.

2.1.1 Nuevos entornos de desarrollo

La ingeniería de software está en un proceso de evolución continua debido a los requerimientos de los usuarios, así como de las tecnologías, por ello, nuevas disciplinas continúan desarrollándose para proporcionar información correcta y soluciones completas, pero con esto también se incrementa la complejidad de las tecnologías para el desarrollador y estas herramientas muchas veces no llegan a aumentar la calidad del producto.

Las herramientas actuales presentan dos problemas:

- Generalmente las aplicaciones se construyen a partir de código de un nivel más bajo al de la especificación del problema y no como debe ser, partiendo de un concepto metodológico bien definido en un alto nivel, abriendo así una brecha semántica entre el modelado y la programación.
  Esto se debe a que los entornos de programación no pueden representar directamente los modelos del mundo real construidos en la fase del modelado conceptual y porque los procesos y notaciones son diferentes.
- Existe mayor complejidad en la tecnología ya que las herramientas son especializadas y se centran en una sola parte del proceso, llevando a la integración de varias herramientas y vinculando el producto software directamente con la tecnología utilizada.

Los procesos de desarrollo actuales

Los procesos de desarrollo actuales se centran en la codificación, en la solución, en el "como"; en lugar de modelar el problema, el "qué". Esto afecta a la calidad del software desarrollado, resaltando algunos efectos negativos como:

- Los documentos realizados en la fase de análisis y diseño no son utilizados al momento de la implementación y al contrario se realiza la programación de manera manual para convertir los modelos en un producto de software. Los modelos quedan únicamente como documentación obsoleta que no evoluciona junto al producto por lo que el tiempo y dedicación empleados en la creación de los modelos es inútil
- Existen problemas de comunicación en el equipo de desarrolladores por las diferencias que hay en el proceso y los modelos realizados
- Por la complejidad de la tecnología es necesario grandes equipos de desarrollo con conocimientos técnicos, creando problemas de comunicación.
- Requiere mucho tiempo y esfuerzo obtener un prototipo funcional para entregar al usuario, por lo tanto el usuario evalúa el producto en etapas posteriores del ciclo de vida, aumentando el costo de los cambios solicitados, debido al estado avanzado en el que se encuentra el producto.

Necesidad de mejores entornos de desarrollo

Producir un sistema de información es costoso, lento, arriesgado e inseguro, todo esto debido a que la perspectiva de desarrollo no ha cambiado mucho en los últimos años, la parte esencial de construir

un sistema sigue siendo “programar”, las tecnologías presentan nuevas propuestas pero únicamente enfocadas en esta actividad.

Lo que se necesita ahora son nuevos enfoques de desarrollo para analizar el problema y tener un mejor nivel de abstracción, herramientas que permitan generar código a partir de modelos conceptuales realizados y entornos que brinden las siguientes características:

- Posibilidad de construir aplicaciones de calidad acorde a los requisitos del usuario
- Cubrir toda la producción del software de manera iterativa e incremental
- Ocultar la complejidad técnica proporcionando un conjunto completo y correcto de primitivas conceptuales
- Los constructores básicos deben tener un soporte formal para garantizar la coherencia sintáctica y semántica
- Permitir la especificación de un sistema independiente de la tecnología y de aspectos de programación

Si las herramientas proporcionan todas estas características se puede dejar atrás el desarrollo tradicional y ofrecer nuevas soluciones, pero por ahora las herramientas existentes tienen capacidades muy limitadas y no producen código funcionalmente equivalente al esquema conceptual planteado, y por otro lado, no cumplen con las características listadas anteriormente. Lo que se obtiene es un conjunto de plantillas que requieren tareas de programación manual.

2.2 Sistemas

El sistema no es algo absoluto sino una concepción de la realidad que lo realiza un observador del sistema (Pastor, Metodología de produccion automática de software, 1992), esto indica que las propiedades y características que se da al sistema dependen del observador en el momento que realiza la observación. Se establece cuatro acepciones distintas:

- Se puede referir a un sistema de proceso de datos creado en base a un sistema de gestión de base de datos.
- Como una abstracción de un sistema indiferente de la implementación.
- Como un conjunto de actividades realizadas en una organización para intercambiar y dar soporte a la información de la misma.
- Como una forma global de manipulación de datos y actividades de la organización.

Todo sistema tiene un entorno que comprende las cosas del mundo que no forman parte del mismo, pero que describen algunas de sus características propias. Al momento de Diseñar e implementar un Sistema se debe agrupar y organizar las cosas de una parte en concreta del mundo real y asignarles ciertas propiedades que describan dichas cosas.

Cuando se modela sistemas del mundo real, el enfoque está en determinar las propiedades más relevantes que lo caracterizan, así como en analizar y comprender los aspectos y comportamientos que lo definen. Pero no basta con su comprensión, hace falta definir buenas gramáticas que serán usadas al momento de describir los sistemas modelados.

Existen ciertas propiedades del mundo real que se deben manifestar de manera significativa al momento de modelar sistemas de información, para ello es necesario plantear diversos modelos que ayudan a la evaluación de las gramáticas que se utilizan.

2.2.1 Sistemas de Información

En el proceso de modelar un sistema de información pueden existir diferentes formas de percepción de la realidad que se tiene, es por esto que es necesario elegir un marco de referencia único con el fin de diseñar y desarrollar sistemas del mundo real que representen de forma precisa dichos aspectos.

Se concibe un sistema de información como un objeto que puede ser estudiado por derecho propio, independiente de la forma en que se implementa en su contexto organizacional y social y la tecnología utilizada para implementarlo. Es decir cuando se modela un sistema de información el enfoque se centra en definir las relaciones del sistema en el mundo real, sin embargo se deja de lado aspectos importantes como la organización, sus usuarios, la manera en la que lo utilizan, etc. Es importante la tecnología sobre el que se desarrolla y la implementación del sistema, pero no se debe considerar como una herramienta independiente, al incluir otros aspectos se puede tener un gran beneficio. (Weber, Spring, 1987)

Generalmente los sistemas de información comparten las siguientes características:

- Grandes volúmenes de datos almacenados en bases de datos estructuradas
- Grandes cantidades de líneas de código para proporcionar los datos y servicios
- Todos los usuarios deben recibir la correcta funcionalidad personalizada
- Todos los puntos anteriores deben estar interrelacionados
- La aplicación puede ser utilizada por una gran cantidad de usuarios distribuidos

Para integrar todas estas características en los productos software se requiere un conocimiento técnico en todas las áreas, con todo ese volumen de conocimiento requerido y todas las disciplinas necesarias que se deben integrar, obtener un software correcto, es un obstáculo muy grande.

Por lo tanto las características de un sistema de información se pueden separar en tres conjuntos:

- Estructura de superficie: se enfoca en la interfaz del sistema, la iteración con los usuarios, por ejemplo los informes que se producen.
- Estructura profunda: se refiere a las características del mundo real que se va a modelar, por ejemplo reglas definidas para contabilizar transacciones en un sistema contable.
- Estructura física: aquí se define la tecnología utilizada para implementar el sistema, por ejemplo el almacenamiento de la información, la comunicación y transmisión de datos.

Las características de la estructura profunda son las que más importancia tienen ya que la estructura de superficie y la física son cambiantes por el avance tecnológico, se pueden modificar en todo momento sin cambiar su estructura profunda que es el núcleo del sistema de información, es por ello que aquí se incorporan todas las características del mundo real que se va a modelar. Es necesario analizar hasta donde esas características se definirán en el modelo, puesto que existe el riesgo de expandirse demasiado y que el modelo que se genere sea demasiado complejo.

2.3 El enfoque FRISCO

Este enfoque asume que el universo está formado por “cosas” y las personas son parte de ello, todo lo que se pueda ver, sentir, tocar o pensar es considerado como una “cosa” (Lindgreen, Mayo 1990). Las cosas que forman parte del mundo tienen propiedades que las describen, son aspectos característicos de cada una. Por esto es necesario una forma consistente de la descripción global puesto que se puede caer en una recursión infinita si se consideran las propiedades como cosas.

Al momento de analizar una parte específica del mundo real se puede definirla como un área que delimita el análisis, esta área está formada por dos elementos: objetos que son la representación de cosas concretas que se puede percibir con nuestros sentidos y conceptos que representan las cosas abstractas del mundo; los elementos de un área pueden estar incluidos el uno en el otro o ser distintos.

Otra forma de clasificar las cosas es por su estado en base al tiempo de existencia, es necesario que una cosa exista en el mundo real, para poder describirla como existente de manera objetiva en el modelo. De lo contrario cuando aparece de manera mental su existencia entonces sería subjetiva. Adicionalmente se debe definir el tiempo de vida de las cosas: unas pueden existir por un instante determinado de tiempo y luego ser destruidas y otras pueden permanecer a lo largo de todo el modelo

A partir de esto, es posible definir las cosas dinámicas y estáticas. Las dinámicas son cambiantes en el tiempo mientras que las otras no sufren ningún cambio, esto depende de algunos aspectos como: las propiedades que definen el estado de la cosa y el intervalo de tiempo en el que las propiedades son observadas.

Para que una cosa sufra cierto cambio o sea creada tiene que pasar por un proceso compuesto por operaciones, que son las encargadas de la modificación de las propiedades de las cosas. Dependiendo del nivel de abstracción que se utilice en ocasiones se definen las operaciones como procesos.

Perspectiva OO del enfoque FRISCO

Para obtener una especificación formal de un sistema del mundo real se considera que está compuesto por un conjunto de cosas que interaccionan entre sí, que se le llama objetos. Estos tienen atributos que describen las propiedades de cada uno, estas propiedades van a ser determinadas por el analista que es el encargado de incluirlas en el modelo dependiendo del propósito de éste.

De igual manera los objetos tienen un estado que indica el momento en el que se crea o destruye, también cuenta con un identificador y sus atributos que van a ser constantes o variables.

Para que un objeto cambie su estado es necesario que ocurra un evento, y esto puede suceder varias veces a lo largo de toda la vida del objeto. Dichos eventos son activados por agentes responsables de su ejecución y pueden producirse de dos maneras como se indica en (Pastor López, 1992):

- Un evento compartido con varios objetos
- O por una actividad introducida que se genera luego de que el objeto cumpla con ciertas condiciones

A través del proceso de clasificación se seleccionan los objetos con las mismas propiedades, se agrupan, y se define una clase, que es una caracterización de un conjunto de propiedades y comportamientos que comparte un conjunto de objetos.

Es posible definir como esquema conceptual como un conjunto de descripciones de objetos que componen un sistema junto con sus clases, atributos y eventos.

2.4 Modelado orientado a objetos

Para definir un método de desarrollo basado en un marco conceptual es necesario introducir una ontología que caracterice los conceptos básicos sobre los cuales se construye el método deseado. Para esto es necesario seleccionar el modelo más apropiado, que en el contexto de este trabajo va a ser el modelo orientado a objetos.

El método es una forma de trabajo que obtiene un resultado deseado, que puede ser la especificación, la implementación de un producto, etc. Existen dos enfoques para determinar los métodos tradicionales:

- Métodos orientados al proceso, que se centran en la descripción de los eventos
- Métodos orientados a datos, que se centran en la descripción de la arquitectura de las entidades dentro del sistema

Las técnicas de análisis y diseño emplean la primera técnica para el proceso de arquitectura desde una perspectiva dinámica y la segunda para la arquitectura de datos con una perspectiva estática, cada uno con su propia notación.

La realidad es que los sistemas no se componen de datos o procesos tratados de forma separada, lo estático y dinámico deben verse como un todo de manera homogénea, teniendo como problema encontrar un método para respetar este principio de homogeneidad, que no utilice conceptos y notaciones diferentes para datos y procesos. (Pastor & Molina, Object-Oriented Modelling as the Starting Point, 2007) Es por ello que se define al modelo orientado a objetos como un enfoque ideal, ya que captura la realidad tal como es, no hay datos o procesos que existan independientemente, todo forma un conjunto. La principal contribución del paradigma es la del objeto que encapsula adecuadamente la perspectiva estática y dinámica para el análisis y diseño de sistemas de información.

El enfoque orientado a objetos presenta las siguientes características:

- Facilita el modelado conceptual por medio de la noción de objeto que permite representar la realidad tal como es.
- Reduce la complejidad del sistema, manteniendo la noción de objeto durante todo el proceso de producción.
- Ayuda a la comunicación de las partes durante la fase de modelado conceptual, ya que el objeto puede ser comprendido tanto por ingenieros o por las partes interesadas.
- Mejora las relaciones entre los elementos ya que se ve al sistema como un conjunto de objetos que interactúan entre sí.

- Enfatiza datos y procesos cuando se especifican objetos como unidades de ejecución autónomas.

2.4.1 Aspectos clave

Existe un conjunto de aspectos a tomar en cuenta antes de la construcción del software orientado a objetos para satisfacer los requisitos de manera efectiva y son:

- Es necesario definir patrones que se encuentren en las especificaciones del sistema, que llegarán a ser las unidades básicas en la construcción del esquema conceptual. Es necesario capturarlos y representarlos como un catálogo de patrones conceptuales.
- Luego se debe enriquecer el modelo de objetos, seleccionando los patrones relevantes, se puede emplear para ello una notación estándar como UML.
- El formato de la especificación se debe realizar utilizando un lenguaje de especificación formal para así tener una caracterización precisa de la ontología asociada al modelo orientado al objeto elegido.
- Un problema importante es la complejidad de UML por la gran cantidad de conceptos para todos sus diagramas, pero como es una notación unificada evita “reinventar la rueda” al asignar representaciones gráficas a los conceptos de la metodología propuesta.
- Es necesario garantizar un producto de software de calidad y para ello es necesario la utilización de un conjunto adecuado de patrones arquitectónicos y de diseño eficaces para la resolución de problemas.

Catalogación de patrones conceptuales

Es necesario tener un conjunto de conceptos para describir de forma correcta algo. Este conjunto debe tener los conceptos necesarios, ni más ni menos, ya que si faltan se da una menor expresividad en los patrones conceptuales y si se excede puede producirse una sobre especificación.

El esquema conceptual realizado debe representar los requisitos expresados por las partes interesadas, partiendo de un conjunto de patrones establecidos para representar las abstracciones y sus relaciones de manera precisa y clara. Estas abstracciones permiten constituir la ontología orientada a objetos.

La mayoría de los modelos orientados a objetos no capturan en su totalidad y de la forma adecuada toda la semántica que requiere un sistema de información. Para esto es necesario enriquecer la expresividad del modelo hasta convertirlo en un modelo completo con los conceptos necesarios.

La semántica de clases

El primer paso para el modelado de requisitos es identificar las clases que deben estar presentes en el modelo, posteriormente establecer los atributos que definen el estado de los objetos y finalmente definir los servicios necesarios para modificar dichos estados. Sin embargo, no basta con especificar los atributos, es necesario también definir las restricciones de valor que tiene cada atributo para evitar servicios de ejecución incorrectos.

Para describir completamente la funcionalidad de los objetos es necesario la especificación de la manera en que los servicios modifican los atributos de dicho objeto. Se puede definir también atributos derivados que sirven para realizar cálculos partiendo de valores de otros atributos.

Con todos los detalles anteriores es posible indicar que es necesario primero establecer un conjunto de primitivas para la especificación completa de un sistema de información. Dichas primitivas permiten realizar especificaciones correctas sin ambigüedad y sin deficiencias.

La semántica de las relaciones de clase

Todo método de desarrollo de software debe incluir un factor importante como es el análisis de las relaciones entre las clases que proporciona. Si la semántica proporcionada para estos casos es ambigua y depende del desarrollador, entonces como resultado se obtiene un producto software impredecible. Es necesario que las relaciones de generalización, asociación y agregación, estén claramente definidas; aquí surge un nuevo concepto que es la cardinalidad y herencia.

Integración de técnicas formales y semiformales

La ontología del modelo se compone de los conceptos determinados con un lenguaje de especificación formal, sin embargo, puede ser necesario también notaciones semiformales para la construcción de los modelos conceptuales. Los enfoques semiformales pueden utilizar técnicas de metamodelado, es recomendable combinar los enfoques formales con los semiformales para lograr un enfoque razonable.

Los enfoques formales ayudan a eliminar ambigüedades en etapas tempranas del desarrollo del software y las herramientas permiten verificar las propiedades formales en la fase de modelado. También permiten trasformar cada primitiva del producto de software final a su equivalente en la especificación original.

Para lograr una unificación, la gramática definida debe combinar patrones conceptuales con elementos del lenguaje de especificación a los que representa. Esta correspondencia viene de construir la notación en el caso de no existir o en su defecto adoptar una notación gráfica del UML estándar que ya existe.

Arquitectura del software: Patrones arquitectónicos

Es necesario emplear gran esfuerzo en la fase de modelado, ya que el objetivo es obtener un esquema conceptual que represente correctamente cada uno de los requisitos de un sistema de software. Por lo tanto la calidad del producto software construido depende de la precisión con la que se definen los patrones conceptuales empleados para construir el esquema conceptual.

Luego de definir estos patrones se procede a determinar los componentes de software que se van a construir. Para ello se debe seleccionar la arquitectura de software que determina los componentes y la manera en la que se relacionan. Hoy en día la arquitectura de tres capas es la más utilizada para el desarrollo de aplicaciones, esta arquitectura presenta tres niveles:

- La capa de presentación, hace referencia a la interfaz de la aplicación, es la encargada de interactuar con el usuario.
- La capa de aplicación, implementa toda la funcionalidad de la aplicación

- La capa de persistencia, asegura la persistencia de los datos utilizados para la aplicación

Capa de presentación

Aquí se definen las interfaces gráficas del sistema, la representación de la aplicación hacia los usuarios finales para que puedan acceder a los datos y controlar la funcionalidad .Existen herramientas de diseño que ayudan en el diseño de la interfaz, los diseñadores son los encargados de construir las interfaces para que luego los programadores realicen la interacción con los usuario y los datos mediante la capa de aplicación.

Uno de los principales inconvenientes en esta capa surge cuando los diseñadores no construyen interfaces acorde al modelo de objetos, entonces se tendría que controlar estas inconsistencias mediante programación adicional. Lo ideal sería construir la interfaz de usuario directamente del modelo de objetos, pero a su vez esto limitaría el uso de herramientas de diseño. Para resolver este problema se debería construir un modelo de presentación que incluya toda la interacción Persona-Ordenador.

Capa de aplicación

Esta capa determina la funcionalidad del sistema que se especifica en el esquema conceptual original, por lo tanto aquí se implementa el comportamiento de las clases que se definen en la fase del modelado. Para garantizar la calidad del producto software es necesario que el comportamiento sea el mismo que se define en el esquema conceptual. Esto implica que cada servicio especificado en la fase del modelado debe estar presente en la capa de aplicación.

Se debe garantizar que todos los comportamientos definidos en la especificación del problema se encuentren definidos en esta capa.

Capa de persistencia

Una vez definidos los componentes en las capas anteriores es necesario establecer los componentes que se encargarán de gestionar la persistencia de objetos en el dispositivo de almacenamiento de datos, también se define el acceso a los servidores y la administración de las bases de datos.

Un mecanismo de implementación común es generar un esquema de base de datos partiendo del modelo de objetos creado en la fase de modelado, en donde las clases se representan como tablas y los atributos como campos de la tabla. En definitiva cada relación se podrá representar adecuadamente.

Para la ejecución de los servicios en necesario transportar los datos de la capa de persistencia a la capa de aplicación y viceversa.

Comunicación entre capas

La conexión entre capas está directamente asociada a las plataformas tecnológicas y es una característica esencial entre los componentes del producto software, todas las capas deben estar vinculadas y esa comunicación requiere ciertos protocolos para estandarizar el tránsito de la información entre capas.

2.4.2 Método OO

A lo largo de los años se han propuesto numerosos enfoques con el afán de llegar a producir software de calidad, y la primera decisión es la elección del método que se va a usar para el modelado conceptual, esos enfoques se dividen en dos categorías:

- Análisis y diseño estructurado
- Análisis y diseño orientado a objetos

Los métodos estructurados se basan en la programación de procedimientos a nivel de modelado y representan los flujos de información por medio de diagramas de flujo de datos. Muchos metodólogos pusieron su enfoque en este paradigma llegando a proponer los diagramas de estado para indicar el comportamiento y enlazarlo a cada funcionalidad.

Statemate (Naamad, 1996) es la primera herramienta del mercado que permite generar código a partir de modelos conceptuales. Al ser la primera en surgir, sus capacidades de generación son reducidas, sin embargo es el punto de inicio para proyectos más ambiciosos.

En cuanto a los métodos Orientados a Objetos surgen a partir de 1980, estos permiten una mayor abstracción para la modelización, el concepto de objeto se convierte en algo útil al momento de realizar el modelado conceptual. Se emplean los diagramas de clase para modelar los sistemas de información, estos diagramas incluyen propiedades estáticas y dinámicas. Para indicar el comportamiento de estas clases se agrega el diagrama de transición de estado, que describe la creación, destrucción y modificación de las clases.

Todas las herramientas que generan código a partir de modelos conceptuales tienen un problema en común y es que las primitivas conceptuales del modelado presentan ambigüedad, para solucionar esto es necesario que cada primitiva esté asociada a una representación correspondiente en el software para que el compilador del modelo pueda implementar los patrones conceptuales en las representaciones de software a ellos asociadas.

Un sistema de información está basado en modelos cuando se obtiene automáticamente del esquema conceptual construido inicialmente, este esquema contiene todas las propiedades estáticas y dinámicas del sistema modelado.

2.4.3 Características generales del método OO

Hasta ahora se ha indicado que desarrollar software a partir de modelos conceptuales puede garantizar una generación de código ágil y fácil y así obtener productos de calidad. Si este proceso de generación se automatiza, se alcanzaría un gran avance en la industria del software.

Oscar Pastor en su libro (Pastor & Molina, The OO-Method, 2007), propone un método denominado OO-Method, que se basa la construcción de un esquema conceptual orientado a objetos y su representación en un entorno de desarrollo comercial para obtener un producto de software de calidad. Esta propuesta tiene los siguientes principios:

- Establecer las nociones de modelado conceptual orientado a objetos en las que se definan de forma precisa todas las primitivas conceptuales.

- Integrar los métodos formales para una aceptación industrial.
- Obtener un entorno de producción de software completo que permita representar todos los aspectos definidos en el esquema conceptual.

Además, en la propuesta se incluyen dos fases:

1- Modelado conceptual: en esta fase se extraen las propiedades esenciales del sistema de información y se representan, formando el esquema conceptual. Los patrones utilizados deben tener un soporte formal en todo momento, esta especificación formal orientada a objetos se encuentra escrita en el lenguaje OASIS. Este paso representa la conversión de la notación gráfica de UML en sintaxis formal del lenguaje OASIS.

2- Generación del producto de software: aquí se determina la representación que tiene en el producto software cada patrón conceptual definido en el esquema conceptual, mediante un compilador de esquemas conceptuales y la correspondencia establecida para cada patrón, se generará automáticamente el código y el producto software, que debe incluir todos los aspectos estáticos y dinámicos incluidos en el modelo conceptual.

2.5 Aproximaciones ontológicas

Una aproximación interesante es la ontológica que trata de la existencia de las cosas en el mundo real, que son concebidas como objetos al momento de modelar. Es una representación del conocimiento de un dominio específico; utilizando una visión ontológica, estas representaciones se definen independientemente de la implementación y tienen una distinción explícita entre objetos y propiedades. Una ontología se construye en relación a un contexto de utilización en el cuál se especifica una conceptualización que puede ser interpretada por una computadora o por el ser humano.

Las ontologías proporcionan un vocabulario común para determinar el significado de los términos y las relaciones entre ellos. Generalmente se interpreta usando cinco tipos de componentes:

- Conceptos: Se puede utilizar el término clase o concepto, y se interpreta como la descripción de una tarea, acción, etc.
- Relaciones: Hace referencia a la interacción entre los conceptos definidos en el dominio.
- Funciones: Se trata de una relación de tipo especial en el que el n-ésimo elemento de la relación es único para los n-1 precedentes.
- Axiomas: Indican que las expresiones pueden ser tomadas siempre como ciertas.
- Instancias: Se utiliza para representar un elemento específico de la ontología.

Las ontologías requieren de un lenguaje lógico y formal para ser expresadas y generalmente su definición es un proceso iterativo, es decir su definición inicial va evolucionando y mejorando a través de una serie de iteraciones.

2.5.1 Equivalencias

La estructura básica de modelización de un dominio de sistema son los objetos por lo tanto se puede aplicar claramente un concepto ontológico que es el encargado de estudiar la modelización de la

existencia de las cosas en el mundo debido a que abordan el mismo tema. Esto lleva a la noción de que al momento de modelar un sistema es más fácil describirlo si se percibe a través de la noción de objetos.

A continuación se establece relaciones entre los conceptos OO y los conceptos ontológicos para determinar principios básicos del modelo objetual.

- Abstracción de datos o encapsulamiento

  En el modelo OO se centra en el comportamiento del objeto despreocupándose de su representación, una aproximación ontológica es definir cambios en los objetos y estos no pueden ser independientes de dichos objetos.

- Independencia o persistencia

  Hace referencia al estado del objeto y su existencia, un objeto puede cambiar su estado únicamente por medio de acciones producidas por sus métodos. Los estados son las propiedades de un objeto por lo que cubre el concepto de independencia.

- Comunicación entre objetos

  Los objetos intercambian mensajes para comunicarse entre sí. En cuanto a la base ontológica la interacción se manifiesta por medio del ciclo de vida del objeto y por los estados que el objeto va tomando a través de él.

- Herencia

  De acuerdo a sus propiedades los objetos se agrupan en clases y en subclases teniendo en cuenta que las subclases heredan las variables y métodos de la clase padre a la que pertenecen. Desde el punto de vista ontológico se define las jerarquías de la clase mediante el alcance de un conjunto determinado de las propiedades.

- Homogeneidad

  Se considera todo el modelo como un objeto. Desde la perspectiva de la ontología existe una separación clara entre los objetos y sus propiedades, por lo tanto no existe una homogeneidad completa, además existe la posibilidad de que un objeto se componga de otro pero existen objetos elementales que no pueden componerse.

2.5.2 Aspectos estáticos

Según el modelo presentado por Wand (Weber, Spring, 1987) el mundo está compuesto por objetos con propiedades que los describen. Los objetos pueden tener o no propiedades independientemente si se las conoce o no, a pesar de esto el observador asigna atributos a los objetos que representan a ciertas propiedades que el observador quiere representar para ciertos objetos.

Las propiedades no se pueden representar solas sino siempre mediante un objeto y pueden ser de dos tipos:

- Atributos observables

Los valores asignados a los atributos pueden tener restricciones sobre los posibles valores que puede tomar un objeto, hace referencia a un solo elemento de un tipo.

- Leyes
  Son propiedades a las que se les asigna valores específicos, es decir, un objeto posee una propiedad determinada, afectan a todo un tipo, es decir, universalmente.

2.5.3 Aspectos dinámicos

Para determinar los estados válidos de un objeto se plantea un conjunto de restricciones, entonces un objeto solo puede estar en un estado válido y si en algún instante llega a un estado inválido debe cambiar espontáneamente a un estado válido, este cambio espontaneo se produce por medio de las transacciones.

Las leyes relacionan variables de estado y de esta manera los objetos interaccionan con el modelo. Teniendo en cuenta que un objeto puede afectar al estado de otro, existen dos tipos de leyes: Intraobjetuales y interobjetuales.

La transición de un estado a otro se denomina evento, por lo tanto se describe un evento como un cambio de estado, que lleva al objeto de un estado inicial a otro final.

En el modelo otológico se describe la interacción entre objetos de la siguiente forma: Un objeto i afecta a otro objeto j si el estado de i es relevante para las transiciones de estado de j; para que los dos objetos interaccionen las leyes de sus transacciones deben tener al menos una variable de estado en común.

CAPITULO 3

Primitivas del modelado conceptual

Se presenta un análisis de las primitivas necesarias para el desarrollo dirigido por modelos, divididas en tres dimensiones: datos, control y procesos. Para ello se define 4 modelos (como vistas parciales de un modelo conceptual global) que se detallan a lo largo de este capítulo.

3.1 Primitivas

Es necesario que los métodos de desarrollo de software orientado a objetos proporcionen primitivas para la especificación de los diferentes componentes de un sistema software. Estas primitivas deben representar patrones del sistema con características independientes de la arquitectura elegida.

Para determinar las primitivas necesarias, este estudio se basa en el método presentado en el libro de Oscar Pastor (Pastor & Molina, Conceptual Modelling Primitives, 2007). El cuál define un proceso para generar productos software y abarca todas las fases de desarrollo, que van desde la toma de requisitos hasta la elaboración del modelo conceptual respectivo. El principal enfoque de este método es la generación automática de código del producto software que a partir de la definición de su esquema conceptual y por medio de un compilador de modelos conceptuales, transforma los constructores conceptuales en una representación específica en el software.

El método OO incluye las siguientes características:

- Cuenta con una notación gráfica basada en un estándar reconocido como es UML (UML, 2004) por lo que resulta fácil su utilización. Utiliza mecanismos de extensión para cubrir las características que no proporciona UML.
- Para el lenguaje de especificación formal emplea OASIS, que proporciona un conjunto de especificaciones de los sistemas de información para tratar al esquema conceptual gráfico que se crea. Produciendo un sistema de alto nivel.
- Finalmente se obtiene un modelo de ejecución, que es la representación del modelo conceptual en un entorno de desarrollo de software.

Una vez definidos los patrones conceptuales, cada elemento debe tener su propia representación en el entorno de programación seleccionado. Estas representaciones se basan en patrones de diseño conocidos como el control de acceso, interfaz de usuario y persistencia de objetos, a partir de allí la implementación será una transformación con las respectivas correspondencias.

Esta representación de las primitivas en herramientas específicas (WebRatio, Integranova) se analiza con cada uno de los modelos posteriores y permite formular una comparativa entre dichas herramientas. Actividad central de este trabajo de I+D.

3.2 Modelado conceptual

El modelado conceptual permite representar un sistema de información de manera gráfica, identificando así sus componentes y definiendo su funcionamiento, reduciendo los errores y fallas en su posterior implementación.

Es por esto que en esta fase se detallan los componentes estáticos y dinámicos del grupo de objetos que representan el sistema, sin tener en cuenta su posterior implementación, para ello se divide a los sistemas en tres dimensiones: **datos, control y proceso**.

Es necesario que el sistema represente la interacción visto como una sociedad de objetos interactivos. Por lo tanto su estructura debe contener una representación adecuada de los objetos y sus relaciones, para esto existen varios métodos que proporcionan técnicas para cubrir las cuatro áreas previamente establecidas. OO-Method establece cuatro modelos para representar toda esa información, cada modelo representa un aspecto en particular del esquema conceptual y son los siguientes:

- Modelo de Objetos, detalla la estructura de las clases que se han definido en el dominio del problema junto con sus relaciones estáticas.
- Modelo dinámico, detalla los servicios de los objetos de la clase y su interacción con otros objetos.
- Modelo funcional, captura los cambios de estado que presenta un objeto que se producen por la ejecución de eventos.
- Modelo de presentación, amplia el modelado conceptual convencional mediante la incorporación de la información relacionada a la interfaz de usuario y la forma en la que los usuarios interactúan con el sistema.

Los tres primeros modelos incluyen los datos requeridos para desarrollar cualquier esquema conceptual pero el cuarto modelo que es el que detalla la interacción del sistema con los usuarios y pasa a ser un componente fundamental para la modelización. Esta interacción se captura mediante patrones de interfaz, sin embargo, esto no determina la interfaz final del sistema ya que dependerá de la implementación particular del producto final.

A continuación se detalla cada uno de estos modelos para determinar las primitivas conceptuales fundamentales que una herramienta MDD debe tener.

3.3 Modelo de objetos

Este modelo utiliza clases y relaciones para describir la estructura del sistema y dar una visión de la arquitectura que tiene. Se considera una forma de pensar de modo abstracto utilizando para ello como base conceptos del mundo real y no conceptos de computadoras. Se define como unidad básica **el objeto** que tiene una estructura de datos con comportamientos y relaciones, para plasmar este concepto se plantea como principales primitivas las clases y la relación entre ellas.

3.3.1 Clases

Es una abstracción del mundo real que se compone de atributos que son propiedades que representan aspectos estructurales característicos del objeto, se establecen solo las propiedades importantes para el problema a resolver y se descarta el resto, también se compone de eventos que representan aspectos de comportamiento, se debe incluir eventos de creación y destrucción.

Una clase agrupa un conjunto de objetos con características similares, mediante la cual se especifica su comportamiento y se determina la estructura que tiene junto con sus servicios.

Todas las clases deben definirse en base a dominios, que son un conjunto de valores y un conjunto de operaciones sobre dichos valores, estos llegan a ser los tipos asignados a los atributos, por ejemplo el Bool (true, false).

Atributos

Los atributos se ubican en el dominio de datos y son las características en una clase que distinguen a un objeto de otro por lo tanto toman un valor para cada instancia de la clase definiendo el estado del objeto y sus cualidades. Entre las principales propiedades que tiene un atributo están: nombre, tipo de dato, tamaño, valor predeterminado, si es requerido o no y un indicador si puede ser nulo.

Como propiedades adicionales están: alias y comentarios. El nombre del atributo debe ser único y no debe contener caracteres especiales. Sin embargo se puede utilizar el alias para mejorar la descripción del atributo.

En cuanto al tipo, no se hace referencia al tipo de dato sino al tipo de atributo que puede ser: constante, variable o derivado.

- Constante. Una vez que el objeto ha sido creado, los valores de los atributos constantes permanecen sin cambios hasta que se destruya el objeto.
- Variable. Los valores de estos atributos cambian a lo largo del tiempo, estos cambios se producen por medio de servicios que causan la modificación del valor del atributo.
- Derivado. Estos atributos tienen un valor que se calcula a partir de otros atributos, las fórmulas de derivación deben ser especificadas.

Servicios

El comportamiento de un objeto de una clase se relaciona directamente con la funcionalidad que tiene por lo tanto aquí se establecen las tareas que puede realizar un objeto.

Los servicios definen el comportamiento de una clase, por lo tanto se ubican en el dominio de control, un objeto se modifica siempre y cuando se produzca un servicio. Se puede considerar como una unidad de procesamiento ya sea atómica cuando se trata de un evento o molecular cuando es una transacción local.

Los servicios tienen un nombre que debe ser único dentro de una misma clase y puede tener adicionalmente un alias y comentarios. También se utilizan para crear y destruir objetos de una clase. Los servicios están compuestos por eventos, transacciones locales y operaciones.

Eventos

Representan un cambio de estado cuando ocurre de manera instantánea, se distinguen los siguientes tipos:

- Creación (New): Es el primer evento en la vida de un objeto, cuando un evento de creación se activa, se valida todos los valores iniciales a los atributos que se encuentran definidos como derivados, no se tiene en cuenta los que están marcados como constantes o variables.
- Destrucción (Destroy). Este evento es el encargado de destruir un objeto.
- Propio: Cuando solo afecta a un objeto.
- Compartido: Cuando afecta a más un objeto.

Argumentos

Los argumentos representan las propiedades básicas de los servicios, por lo tanto todos los servicios con excepción de los de creación llevan un argumento implícito que hace referencia a la clase del servicio. Los argumentos tienen propiedades como: nombre, tipo de datos, tamaño, indicador si acepta valores nulos, entre otros. El nombre del argumento debe ser único.

Se puede definir dos tipos de argumentos:

- Argumentos de entrada: su valor es determinado antes de ejecutar el servicio.
- Argumentos de salida: su valor se asigna luego de que el servicio se ejecuta.

La expresión de valor de un argumento de salida es un par <Condición, Efecto>.

Transacciones

Se puede definir como una unidad de ejecución compuesta por servicios. Presenta las siguientes propiedades:

- Sin visibilidad de estados intermedios. Cuando los estados de los múltiples servicios que componen una transacción no son visibles desde afuera.
- Ejecución “todo o nada”. Si la ejecución de una transacción ha iniciado entonces continúa hasta terminar: en caso contrario inicia nuevamente desde el principio.

Se puede establecer transacciones locales que únicamente afectaran a la propia clase y sus relaciones y transacciones globales que afectan a clases de todo el sistema.

Operaciones

En ciertos casos no se requiere que cuando falla la transacción regrese al estado inicial, cancelando todo lo que ha hecho, es por esto que aparecen las operaciones que se consideran unidades de ejecución molecular no transaccionales (Pastor & Molina, Conceptual Modelling Primitives, 2007).

Puesto que las operaciones son similares a las transacciones, se incluye todas las especificaciones dadas en las transacciones con una diferencia: en las operaciones los dos requisitos transaccionales básicos no se satisfacen.

Restricciones de integridad

Los atributos de un objeto pueden tomar ciertos valores o no, eso depende de las propiedades que deban cumplirse en los estados del objeto. Al momento de definir la clase con sus atributos y servicios también se deben definir las restricciones que va a tener, estas restricciones son condiciones que se deben cumplir en cada estado de un objeto, generalmente se usa la palabra "constraint" para describir las restricciones de integridad dentro de una clase. Se distinguen los siguientes tipos:

Estático

Estas fórmulas se mantienen en todos los estados que dure el objeto a pesar de que su evaluación se la realiza en un solo estado, para construir la fórmula se pueden utilizar los valores de los atributos, constantes y funciones visibles desde la clase y los valores se conectan con operadores lógicos AND y OR.

Dinámico

Estas fórmulas a diferencia de las anteriores involucran a más de un estado y requieren operadores temporales. Por lo tanto se enriquece la expresividad de las restricciones estáticas y son más utilizadas para condiciones sobre el futuro del objeto o sobre un estado pasado.

Pre-condiciones

A los servicios que se definen en las clases se pueden agregar precondiciones, esto quiere decir que son condiciones o requisitos que deben cumplirse para que dicho servicio se pueda ejecutar. Se define mediante una fórmula booleana que utiliza constantes, funciones y atributos del servicio.

Las precondiciones son propiedades importantes que el sistema debe comprobar antes de la ejecución de cierta operación, de estas propiedades puede depender el éxito de la ejecución de la operación ya que verifica el estado del sistema antes de ejecutar el servicio.

Una precondición incluye las siguientes propiedades: mensaje de error que es enviado al usuario en tiempo de ejecución cuando no se cumple la precondición, lista de agentes para los que se restringe su verificación y los comentarios que permiten documentar la precondición.

Post-condiciones

Hace referencia al estado final que presenta el sistema después de ejecutar un determinado evento, por lo tanto es el efecto de su ejecución, ya que representa el cambio del sistema una vez terminada la ejecución de la operación. Se pueden definir post-condiciones de creación y eliminación de objetos y modificar atributos

Las post-condiciones indican las condiciones que cumplirán el valor de retorno y los parámetros recibidos en caso de alguna modificación.

3.3.2 Relaciones entre clases

Una vez que se ha definido las clases entonces es necesario establecer el tipo de relación que va a existir entre ellas, en este punto se define como se comunican los objetos de las clases entre sí y como se puede interrelacionar entre dos o más clases cada una con características y objetivos diferentes.

Esto es de gran importancia ya que aquí se diferencian la mayoría de propuestas de modelos conceptuales.

Por medio de las relaciones se indica qué objetos se pueden ver o qué pueden hacer cuando actúan como agentes en los objetos de las clases del servidor (Pastor & Molina, Conceptual Modelling Primitives, 2007). Los objetos determinan el comportamiento del sistema colaborando entre sí, esta colaboración se logra gracias a las relaciones que existen entre ellos, para este proceso se identifican los siguientes tipos de relaciones:

Relaciones de agentes

Los servicios definidos en las clases son activados por medio de agentes por lo tanto cada clase debe tener uno o más agentes encargados de activar ese servicio.

Los agentes también intervienen en la parte de los atributos ya que en la clase se establece los atributos que son visibles para un agente de otra clase, definiendo con ello la seguridad. Es necesario tener en cuenta que por defecto una clase tiene todos sus atributos visibles para otra por medio de su relación de agente, es aquí en donde se debe modificar para restringir el acceso a ciertos atributos.

Si se amplía la noción de agente, también es posible especificar las relaciones por las que el agente puede navegar es decir se indica que objetos de una clase pueden ver a objetos de otra clase (Pastor & Molina, Object Model, 2007).

Relaciones de asociación

Luego de definir los atributos y servicios de una clase, es necesario agregar otro tipo de atributos (objeto-valuados) cuyo valor representa a un objeto de otra clase, con esto se determinan las relaciones semánticas entre las clases que representan relaciones semidirigidas entre ellas. Para conceptualizar estas relaciones se definen primitivas de asociación, agregación y composición.

Asociación

Las asociaciones son las relaciones más generales que existen entre clases, y describen las conexiones entre los diferentes objetos de dichas clases

Establece las siguientes propiedades básicas:

- Nombre de la asociación y su rol: que permite identificar desde otra clase
- Multiplicidad: es la especificación de las cardinalidades, indica el número de objetos que pueden estar asociados simultáneamente entre dos clases, estableciendo el número menor como cardinalidad mínima y el máximo de objetos como cardinalidad máxima, también se establece si la asociación es obligatoria o no pudiendo indicar si acepta o no nulos.
- Navegabilidad: Es la comunicación entre los objetos que puede ser unidireccional o bidireccional, con ello se determina la capacidad que tiene una instancia de la clase origen para acceder a las instancias de la clase destino por medio de la asociación que las conectan, es decir navegar desde objetos de una clase hacia objetos de otra clase. Por defecto la navegación es bidireccional.

Es importante aclarar que en el caso de cardinalidades uno a muchos si se desea eliminar un objeto del lado en que la cardinalidad es uno, contiene las siguientes opciones: no permitir, eliminar el objeto sin eliminar los otros objetos de la asociación o eliminar el objeto con todos los objetos relacionados.

Agregación

Existe ciertas clases que son componentes de otra clase, esto se representa mediante una relación de un todo con sus partes, considerando una clase como "todo" y la otra como "parte" de la anterior, cada uno con tiempos de vida independientes, entonces se puede considerar a la agregación como una especialización de la asociación, por lo tanto hereda todas sus propiedades.

En la agregación un objeto de la clase componente puede existir sin la necesidad de estar relacionado con un objeto de la clase compuesto y un objeto componente puede ser compartido por múltiples objetos de la clase compuesto.

Composición

Es un tipo especial de la Agregación, pero con una relación más fuerte, para ello agrega dos propiedades adicionales:

- Los objetos contenidos no pueden ser compartidos, por lo tanto cada uno de los componentes de la agregación pertenecen a un solo objeto compuesto.
- El objeto contenido es parte vital del que lo contiene, cuando el objeto compuesto se elimina los objetos de la clase componente se eliminan automáticamente. En caso de destruirse el objeto compuesto deben destruirse también sus partes o bien traspasar esas partes a otros objetos.

Relaciones de Herencia

Se puede definir la herencia entre clases como una clase secundaria que además de tener sus propios métodos y atributos hereda la estructura y comportamiento de una clase base o padre, estas propiedades heredadas pueden ser modificadas por las clases hijas, definiendo los conceptos de especialización y generalización.

Especialización

En este caso se toma un conjunto de entidades de un nivel superior y se forma conjuntos de entidades de un nivel más bajo, es decir considera una entidad como un supertipo y a partir de ésta genera varias entidades subtipos.

Es necesario establecer si un objeto va a estar especializado durante su vida útil que llegaría a ser temporal o durante toda su vida que llega a ser permanente, teniendo en cuenta que la clase hija no existe hasta que se genere un evento en la clase padre que desencadene el evento de creación

Generalización

Para que se pueda generalizar, las clases deben tener un conjunto de atributos iguales que se agrupan y forman una entidad global de nivel superior. Es la inversa del caso anterior, aquí se agrupan los subtipos para formar un supertipo. Se pueden agregar propiedades adicionales a la clase principal las cuales también serán accesibles para las clases hijos.

3.4 Modelo Dinámico

En el punto anterior se ha establecido toda la estructura estática del sistema con la definición de las clases sus atributos, relaciones y servicios; ahora se definen todos los aspectos dinámicos, estableciendo la comunicación que existe entre los objetos del sistema, para posteriormente determinar las características funcionales de los servicios y finalmente abarcar la interacción del usuario con el sistema, abordando con ello todos los modelos mencionados (estático, dinámico, funcional y presentación).

El Modelo Dinámico es el encargado de determinar los aspectos organizacionales del sistema como la secuencia en la que ocurren los servicios en un objeto o las interacciones que pueden ocurrir entre objetos. Para esto se especifica un conjunto de primitivas que se pueden representar mediante dos diagramas, el Diagrama de Transición de Estado para describir las secuencias de eventos en una clase y el Diagrama de Interacción para describir la comunicación e interacción entre los objetos.

3.4.1 Diagrama de Transición entre Estados

El Diagrama de Transición de Estados permite describir la secuencia de eventos válidos que puede ocurrir para un objeto. Para ello se utiliza estados que marcan las diferentes situaciones en las que puede estar un objeto y sus cambios de estado o transiciones, por lo tanto muestra el comportamiento de un sistema de información dependiente del tiempo. Este diagrama indica cómo se comportaría el sistema en tiempo real si existiera la tecnología perfecta.

Se pueden activar los objetos dependiendo del estado actual en el que se encuentren, por lo tanto la vida de un objeto está determinada por un conjunto de estados que se producen de manera secuencial. El estado actual es el que indica lo que le puede pasar al objeto en ese instante, su representación incluye estados y transiciones.

Estados

Los estados definen la situación en la que se encuentra un objeto perteneciente a una clase en un determinado punto, que es observable externamente y que perdura durante un periodo de tiempo finito, se pueden distinguir los siguientes tipos:

- Estado previo a la creación: Se produce antes de que los objetos se creen para que se produzca este tipo de estado tiene que suceder un evento de creación.
- Estado de destrucción: Se produce después de que los objetos son destruidos para que se produzca este tipo de estado tiene que suceder un evento de destrucción.
- Estado simple: son todos los posibles estados que se producen en un objeto por medio de los servicios.

En un diagrama de transición no se puede representar más de un estado previo a la creación y el estado de destrucción puede ser cero o uno, en cuanto a los estados intermedios no se tiene límite.

Transiciones

Las transiciones son el cambio de un estado a otro en un objeto, este cambio es producido por un servicio que es activado por un agente y refleja los posibles caminos que pueden existir para llegar a un estado final desde un estado inicial.

Las transiciones producidas desde un estado previo a la creación corresponden a servicios de creación mientras que las transiciones desde un estado de destrucción corresponden a servicios de destrucción, mientras que las que no se producen es los estados mencionados anteriormente corresponden a estados simples. Una transición puede comenzar y terminar en el mismo estado, en este caso el estado final es el mismo que el inicial.

Condiciones y Acciones

Una condición es una acción que el sistema puede detectar y es la responsable de los cambios de estados y las acciones, es lo que el sistema realiza cuando se cambia de estado, como parte de ese cambio el sistema puede realizar una o más acciones que son las respuestas a algún evento producido.

Por lo tanto las condiciones son los eventos del entorno que el sistema detecta. Estas interrupciones causan un cambio de estado pero para que se produzca dicho cambio el sistema realiza una o varias acciones, por lo tanto las acciones son las respuestas devueltas ya sean por medio de pantalla o almacenadas en variables para futuros eventos.

3.4.2 Diagrama de interacción entre objetos

Representa la comunicación entre objetos, los objetos interactúan a través de mensajes para cumplir ciertas tareas. Esta interacción puede ser representada mediante disparadores (triggers) y operaciones globales.

Disparadores / Triggers

Un disparador es un tipo especial de procedimiento almacenado que se ejecuta automáticamente cuando ocurre un evento en particular sobre los objetos asociados, para ello es necesario especificar las condiciones de disparo cuando se crea el trigger.

Ya que no necesita que el usuario lo ejecute, es utilizado para implementar reglas de negocio, es decir restricciones, requerimientos, validaciones o actividades especiales que se ejecutarán en un momento determinado. Presenta las siguientes propiedades:

- Clase de definición: Es la clase en la que especifica la condición de activación.
- Condición de desencadenante: Es una fórmula booleana bien formada que debe cumplirse para que el disparador se active.
- Destino: Hace referencia a la clase que recibe el servicio disparado.
- Ruta al destino: es la secuencia de nombres de roles desde la clase de definición hasta el destino.
- La cláusula de selección de Objeto: Es una fórmula bien formada que restringe el conjunto de objetos en el que se ejecuta el servicio desde la clase de definición.
- Servicio: Es el servicio que se activa junto con sus valores de inicialización

- Comentarios: Sirve para documentar la definición desencadenante

Dependiendo del destino los triggers pueden tomar la siguiente forma:

- SELF: Si el objeto definición es el mismo que el destino
- OBJETO: Si el desencadenante se dirige a un objeto de otra clase
- CLASE: Si el desencadenante se dirige a todos los objetos de la misma clase
- PARA TODO: Si el desencadenante se dirige a un subconjunto de los objetos de una clase

Debido a que los servicios se activan cuando se cumple una determinada condición, el usuario no se entera de esto y se considera a los disparadores como una actividad interna dentro del sistema.

Transacciones y operaciones globales

Son unidades de ejecución compuestas por múltiples servicios que pertenecen a diferentes clases que no están necesariamente relacionadas. Estas transacciones están compuestas por un nombre que en caso de coincidir con el de una clase o servicio no generará conflictos, además cuenta con propiedades como: alias, comentarios y mensajes de ayuda y pueden llevar argumentos teniendo las mismas propiedades que los de los servicios locales. Estos argumentos pueden ser de dos tipos:

- Argumentos de entrada: Antes que el servicio global se ejecute el agente encargado de activar el servicio debe asignar el valor a estos argumentos.
- Argumentos de salida: El valor de estos argumentos se produce cuando el servicio global finaliza, su valor puede depender si el servicio finaliza con éxito o no

Puesto que con las operaciones globales se tiene acceso a todas las clases, estas operaciones pueden afectar al operador “FOR ALL” por lo tanto se debe utilizar “Where” para así restringir la población en la que deben ejecutarse estos servicios.

3.5 Modelo Funcional

El modelo funcional representa todas las actividades, procesos y operaciones que el sistema debe realizar para satisfacer las necesidades del usuario. El objetivo de este modelo es determinar la manera en la que un estado del objeto es afectado por los eventos ejecutados, este punto es muy importante ya que en la mayoría de los casos se los deja a cargo de los lenguajes de programación formando una dependencia a la tecnología que se usa en la implementación.

No existe una técnica para especificar la funcionalidad de los servicios de una clase, por lo tanto en algunos casos se utiliza ciertos diagramas como los diagramas de iteración o los diagramas de transición de estados o incluso se puede llegar a utilizar una especificación tipo OCL, el lenguaje de especificación de restricciones de objetos de UML.

Para determinar las primitivas necesarias este documento se basa en especificaciones anteriores como la presentada en OASIS (Pastor, Oasis: Un lenguaje de especificación OO, 1992) y mejorada en el OO-Method (Pastor & Molina, The OO-Method, 2007), obteniendo especificaciones que detallen de

manera precisa los cambios de estados que sufre un objeto a partir de una serie de eventos ocurridos, todo esto mediante evaluaciones, condiciones de evaluación y categorías de evaluación.

3.5.1 Especificación del efecto del evento usando evaluaciones

Es necesario determinar el valor de los atributos que tendrá un objeto cuando ocurra un evento que afecte a dicho objeto, para ello es necesario determinar los eventos relevantes para un atributo, es decir los que afectan o cambian el valor de ese atributo. Cada atributo va a tener un conjunto de eventos relevantes que afectan su valor y un evento también va a tener un conjunto de atributos a los que modifica.

3.5.2 Condiciones de evaluación

Generalmente un evento puede modificar de distintas maneras un atributo de acuerdo a una serie de condiciones establecidas, para ello se puede agregar varios eventos, o se puede crear múltiples eventos artificiales. Para determinar la evaluación que debe elegirse cuando ocurre un evento se establecen condiciones de evaluación. Para ello primero se establece el estado que presenta un objeto antes de que el evento suceda, entonces se selecciona la evaluación que cumpla con la condición definida.

Es necesario considerar que las condiciones de evaluación deben ser mutuamente excluyentes para garantizar que solo se cumpla para una determinada condición, en el caso de que un atributo tenga una única evaluación entonces no se debe asignar condiciones de evaluación y en el caso de tener varias evaluaciones, todas deben tener una condición, menos la que es por defecto.

3.5.3 Categorización de evaluaciones

Uno o múltiples eventos pueden modificar un atributo de un objeto de diversas formas, por lo tanto es necesario una especificación simple de las evaluaciones que describan la forma en la que un evento puede modificar un atributo, se describe tres categorías: estado, cardinal y situación.

Las evaluaciones de estado

Estas evaluaciones son aquellas en las a partir de un evento el atributo toma un nuevo valor sin importar los valores que haya tenido anteriormente: es decir el estado final del atributo después de producirse un evento no depende del valor del estado inicial.

Las evaluaciones cardinales

En estas evaluaciones el nuevo valor del atributo cuando se desencadena un evento depende del valor que ha tenido el atributo previo al evento. El nuevo valor tiene un incremento o decremento respecto al valor anterior, esta variación en el valor es cuantificable por lo tanto el efecto que tiene el evento en un atributo depende del valor que tiene el atributo antes de que se produzca el evento.

Las evaluaciones de situación

Aquí pertenecen las evaluaciones en las que el valor de un atributo se da dentro de un dominio bien definido en donde los valores que puede tener un atributo refleja cada una de las situaciones que

puedan describirse en un atributo, este nuevo valor podría depender o no del valor anterior. En esta evaluación representa un nivel dinámico lógico en las que se basan las evaluaciones.

La especificación de la funcionalidad de los eventos junto con la especificación de las transacciones y operaciones y el diagrama de Interacción componen el conjunto de elementos que permiten definir toda la funcionalidad de un sistema de información.

3.6 Modelo de Presentación

Hasta ahora se ha tenido en cuenta tanto los aspectos estáticos como dinámicos para definir un sistema ahora el presente estudio se enfoca en definir los aspectos de interacción de los usuarios para esto se puede detallar propuestas que se basan en modelado de tareas para representar las interfaces de usuario.

Es necesario que el sistema refleje la forma en la que los usuarios ven e interactúan con el entorno por medio de la inclusión de aspectos Interacción Persona-Ordenador (HCI, Human-Computer Interaction, en inglés).

Se representa mediante patrones de interfaz para especificar los aspectos importantes y determinar la manera que se da la interacción del usuario con el sistema. Los aspectos importantes de la interfaz de usuario son capturados de manera independiente del diseño y de la implementación, es necesario que cada patrón de interfaz sea desarrollado en el espacio del problema que tendrá una representación de software adecuada dentro del espacio de solución.

3.6.1 Estructura general

La descripción del modelo de presentación se realiza en tres niveles que van desde el más específico al general.

Nivel 1: Estructura de acceso al sistema.

Es representado con su árbol jerárquico de acciones.

Este nivel determina la interacción del usuario con el sistema al momento de acceder al sistema, lo realiza mediante una abstracción con forma de árbol para representar en un primer nivel a los nodos raíz que tienen nodos de hoja que definen unidades de interacción que serán representados en el segundo nivel.

Nivel 2: Unidades de interacción

Pueden ser Simple (Servicio, Instancia, Población) o Compuesta (Maestro/Detalle).

En este nivel se establecen las unidades de interacción y la navegación existente entre estas. La unidad de interacción representa el alcance que tiene el usuario en el sistema, con las siguientes opciones:

- Ejecutar un servicio.

- Consultar la población de una clase.
- Visualizar los detalles de un objeto específico.
- La agrupación de una colección de unidades de interacción preexistentes en una unidad de interacción compuesta, siguiendo una estructura maestro / detalle.

Para modelar los aspectos mencionados anteriormente se definen cuatro tipos de unidades de interacción, estos pueden ser simples o compuestos, en cuanto a los simples son modelados en el tercer nivel y se los conoce como elementos básicos

Nivel 3: Elementos básicos

Este nivel está conformado por elementos básicos mediante los cuales se construyen las conexiones, estos elementos tienen un ámbito de aplicación dependiendo de la unidad de interacción a la que pertenezcan por lo tanto no pueden combinarse de forma arbitraria.

Estos elementos pueden ser de Entrada, Selección definida, Agrupación de argumentos, Dependencia de argumentos, Filtro, Criterio de orden, Conjunto de pantalla, Acciones disponibles, Navegación.

Elementos básicos

Estos elementos determinan aspectos concretos de la interacción del usuario con el sistema, refiriéndose al contexto en el que se ejecuta un servicio, por lo tanto se relaciona con una unidad de interacción del servicio.

Los elementos básicos pueden ser utilizados en diversos contextos, por ejemplo el argumento de dependencia es útil solo en el contexto de la ejecución del servicio. Es por esto que es necesario determinar el contexto para dar así un correcto uso a cada elemento.

Entrada

Este tipo de elementos se utiliza para capturar aspectos relevantes de los datos que el usuario ingresa al momento de asignar los valores a los argumentos de los servicios, estos aspectos involucran máscaras, validaciones, valores válidos y mensajes de ayuda.

Generalmente se utilizan los elementos de entrada para asignar valores en los argumentos o para filtros mientras que con los argumentos de servicios no deben relacionarse con elementos de entrada.

Selección definida

Al momento de ingresar información en ciertas ocasiones es necesario que el usuario pueda seleccionar un valor de un conjunto de valores establecidos, por esto la selección definida permite especificar con anticipación un conjunto de valores válidos para un elemento del modelo.

A menudo se utilizan los elementos de selección definida para variables de filtro y atributos de asignación limitando así los posibles valores que pueda llegar a tomar dicho atributo.

Agrupación de argumentos

Permite organizar los argumentos de entrada como una lista ordenada que define la secuencia en la que los argumentos se muestran al usuario, también es utilizada para organizar y armar grupos y subgrupos de argumentos. Tiene las siguientes propiedades:

- Orden de argumentos.
- Grupos de argumentos.
- Relaciones de contención de grupos y argumentos.

El orden de argumentos indica la secuencia en la que los argumentos se muestran al usuario. Todos los argumentos de cada unidad de servicio de interacción tienen un orden, que es en el orden en el que fueron definidos, por lo tanto cada uno tiene un elemento básico de agrupamiento por defecto. Los argumentos pueden pertenecer solo a un grupo sin importar si ese grupo está dentro de otros.

Argumento de dependencia

Permite definir relaciones de dependencia tanto de los valores o de los estados entre dos argumentos que pertenecen a un mismo servicio. Utiliza las reglas ECA (evento, condición y acción) para todos los argumentos de entrada de un servicio determinado.

En el momento en el que produce un evento de interfaz con un argumento específico entonces se realiza cierta acción dependiendo si cumple con la condición. Con esto se puede especificar una conducta dinámica para la interfaz del usuario. Un argumento de dependencia tiene los siguientes datos:

Evento de interfaz

Define lo que ocurre con el argumento en el cuál se aplica el argumento de dependencia, puede ser de dos tipos: SetValue (indica que el argumento ha recibido un valor) y SetActive (indica la activación o no de un argumento); ambos tienen un argumento de dependencia que representa:

- El valor que se le ha dado al argumento, SetValue.
- El estado activado, v = verdadero indica que el argumento está activado; f = falso indica que está desactivado.

Cada evento tiene un agente asociado, el agente puede ser:

- Usuario, cuando el evento es causado por el usuario.
- Interno, cuando el evento es causado por la acción de otro elemento básico de Argumentos de dependencia.
- *, que abarca tanto el usuario como el interno

Condición

La condición es una fórmula booleana que debe cumplirse para que sucedan ciertas acciones, puede estar compuesta por constantes, funciones y argumentos.

Acciones

Tienen un conjunto de elementos y cada elemento consta de tres partes:

- Argumento: Hace referencia al nombre de un argumento del servicio
- Evento: Se refiere a las acción que es realizada en un argumento, que puede ser SetValue o SetActive
- Fórmula: Define el valor que el programa recibirá

Filtro

Es utilizado en situaciones en las que la selección de un objeto de una clase se basa en ciertos criterios aplicados para los valores de sus atributos, entonces al aplicar los criterios en la población de objetos, se reduce el conjunto de objetos lo que facilita la búsqueda y selección. Las búsquedas son establecidas con valores concretos que dependen de la operación en la que se encuentran, estos valores se les denomina variables de filtro.

Variables de filtro con valores de referencia

Una variable de filtro con valores de referencia tiene las siguientes propiedades:

- Nombre: Debe ser único ya que identifica a la variable.
- Alias: identifica el parámetro de búsqueda
- Tipo de datos: Establece los valores que el usuario puede ingresar.
- Valor por defecto: Configura un valor inicial que es sugerido al usuario como parámetro para la búsqueda.
- Elemento básico de entrada: Determina el rango de valores para los parámetros de búsqueda.
- Elemento básico de selección definida: Restringe al usuario los valores que puede ingresar en el parámetro de búsqueda.

Variables de filtro con valores de objetos

Tiene propiedades iguales a las de las variables de filtro con valores de referencia, la diferencia es el tipo de datos que se usan para el conjunto que se muestra en pantalla y la interacción de la población. Las variables de filtro son útiles para definir los criterios de búsqueda.

Criterio de orden

Indica la manera en la que se ordenan los resultados que obtenidos luego de un proceso de búsqueda, considerando los valores de las propiedades del listado de objetos que se obtienen como resultado de una búsqueda. Se puede considerar dos opciones ascendentes y descendentes.

Estos elementos se presentan como una lista de pares conformados por atributo y dirección, en donde el primero indica el nombre del atributo de la clase relacionada y el segundo define la forma de ordenar los resultados, pueden ser ascendente o descendente.

Conjunto de visualización

Cuando se presenta la lista de objetos, luego de aplicar los filtros de búsqueda y los criterios de ordenación, entonces se definen las propiedades que van a ser visualizadas por el usuario de dichos objetos. Para esto es importante analizar las propiedades que son relevantes en el contexto en el que se encuentra.

Navegación

Cuando se define un conjunto de visualización, se identifican las propiedades que se muestran al usuario, teniendo en cuenta que se puede mostrar propiedades de objetos relacionados ya que las clases en el sistema están relacionadas, por lo tanto el usuario puede navegar a través de un objeto seleccionando sus propiedades. La navegación se realiza por medio de las relaciones que existen entre las clase, ya sea asociación, agregación o composición.

Un modelo de navegación es una parte esencial del proceso de modelado.

Acciones

Además de permitir la navegación a partir de un objeto, también se pueden ejecutar ciertas acciones sobre dicho objeto mediante un servicio definido. Para esto es necesario primero definir una lista de servicios que estarán disponibles para la ejecución desde un objeto de la clase.

Unidades de Interacción

Este concepto es de gran importancia en el modelo de Presentación debito a que describe un escenario en el que un usuario puede realizar tareas específicas por lo tanto detalla la interacción entre el usuario y el sistema. Una colección de unidades de interacción forma la interfaz de usuario, en la que existen distintos niveles. Se distinguen los siguientes tipos de unidades de interacción:

- Ejecución de un servicio que forma la unidad de interacción de servicio.
- Manipulación de un objeto que forma la unidad de interacción de instancias.
- Manipulación de una colección de objetos que forma la unidad de interacción de la población.
- Manipulación de múltiples colecciones relacionadas de objetos, que forma la unidad de interacción Maestro/Detalle.

Unidades de interacción de servicio

Por medio de estas unidades es posible definir un escenario en el que el usuario ejecuta un servicio por medio de la interacción con el sistema, incluyen las siguientes características:

- Presentación de argumentos de servicio ordenados y agrupados.
- Presentación de los valores predeterminados para los argumentos del servicio, para ello se utiliza la propiedad de valor por defecto.
- Validación de valores de argumento.
- Información de ayuda sobre cada argumento.
- Información de ayuda sobre el servicio.
- Comentarios en caso de error durante la ejecución del servicio, brindan información sobre los argumentos del servicio.

Es necesario resaltar que una unidad de interacción de servicio detalla un escenario en concreto pero no el orden en el que se realiza, entonces la información que se captura en el modelo conceptual es independiente de la implementación final. Por lo tanto es necesario hacer una distinción entre nivel de modelo de presentación abstracto en donde se detallan los patrones de presentación independientes de la implementación y nivel de modelo de presentación concreto que se asocia directamente a la parte de presentación.

Unidad de Interacción de Instancias

En la unidad anterior se hacía referencia a la ejecución de servicios, en este caso esta unidad hace referencia a la organización de objetos individuales. En esta unidad la información que se detalla es la de un objeto individual junto con los servicios que éste puede ejecutar.

Al igual que la unidad vista anteriormente aquí se define la estructura de un escenario pero no la forma en la que implementa, la información que se captura en el modelo conceptual es independiente de la implementación final.

Unidad de interacción de la población

Una vez definidas las unidades para la ejecución de servicios y para la visualización de un objeto entonces ahora se parte de las unidades para manipular un conjunto de objetos de cualquier clase. Esta unidad de interacción presenta varios objetos y permite seleccionar y clasificar objetos, visualizar la información y servicios disponibles y listar otros escenarios a partir de este.

Al igual que las unidades anteriores se especifica la estructura de los escenarios para la búsqueda y consulta pero no la forma de implementación.

Unidad de Interacción Maestro / Detalle

En esta unidad se definen varias colecciones de objetos de diferentes clases que están relacionadas entre sí, por lo tanto combina las tres unidades revisadas anteriormente, se pueden distinguir dos tipos de rol:

- Maestro: es el centro de interacción principal, es una unidad de interacción poblacional.
- Detalle: son las interacciones secundarias que están relacionadas con el rol maestro

Una clase maestra puede relacionarse con varias instancias de la clase, mientras que un detalle solo puede relacionarse con una instancia maestra. Se puede utilizar una unidad de interacción maestro/detalle como un detalle, permitiendo así un escenario de interacción más complejo.

Árbol de Jerarquía de acciones

Una vez descritas las diferentes unidades de interacción, entonces se debe la estructura y la presentación de estas interacciones al usuario, esto se puede comparar con lo que es el menú principal en una aplicación por lo tanto describe la funcionalidad del sistema y como un usuario puede acceder a ello.

Toda esta funcionalidad se agrupa en forma de árbol partiendo desde un nodo más general hasta llegar a uno más detallado en las hojas que son las acciones a las que el usuario accede directamente considerados como las unidades de interacción. También existirán acciones a las que el usuario podrá acceder mediante la navegación definida.

Se puede definir un árbol de jerarquía para cada tipo de usuario dependiendo del acceso a la información que tenga un determinado usuario y los atributos y servicios a los que tenga visibilidad.

En un árbol de jerarquía de acciones se pueden distinguir tres niveles:

- Nivel global: tiene como propiedad los comentarios que se utilizan para la documentación.
- Nivel de nodo no hoja: tiene como propiedades: alias que permite al usuario identificar el nodo de mejor manera, mensaje de ayuda para mostrar al usuario un texto con información sobre el nodo y comentarios para la documentación respectiva.
- Nivel de nodo hoja: contiene todas las propiedades del nodo no hoja con una adicional que es la Unidad de interacción que representa al escenario específico asignado en el nodo.

Los árboles de jerarquía de acción establecen como están organizadas las acciones posibles en el sistema sin embargo no define una implementación en particular, se puede acomodar a diferentes implementaciones.

3.7 Presentación de Primitivas

Se ha descrito los cuatro componentes del modelo conceptual: modelo de objeto, modelo dinámico, modelo funcional y modelo de presentación, entonces se cuenta con la información necesaria para representar un sistema de información utilizando la colección de primitivas conceptuales que se ha presentado y que a continuación se resumen en la tabla 1:

Modelo	Primitivas
Modelo de Objetos	• Clases: atributos y servicios • Relaciones entre clases: asociación y herencia • Relación de agentes
Modelo Dinámico	• Transición de estados • Interacción entre objetos
Modelo Funcional	• Evaluaciones • Condiciones de evaluación • Categorización de evaluaciones
Modelo de Presentación	• Elementos básicos • Unidades de Interacción • Árbol de jerarquía de acción

Tabla 1. Primitivas

CAPITULO 4

Desarrollo de software a partir de modelos conceptuales (WebRatio)

En este capítulo se detalla como soporta una herramienta de desarrollo de software dirigido por modelos cada una de las primitivas planteadas anteriormente. Para ello se ha seleccionado WebRatio como herramienta y se desarrollará un caso de estudio en concreto que es la gestión de notas de gasto, esto permitirá medir el nivel que tiene para representar dichas primitivas

4.1 Herramientas de desarrollo

El desarrollo de software dirigido por modelos conceptuales facilita el tratamiento de la información y mejora la calidad del producto software, para ello es necesario que el modelo abarque todos los requisitos y represente el escenario correcto del sistema a desarrollar.

A pesar de tener un modelo de calidad puede suceder que la herramienta generadora de código utilizada no sea capaz de representar todos los aspectos detallados en el modelo y con ello se pierdan propiedades importantes y no se obtenga el resultado esperado.

Existen varias herramientas para el desarrollo de aplicaciones a partir de modelos conceptuales que permiten diseñar un sistema considerando: capa de datos, presentación y navegación. Entre las más conocidas están WebRatio e Integranova, las dos tienen un grado de automatización elevado.

La manera en la que represente todas las primitivas planteadas una herramienta y su independencia respecto a la plataforma, determinará el éxito y la calidad del software desarrollado.

4.2 WebRatio

WebRatio es una plataforma de desarrollo de software que a partir del modelo conceptual diseñado por el desarrollador genera automáticamente el código para la aplicación, integrando las diversas partes que conforman el proyecto. En la figura 1 se muestra el logo de la herramienta.

WebRatio tiene su propio lenguaje de modelado, IFML que fue reconocido como estándar por el consorcio Object Management Group (OMG) en 2013. Este lenguaje con ayuda de algunas extensiones puede modelar de manera visual el front-end de las aplicaciones y es independiente de la tecnología que se utilice para la implementación de la interfaz del usuario. (WebRatio, 2014)

Figura 1. WebRatio

A continuación se analizará si esta herramienta cubre las primitivas planteadas en el capítulo anterior y en qué nivel cubre, esto se verificará con cada uno de los modelos definidos: objetos, dinámico, funcional y de presentación, para este análisis se realizó el desarrollo de la gestión de notas de gasto cuya descripción se detalla en el Anexo 1 – Ejercicio planteado (Gestión de notas de gasto)

4.2.1 Modelo de objetos

Para la creación de clases y relaciones el editor de WebRatio permite la definición de objetos y el vínculo entre estos, por medio de un modelo de datos con el cuál se generará un fichero XML. Para la representación de los datos se utiliza el modelo de Entidad-Relación que produce un esquema del sistema que se convierte en la base de datos relacional.

WebRatio presenta las opciones para generar entidades, relaciones y dos componentes adicionales para indicar la relación de agentes y la generalización como se muestra en la figura 2.

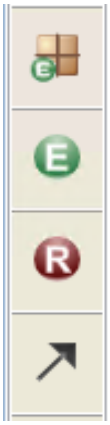

Figura 2. Modelo de Dominio

Clases

Las entidades representan a un conjunto de objetos que comparten características en común y están compuestas por objetos a los que describe, llamados instancias.

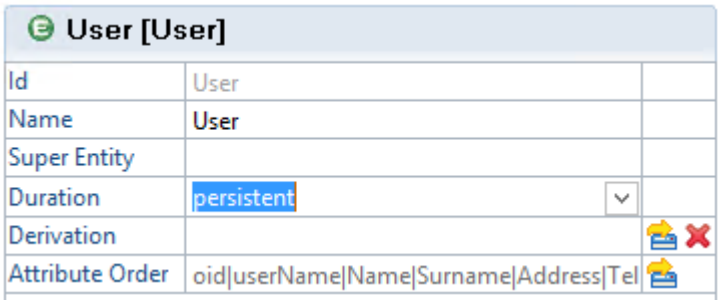

Figura 3. Propiedades de la Entidad

Las clases son representadas por entidades, al momento de generar una entidad tiene las propiedades que se muestran en la figura 3 y son:

- Id: es el identificador para la entidad

- Nombre: permite reconocer de una manera más fácil a la entidad, se puede utilizar caracteres especiales.
- Super Entidad: en caso de tener herencia le muestra el nombre de la superclase a la cual hereda
- Duración: esta propiedad hace referencia al tiempo de vida de los datos, tiene tres opciones:
    - Persistencia, indica que los datos van a ser almacenados en la base de datos
    - Alcance de sesión, la entidad permanece almacenada en memoria mientras dure la sesión, una vez finalizada la sesión los datos se perderán
    - Alcance de aplicación, similar al caso anterior pero con la diferencia de que los datos existen mientras la aplicación esté activa y esa información es compartida por todas las sesiones.
- Derivación: Mediante esta propiedad se puede utilizar datos procedentes de otras tablas que no se encuentren dentro del modelo.
- Orden de los atributos: Para la organización de los atributos dentro de la entidad.

Atributos

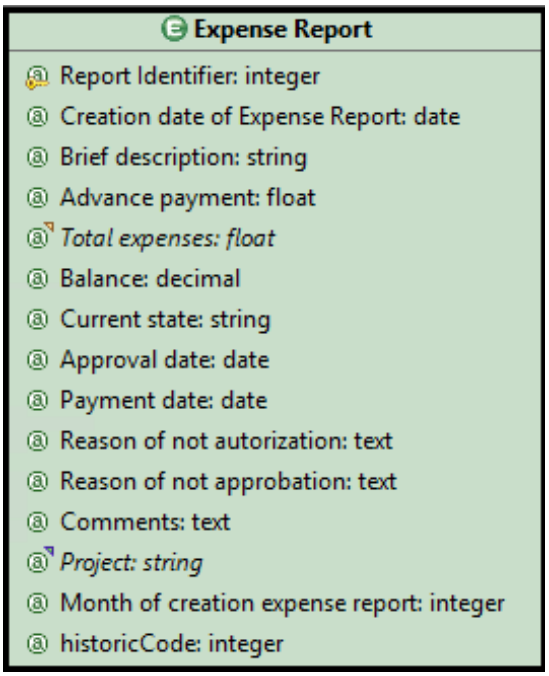

Figura 4. Atributos de una Entidad

Los atributos de una clase varían de acuerdo a la información que se necesita almacenar, en la figura 4 se muestra los atributos que tiene un reporte de gastos. Es necesario definir un atributo como llave principal de la clase, es el que va a identificar de forma única cada instancia que se genere de la entidad. Los atributos cuentan con las siguientes propiedades:

- Id: Permite identificar de manera única al atributo
- Nombre: Se puede utilizar nombres nemotécnicos para los atributos, lo que permite identificarlos de mejor manera
- Llave: Sirve para indicar si el atributo es la llave primaria de la entidad

- Tipo: Hace referencia al tipo de contenido que tiene este atributo, es decir al tipo de información que se almacenará en este campo, WebRatio soporta diversos tipos, entre los cuales están los condicionales (booleano), fechas (date), enteros (integer), caracteres (string), entre otros.
- Derivación: aquí se puede definir un atributo como constante o derivado. WebRatio tiene varias opciones como se muestra en la figura 5:

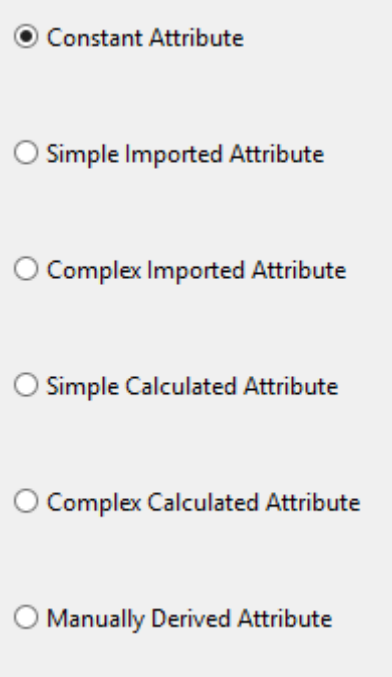

Figura 5. Derivación

  - Constante: una vez que se crea el objeto el valor de este atributo permanece constante hasta que se destruya.
  - Simple Importado: obtiene el valor de un atributo que se encuentra en otra tabla
  - Complejo Importado: hace referencia al anterior pero en este caso se toma los valores que cumplan ciertas condiciones
  - Simple Calculado: almacena valores calculados, es decir se pueden hacer operaciones con otros atributos y almacenarlos en este campo
  - Complejo Calculado: como en el caso anterior, pero incorporando también condiciones en los cálculos.
  - Derivado Manualmente: Permite utilizar información de base de datos ya existentes.

Servicios

Para definir el comportamiento de las entidades WebRatio permite incorporar eventos.

Eventos

El evento afecta al estado de la aplicación, por lo tanto en WebRatio no se lo realiza dentro del modelado de datos sino en otra sección. Los eventos incorporan las operaciones básicas como se muestra en la figura 6, que son: creación, modificación y eliminación.

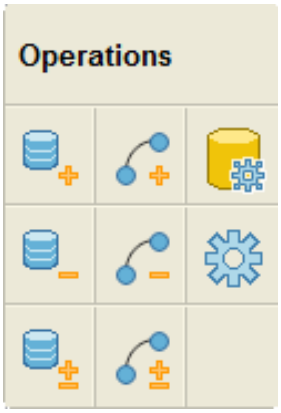

Figura 6. Eventos WebRatio

- Creación: Esta operación permite crear nuevas instancias de la clase a la que está relacionada. Tiene la posibilidad de almacenar múltiples instancias al mismo tiempo y al momento de crear el evento se relaciona con la entidad a la que representa, en la figura 7 se utiliza para la creación de los proyectos.

Figura 7. Evento Crear

- Modificación: Es el encargado de modificar la información en la base de datos. Es necesario que reciba como argumento el identificador de la instancia que se va a modificar, y al igual que en el anterior al momento de crear es necesario que se relacione con la entidad a la que hace referencia. Siguiendo con el mismo ejemplo en la figura 8 se muestra la modificación de un proyecto.

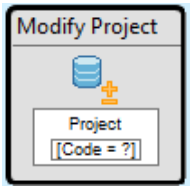

Figura 8. Evento Modificar

- Eliminación: Se utiliza para eliminar una instancia de la base de datos. Requiere como argumento el identificador de la instancia a eliminar y automáticamente elimina esa instancia. Al momento de crear se relaciona directamente con la entidad a la que representa. La figura 9 representa la eliminación de un proyecto.

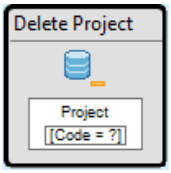

Figura 9. Evento Eliminar

Además de estas operaciones WebRatio también habilita otras operaciones como:

- Conexión: Permite asociar instancias de distintas clases, una origen y una destino. Estas instancias se conectan mediante una función de relación predefinida que aplica condiciones. La figura 10 muestra cómo se asigna un empleado a un proyecto.

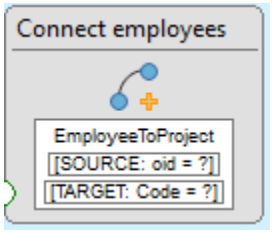

Figura 10. Evento Conexión

- Desconexión: Permite eliminar la asociación entre instancias de distintas clases que están conectadas mediante una relación predefinida y así poder eliminar las relaciones que existen. Como se puede apreciar la figura 11 en donde se quita al empleado del proyecto mediante la desconexión.

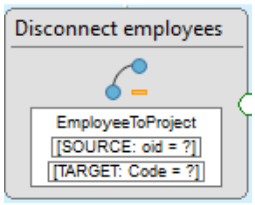

Figura 11. Evento Desconexión

- Reconexión: Modifica la asociación entre instancias de distintas clases conectadas mediante una relación predefinida. Esta operación permite conectar o desconectar las instancias de las entidades mediante condiciones definidas con anterioridad. En la figura 12 se vuelve a conectar al empleado con el proyecto.

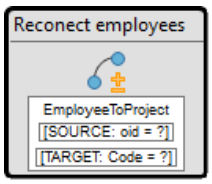

Figura 12. Evento Reconexión

- Procedimiento Almacenado: Se utiliza para ejecutar un procedimiento que ha sido previamente creado y almacenado en la base de datos. La figura 13 muestra la representación de un procedimiento en WebRatio.

Figura 13. Evento Procedimiento Almacenado

- No Operación: Es un evento auxiliar que se utiliza para ejecutar procesos en los que se puede incorporar condiciones. La figura 14 muestra su representación.

Figura 14. Evento No Operación

Argumentos

Como los argumentos son una parte esencial de los servicios, estos están presente en las operaciones de WebRatio. En los eventos de creación, modificación y eliminación es necesario el envío de argumentos del formulario que recibe la información hacia la operación a ejecutarse, y existe la posibilidad del paso de argumento en viceversa.

WebRatio diseña los eventos mediante enlaces de navegación, estos enlaces asocian los eventos y también permiten el paso de argumentos. Estos parámetros se definen como “Parameters Binding” que se muestran en la figura 15.

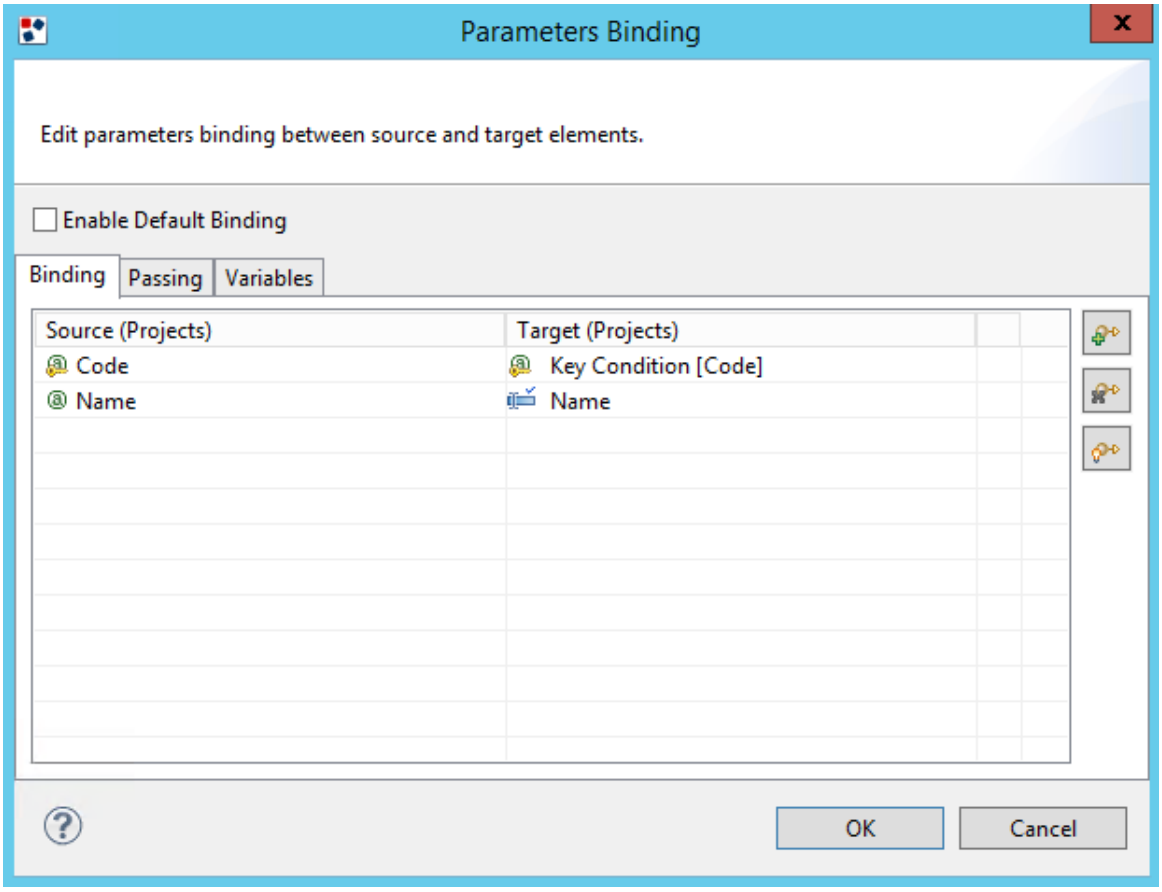

Figura 15. Paso de Parámetros

- Se permite el envío de varios parámetros a la vez, relacionando el campo del formulario de envío con un determinado campo de la entidad de llegada.
- Para la recepción de argumentos, es necesario crear un nuevo enlace entre las dos clases como muestra la figura 16. Lo que determina el flujo de la información es la dirección de la flecha del enlace creado.

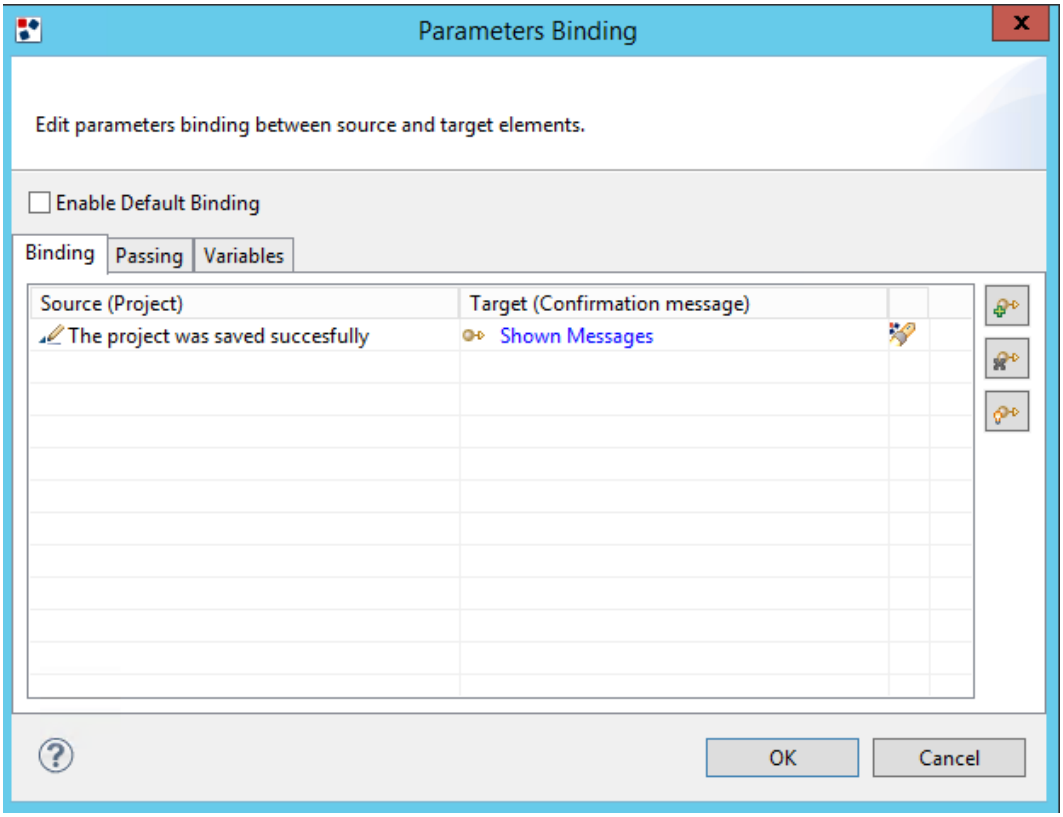

Figura 16. Recepción de Parámetros

Transacciones

Consiste en dividir los proyectos en partes que se puedan reutilizar en distintos sitios dentro de un mismo proyecto. Es útil para el modelado de proyectos extensos y complejos. Para soportar esta propiedad WebRatio ha destinado la creación de módulos.

Figura 17. Módulos

Se pueden diferenciar tres tipos de módulos dependiendo de la información que se asigna, como se muestra en la figura 17. Estos son: de acción, de contenido y los híbridos.

- Módulos de acción: Permiten agrupar una secuencia de operaciones que se ejecutan en un grupo de argumentos de entrada para posteriormente devolver datos procesados como argumentos de salida.
  Se puede observar en la imagen 18 que el módulo recibe como datos de entrada: el empleado y el mes, para proceder a seleccionar los gastos de ese empleado en ese mes y con ello crear un histórico. Como respuesta el módulo da un mensaje de correcto si la transacción se realizó con éxito caso contrario un error.

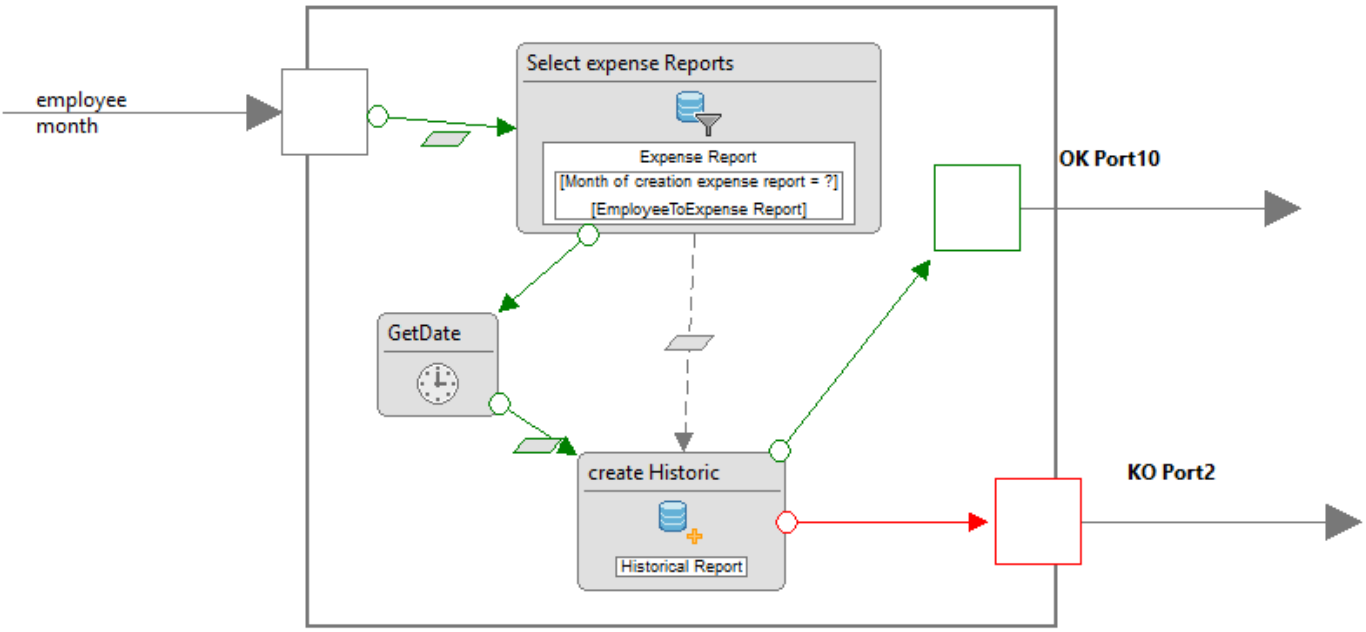

Figura 18. Módulo de acción

- Módulos de contenido: A diferencia del módulo anterior este únicamente contiene elementos visuales sin incluir nada de operaciones. Por ejemplo se puede utilizar este módulo para mostrar la información del usuario que inicia sesión, esta información se va a visualizar en cada una de las páginas entonces únicamente se reutiliza el módulo. También se puede utilizar este tipo de módulo para crear subpáginas.

- Módulos Híbridos: Este módulo mezcla los dos módulos revisados anteriormente por lo tanto contiene una colección de páginas y acciones (WebRatio, 2014). La figura 19 muestra un módulo híbrido realizado para ver un reporte de gastos, recibe como parámetro el identificador del reporte y con esa información busca en todas las tablas relacionadas al gasto que pueden ser: proyecto, líneas de gasto, tipo de pago, etc. Con esa información detallada arma el reporte que se muestra al usuario.

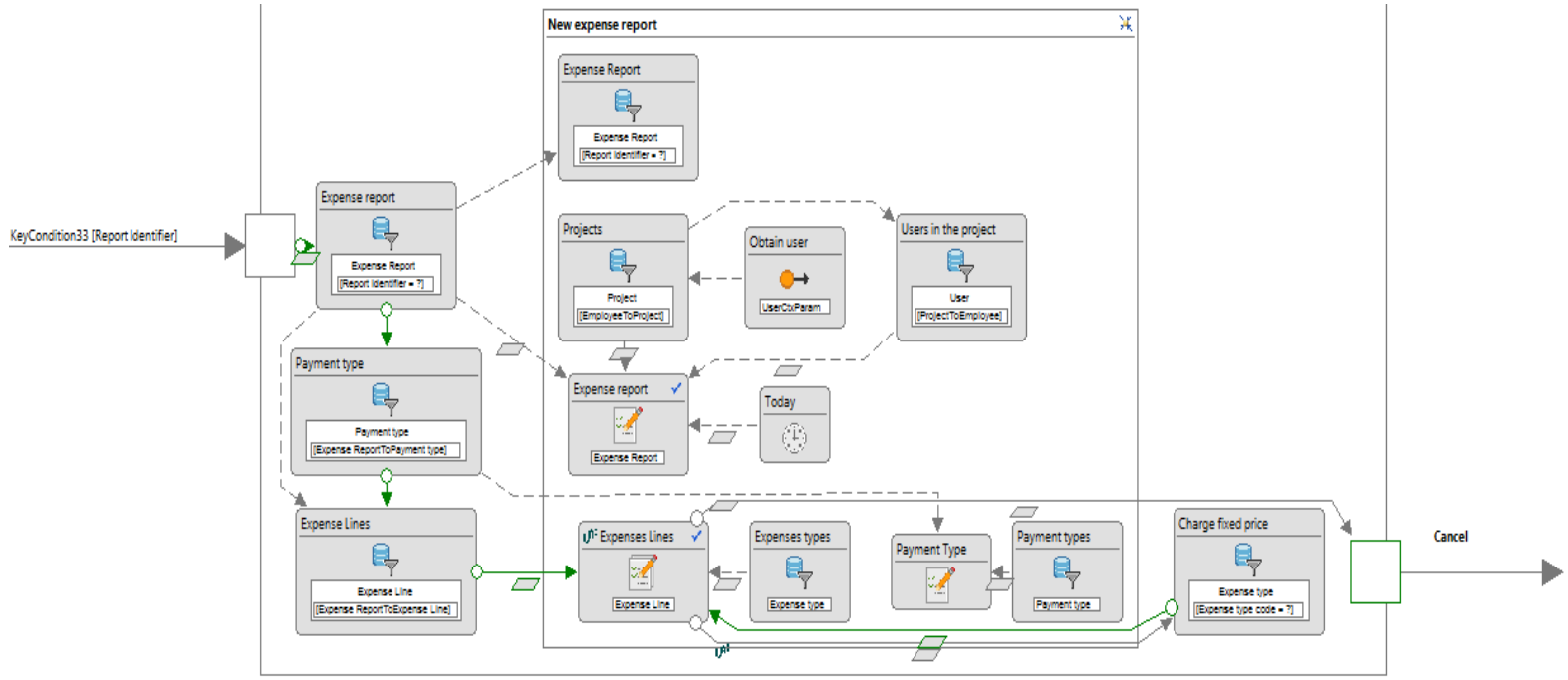

Figura 19. Modulo Híbrido

Restricciones de Integridad

En WebRatio las condiciones de integridad se agrupan en: condiciones de selectores y condiciones de visibilidad.

- Condiciones de selectores: están formadas por un conjunto de criterios o condiciones. Se utilizan para recuperar las instancias de una entidad (Bruno, 2011), se distinguen tres tipos:
    - Condición de atributo: Recupera únicamente las instancias de la entidad que satisfagan un valor de atributo determinado. En la figura 20 se crea una condición para recuperar al gerente de un departamento por lo tanto se indica que la entidad en el atributo Manager sea igual a “true” para que cumpla con la condición.

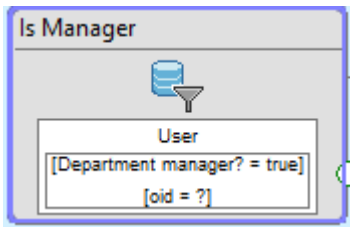

Figura 20. Condición de atributo

    - Condición de clave: Recupera las instancias de la entidad que tengan en su atributo seleccionado como clave, el valor que se indica. En la figura 21 se puede observar que al momento de eliminar un reporte de gasto es necesario crear una condición indicando que el identificador del reporte debe ser igual al parámetro enviado para proceder con la eliminación.

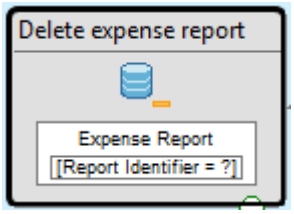

Figura 21. Condición de clave

- Condición de relación: Recupera las instancias de una entidad que están relacionadas con instancias de otra entidad, esta relación debe definirse anteriormente. En la figura 22 se utiliza esta condición para extraer los empleados relacionados a un determinado proyecto.

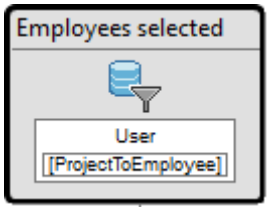

Figura 22. Condiciones de Relación

- Condiciones de visibilidad: Por medio de estas condiciones es posible mostrar u ocultar elementos de las páginas creadas. Es útil en formularios ya que se pueden deshabilitar campos de acuerdo a ciertos criterios establecidos. Se pueden aplicar estas condiciones en atributos, campos, enlaces, unidades, celdas y grillas. (Bordonaro, 2011)
  Por ejemplo en el formulario de un nuevo reporte de gastos se crea una condición para que sea visible únicamente en el caso de recibir un submit caso contrario si es rechazado no se va a visualizar, esta condición se muestra en la figura 23.

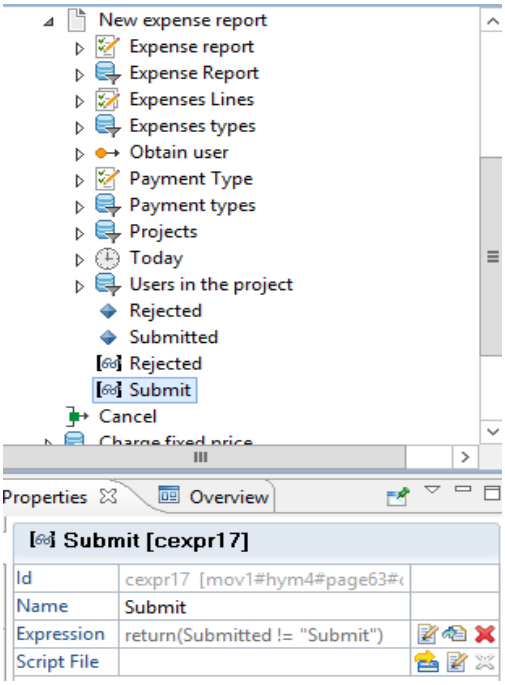

Figura 23. Condiciones de Visibilidad

Relaciones entre entidades

Para definir las conexiones entre las entidades WebRatio permite generar relaciones, las cuales se representan como en la figura 24:

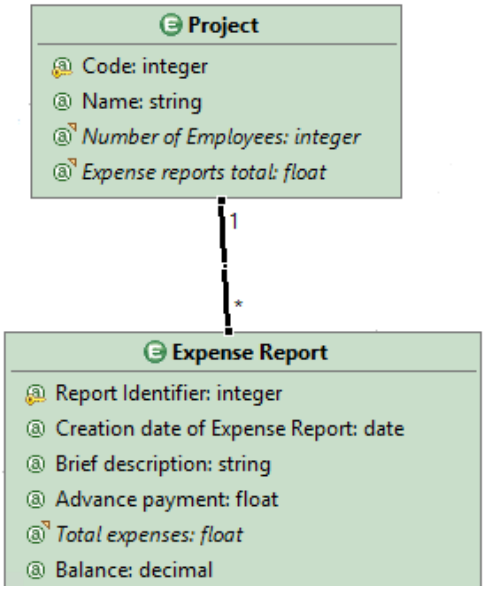

Figura 24. Relaciones entre Entidades

Las relaciones están compuestas por atributos, entre los cuales destacan: la cardinalidad que puede ser de uno a uno, de uno a muchos y de muchos a muchos y el atributo para definir si la eliminación se realiza en cascada con todos los atributos derivados. Todas las propiedades que tiene una relación se muestran en la figura 25:

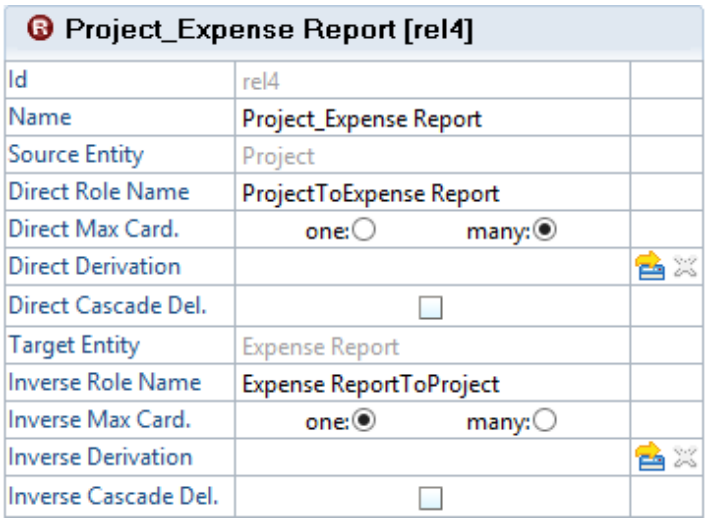

Figura 25. Propiedades de las relaciones entre clases

Las relaciones que se pueden definir en WebRatio son de asociación, y herencia. En cuanto a la herencia tiene un componente adicional el cual representa la generalización. En la figura 26 se muestra como representa la herencia WebRatio.

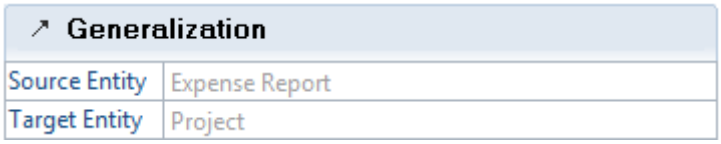

Figura 26. Generalización

Por medio de la generalización es posible conectar varias subclases con una clase padre llamada superclase. Las subclases heredan todos los atributos y operaciones de la clase padre y es posible que además tengan atributos propios.

Relación de Agentes

Para representar la relación de agentes, se puede empaquetar las clases, formando así un módulo con su respectiva seguridad.

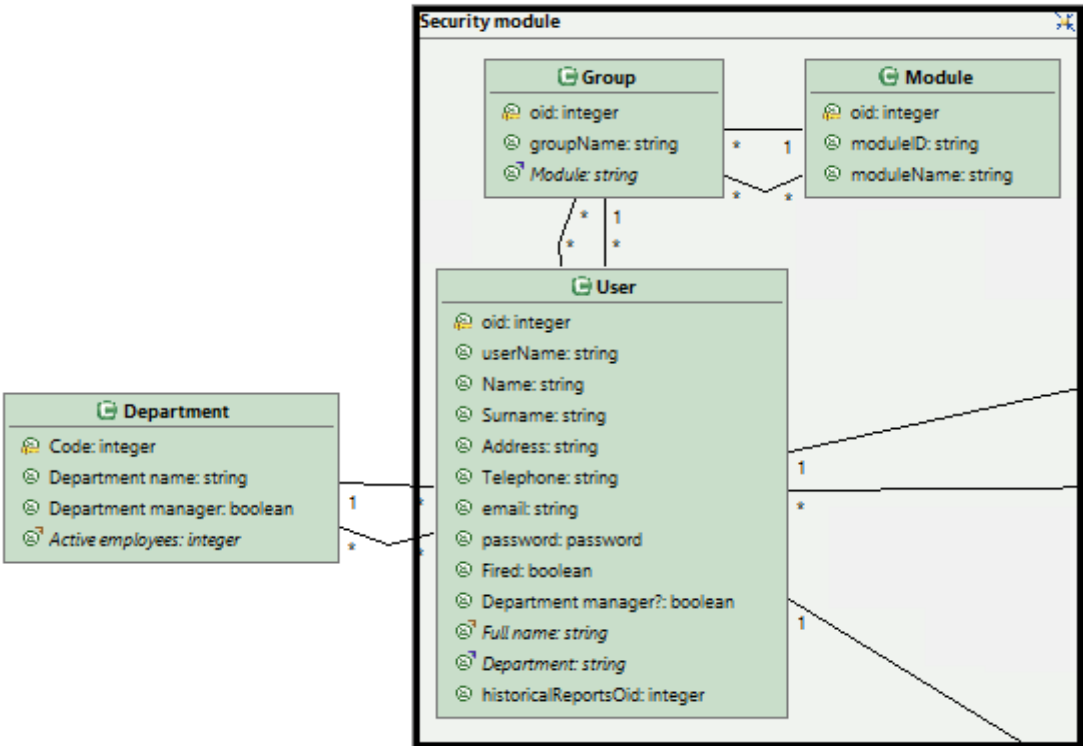

Figura 27. Relación de agentes

Para otorgar el acceso a partes específicas de la aplicación de acuerdo con el usuario que ingresa y su respectivo rol WebRatio integra automáticamente en el sistema las entidades de Modulo, Usuario y Grupo como se muestra en la figura 27.

- Modulo: Es la parte del programa que se agrupa, contiene una serie de eventos y páginas.
- Usuario: Representa el usuario que ingresa a la aplicación.
- Grupo: Representa el rol de usuario, aquí se crean las categorías de los usuarios que van a compartir los mismos permisos.

De esta manera cuando un usuario ingresa al sistema se carga únicamente la información de los módulos a los que tiene acceso. Esto se determina por el grupo al que pertenece el usuario.

4.2.2 Modelo dinámico

Triggers

WebRatio permite realizar formularios que mediante una interacción con el usuario puedan desencadenar eventos específicos. Se pueden trabajar con algunos eventos en los formularios que son: cambio, desenfoque, enfoque, teclado y pulsación de tecla. (Frigerio, 2017)

Por ejemplo en el formulario de reporte de gastos. Cuando se crea un nuevo reporte de gastos, es necesario asegurar que el reporte esté aprobado, entonces se genera un Trigger como el de la figura 28 que se ejecuta al momento de guardar el formulario y verifica esa información. Las reglas de validación que se agregan se muestran en la figura 29.

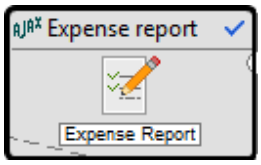

Figura 28. Trigger

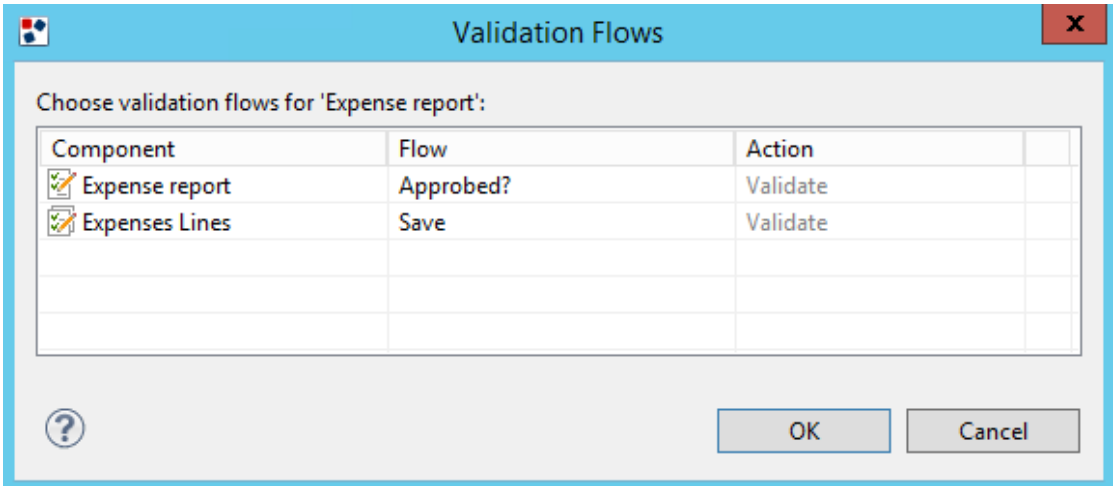

Figura 29. Validador

Transacciones y Operaciones globales

WebRatio permite ejecutar varios servicios y de diferentes clases mediante la creación de módulos, éstos tienen las propiedades básicas que son un id, nombre, comentarios, etc. Para ejecutar las diferentes acciones los módulos reciben argumentos de entrada y también son capaces de enviar datos de salida, indicando si la operación se realizó con éxito o no. En la figura 30 se muestran los módulos creados para nuestro caso de estudio.

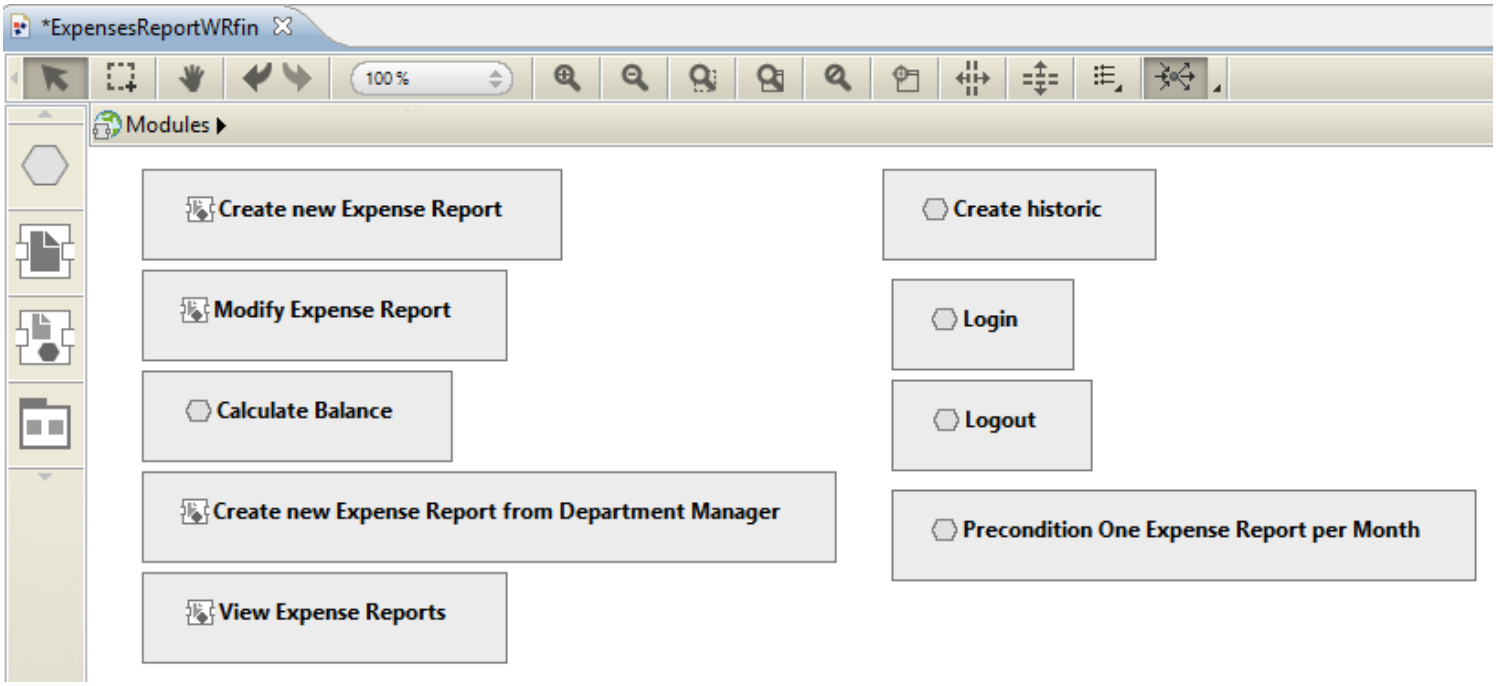

Figura 30. Operaciones Globales

4.2.3 Modelo Funcional

En WebRatio es importante primero determinar las actividades que pueden realizar cada usuario y a partir de allí se especifican los requisitos funcionales que se proporcionan a los usuarios. Para esto se descompone la aplicación en vistas de sitio y se asigna una vista a cada usuario.

WebRatio adopta un lenguaje gráfico para la especificación de requisitos funcionales de la aplicación y es el lenguaje de modelado web WebML. Este lenguaje tiene especificaciones gráficas para el diseño de la aplicación por lo tanto las herramientas son visuales.

WebML es un lenguaje visual para expresar el front-end de una aplicación web que permite al usuario navegar y administrar el sitio. Incluye primitivas para modelar aspectos tales como:

- La estructuración de la aplicación en diferentes hipertextos (llamados vistas de sitio) dirigido a diferentes grupos de usuarios o dispositivos de acceso.
- La organización jerárquica de una vista de sitio en áreas.
- Las páginas que constituyen la interfaz de la aplicación real, las unidades de contenido contenidas en cada página, con su relación con los elementos del modelo de datos (entidades y relación).
- Las operaciones y servicios que se pueden activar desde las páginas de la aplicación.
- Los enlaces que conectan páginas, unidades de contenido y operaciones para proporcionar a los usuarios las interacciones adecuadas en los navegadores (por ejemplo, anclajes, botones de radio, formularios para el ingreso de datos).
- Información de la sesión y aspectos de personalización.

4.2.4 Modelo de Presentación

Para mejorar la interacción del software con el usuario WebRatio tiene una serie de componentes que se pueden configurar y así facilitar la transferencia de la información. Estos se denominan componentes de vista.

Elementos básicos

Estos elementos permiten la conexión entre el usuario y el sistema. Existen diferentes tipos de elementos que pueden ser utilizados de acuerdo a un contexto en concreto.

Entrada

Para capturar los datos del usuario WebRatio dispone de formularios, éstos permiten el ingreso de cada uno de los atributos de la entidad relacionada. En la figura 31 se muestra un formulario utilizado para la creación de un nuevo proyecto.

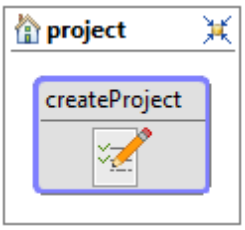

Figura 31. Entrada

En cada campo creado en el formulario se pueden añadir reglas de validación. Esto permite configurar máscaras, comparar valores, definir valores obligatorios, validar el tamaño, entre otros que se muestran en la figura 32.

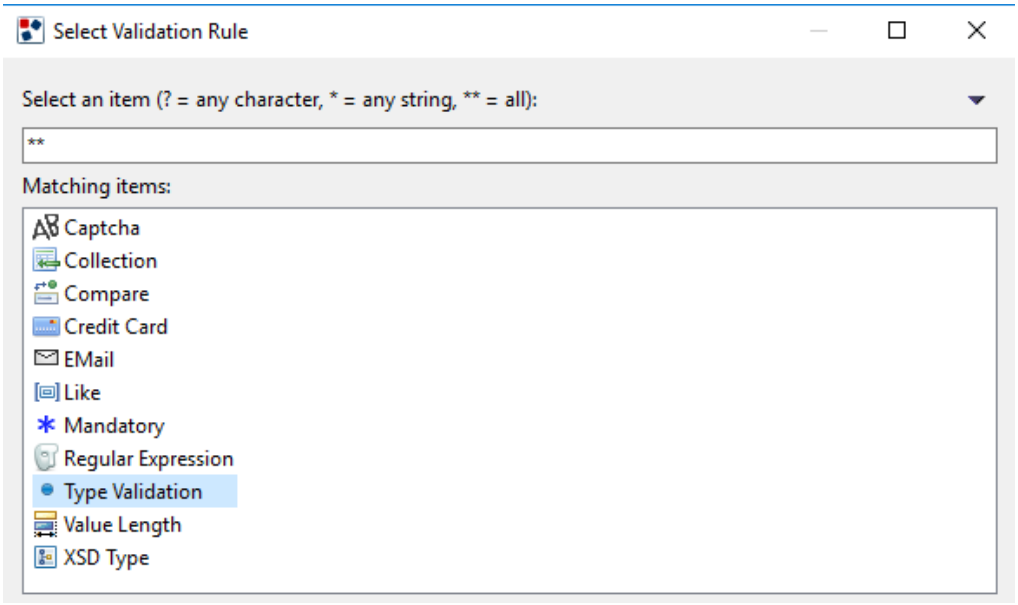

Figura 32. Reglas de validación

Como ejemplo se agrega en el campo para el ingreso del nombre del proyecto que no se puede ingresar más de 50 caracteres. Por lo tanto se agrega una regla de validación para el tamaño:

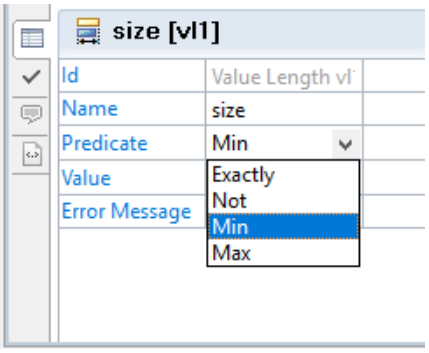

Figura 33. Validación de Tamaño

Cada regla de validación tiene propiedades básicas que son: el identificador, nombre y el mensaje de error que se presenta al usuario en caso de no cumplir con el formato. También incorpora propiedades propias que varían de acuerdo a su función, en este caso para validar el tamaño tiene cuatro tipos, que se pueden observar en la figura 33: exacto, no igual, mínimo y máximo. Además agrega una propiedad adicional que es el valor para la comparación.

En un campo del formulario se pueden agregar varias reglas de validación si así lo requiere.

Selección definida

Permite presentar al usuario un listado de valores válidos para el atributo, de los cuales puede seleccionar. WebRatio incorpora el componente "Campo de Selección" y "Campo de Selección múltiple", el primer componente permite seleccionar un solo valor de la lista mientras que en el segundo componente se pueden seleccionar varios valores.

En el ejemplo de la nota de gasto, es necesario mostrar un listado de estados que la nota puede tener para que el usuario seleccione el correcto su configuración se muestra en la figura 34:

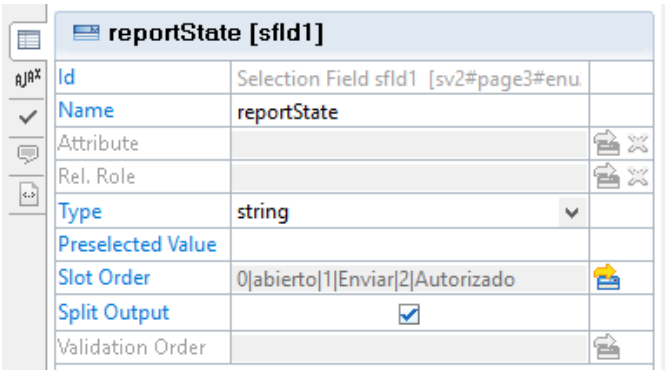

Figura 34. Campo de Selección

Una vez que se define un campo de formulario como campo de selección es necesario agregar la lista con los valores para ello se debe seleccionar el tipo de dato que tendrán los elementos y adicionalmente se puede indicar el orden con el que se van a desplegar los elementos de la lista.

Filtro

WebRatio incorpora expresiones de condición para obtener instancias de interés de una lista de objetos. Estas expresiones pueden estar compuestas por muchas condiciones dependiendo del tipo de información que se requiera filtrar, se distingues tres tipos:

- Condición de llave: Permite realizar condiciones con el atributo que es la llave principal de la entidad.
- Condición de atributos: Permite ingresar condiciones con los atributos de la entidad
- Condición de relación: Permite realizar condiciones con las entidades relacionadas

Por ejemplo se quiere obtener un listado de los empleados que han sido despedidos. Para esto se agrega un componente de lista en el formulario, esta lista está enlazada con la entidad empleado y para mostrar únicamente los despedidos se agrega una condición de atributo, indicando que obtenga todas las instancias cuyo atributo despedido sea verdadero, esta condición se muestra en la figura 35.

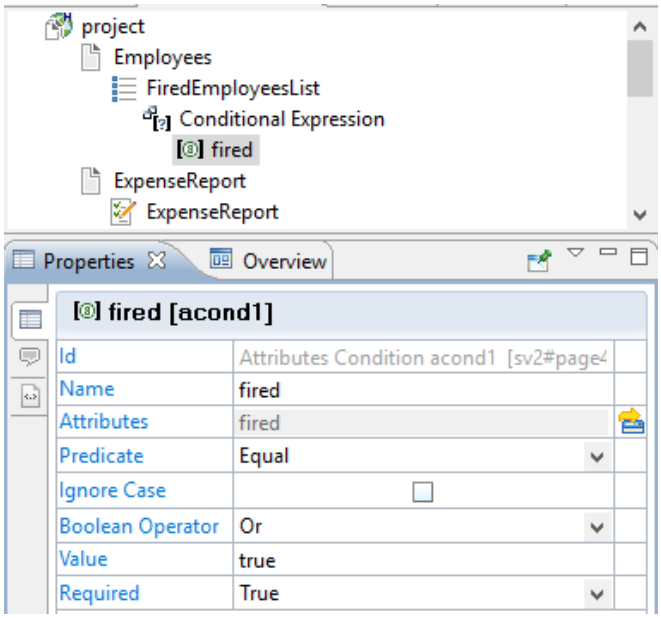

Figura 35. Filtro de Empleados

Criterio de orden

WebRatio permite configurar el orden en el que muestran los atributos de los objetos en una lista. Permite determinar si el orden es ascendente o descendente para cada atributo, esta opción se configura en las propiedades de la lista.

En la imagen 36 se indica que el listado va a estar ordenado por nombres y apellidos del empleado en forma ascendente y luego por dirección en forma descendente.

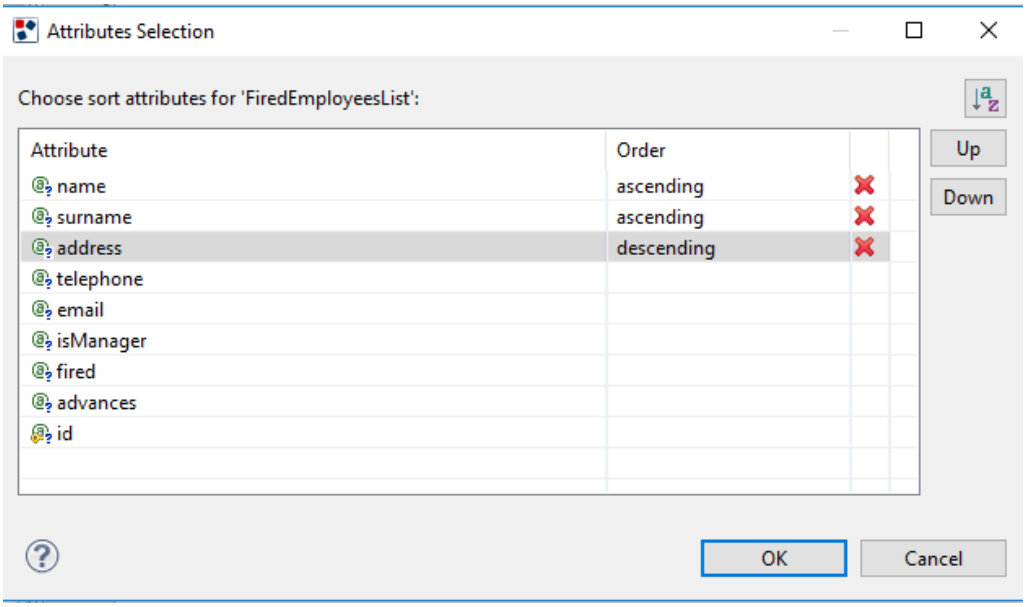

Figura 36. Criterio de orden WebRatio

Conjunto de visualización

De la misma manera que permite determinar el orden en el que muestran los objetos de una lista WebRatio también incorpora en las propiedades de la lista la posibilidad de seleccionar los atributos que se quieren mostrar de la entidad.

En el listado que se configura como se muestra en figura 37, se requiere que únicamente se muestren los datos del nombre, apellido, dirección, teléfono, email, el resto de la información del empleado permanecerá oculta.

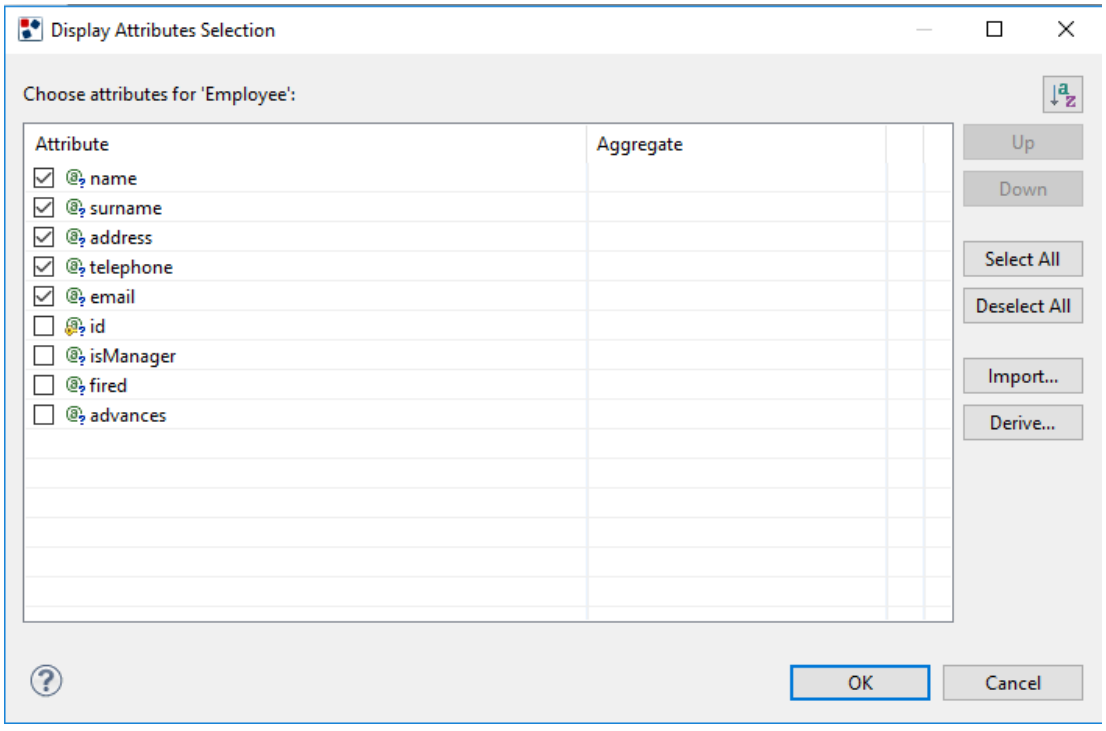

Figura 37. Visualización de atributos

Navegación

WebRatio permite configurar un flujo de navegación que es una ruta que se sigue cuando se produce un evento desencadenante. Un ejemplo en este caso es la selección de un reporte de gasto para ver sus detalles, este flujo se representa con una flecha gris dentro del modelo que conecta una fuente y un objetivo de navegación como se muestra en la figura 38. La dirección del flujo está representada por la flecha.

Por medio del flujo de navegación también se puede realizar el envío de parámetros.

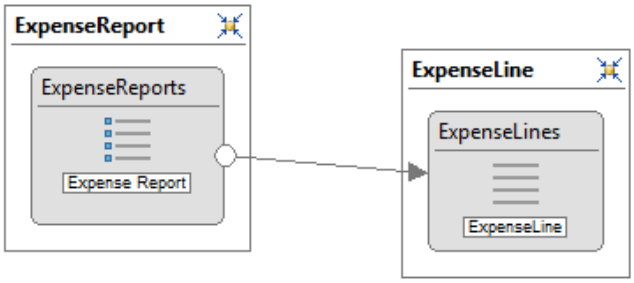

Figura 38. Flujo de navegación

Unidades de Interacción

Es importante modelar la interacción del usuario con la aplicación. Para esto es necesario identificar perfiles para los accesos dependiendo de las necesidades de cada usuario y especificando interacciones entre los usuarios y los perfiles.

WebRatio utiliza contenedores de vista para organizar la aplicación. Estos contenedores muestran contenido e interactúan con el usuario, se distinguen tres tipos que se pueden ver en la figura 39: vista de sitio, área y página. Con estos contenedores se puede modelar la estructura deseada de la aplicación (WebRatio, 2014)

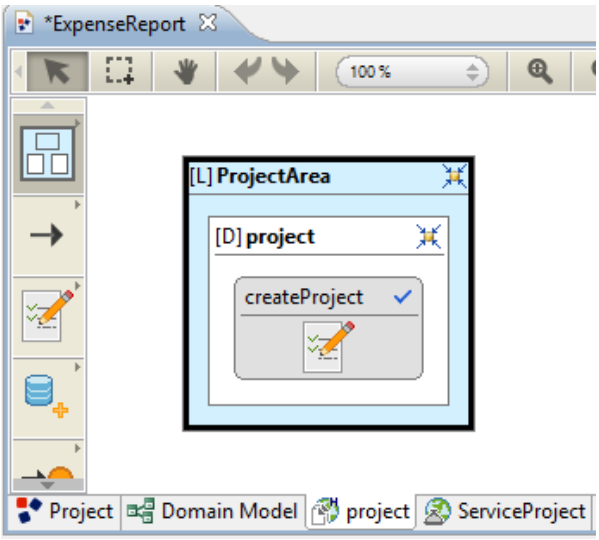

Figura 39. Unidades de Interacción

- Vista de sitio: Es una parte autónoma del flujo de interacción para los roles de usuarios específicos. Por ejemplo existe el administrador y el usuario, el administrador tiene funciones

específicas en el sistema mientras que el usuario tiene funciones más generales. Cuando ingresan a la aplicación tienen objetivos diferentes y el modelo de interacción que se asigna a cada uno es diferente, por lo tanto es necesario que cada uno tenga su vista de sitio correspondiente.

- Área: Es un contenedor formado por páginas, sirve para gestionar el contenido. Por ejemplo existen “Empleados”, “Proyectos”, “Reporte de Gastos”, entonces se puede asignar en cada área objetos con características relacionadas. El área se representa con un rectángulo azul claro y se puede crear áreas dentro de una vista de sitio o dentro de otras áreas. No existe un límite para los componentes anidados.
- Página: En la página se agregan los componentes de vista para el usuario. Se representa con un rectángulo blanco y se pueden crear dentro de una vista de sitio o en un área.

La figura 40 muestra las principales propiedades de un contenedor, éstas son:

Figura 40. Propiedades de un contenedor

- Inicio: Permite que el contenedor se muestre de manera predeterminada cuando el usuario ingresa a la aplicación.
- Punto de referencia: Indica que el elemento es parte del menú principal, está habilitado para el área y la página.
- Defecto: Permite que el componente se muestre por defecto al usuario en una determinada área.
- Protegido: Establece que se requiere autentificación para mostrar o ejecutar.
- Seguridad: Indica que necesita una conexión segura para que se pueda ejecutar.
- Localizado: Permite que la información pueda ser vista en distintos idiomas de acuerdo a la localización del usuario.

Acceso al sistema

Se puede guiar al usuario en el uso de la aplicación web creando un conjunto de rutas de navegación predefinidas. Una ruta de navegación es una secuencia de interacciones del usuario, que le permite alcanzar una característica específica de la aplicación web. El ejemplo más común de ruta de navegación es el menú principal de la aplicación web. (WebRatio, 2014)

La opción más común para crear un menú es usar los puntos de referencia definidos en la sección anterior pero se usarían las páginas individuales sin agrupación. Por esto WebRatio por medio de la integración de hojas de estilo permite modelar un menú estructurado con varios niveles.

CAPITULO 5

Desarrollo de software a partir de modelos conceptuales (Integranova)

Una vez definidas las primitivas necesarias para el desarrollo dirigido por modelos, en el capítulo anterior se detalló como soporta cada una de esas primitivas la herramienta WebRatio. En este capítulo se realiza el mismo análisis pero con la herramienta Integranova, mediante el desarrollo del mismo caso de estudio que es la gestión de notas de gasto

5.1 Integranova

Integranova permite crear modelos conceptuales que mediante una transformación pueden generar automáticamente el código fuente de las aplicaciones. En la figura 41 se muestra el logo de la herramienta.

Puede transformar aplicaciones completas generando el 100% del código y permite la inclusión de lógicas de negocio complejas a más de las funcionalidades básicas. Todo esto mediante un proceso iterativo que consiste en definir el modelo, transformarlo en código fuente y compilarlo.

El uso de esta herramienta pretende centrar toda la atención en el problema, permite a los desarrolladores dedicar todo su tiempo a la definición de los modelos junto con la lógica de negocio y no en la tecnología utilizada. La herramienta convierte automáticamente los modelos en aplicaciones a medida, fiables, funcionales, escalables, seguras y por su puesto fácil de manejar. (Integranova, 2016)

Integranova maneja una arquitectura formada por tres capas: Persistencia, Lógica de negocio y presentación. Para la transformación del modelo en código ofrece varias opciones tecnológicas que se muestran a continuación:

- Capa de Persistencia: Oracle, SQL Server, DB2, MySQL
- Capa de Lógica de Negocio: Microsoft (C#, .NET 2.0), Java (EJB 2.1)
- Capa de Presentación: Desktop (C#, .Net 2.0), Web (JSF, ASP .NET 2.0)

Imagen 41. Integranova

A continuación se analiza si esta herramienta cubre las primitivas planteadas en el capítulo 3 y en qué nivel cubre, para esto se desarrolló un sistema para la gestión de notas de gasto que se encuentra descrito en el Anexo 1 – Ejercicio planteado (Gestión de notas de gasto). Luego se procede a verificar con cada uno de los modelos definidos: objetos, dinámico, funcional y de presentación. Se puede

observar en la figura 42 que esta herramienta separa claramente los modelos mencionados lo que facilita el análisis de cada uno de ellos.

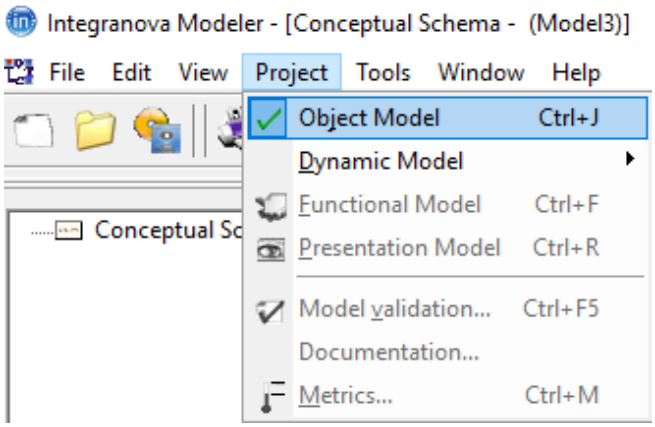

Figura 42. Modelos en Integranova

5.1.1 Modelo de objetos

La capa de persistencia de Integranova contiene toda la información de las entidades y los atributos que se definen en el sistema, también se incluye las relaciones que existen entre estos.

Clases

Para agrupar objetos con características en común en Integranova se definen las clases, en la figura 43 se puede observar la creación de la clase Empleado. Las clases se componen de los siguientes atributos:

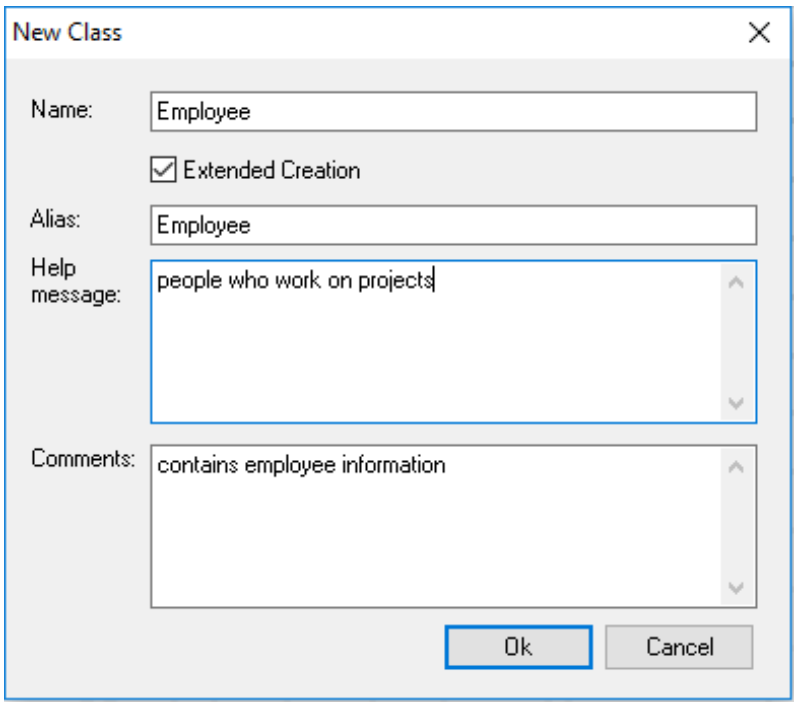

Figura 43. Propiedades de la Clase

- Nombre: Es el identificador único de la clase

- Alias: Es un nombre nemotécnico que se le da a la clase y permite reconocerla de manera más fácil
- Mensaje de ayuda: Información relacionada a la clase que sirve para orientar al usuario sobre el contenido de la entidad.
- Comentarios: Información interna que es utilizada para la documentación del sistema

Atributos

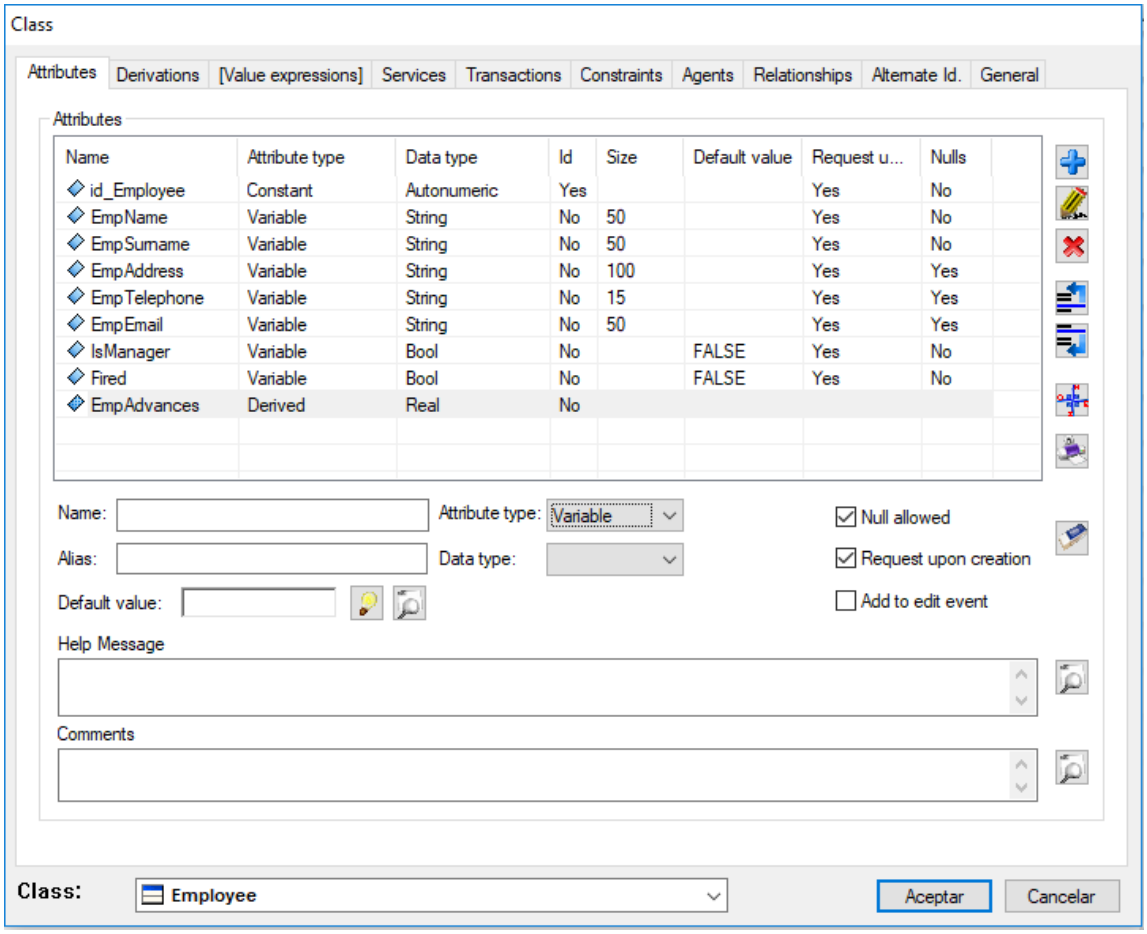

Figura 44. Atributos de Empleado

Al momento de crear una clase en Integranova, se crea automáticamente un atributo que es el identificador de la clase, este atributo es de tipo auto numérico, también brinda la posibilidad de crear nuestro propio identificador de acuerdo a las necesidades.

El contenido principal de una clase son sus atributos, en la figura 44 se muestran los atributos para la clase empleado. Integranova tiene una serie de propiedades que facilitan el modelado de los objetos del mundo real, entre las cuales están:

- Nombre: Sirve para identificar de manera única al atributo
- Alias: Es un nombre nemotécnico que permite reconocer al atributo de una manera más fácil
- Tipo: Es el tipo de atributo, no hace referencia al tipo de información que contiene, se identifican tres tipos:

    - Constante: Cuando la información de un atributo no se puede modificar después de haber sido almacenada, se utiliza este tipo de atributo para los identificadores o llaves primarias de la clase.
    - Variable: Permite modificar la información después de haber sido almacenada. Este tipo es utilizado por la mayoría de atributos que contienen información que se debe actualizar.
    - Derivado: Este tipo de atributo se define para aquellos campos que requieren información de otros atributos que pueden ser de la misma clase o de otra. Puede almacenar operaciones con atributos.
- Tipo de dato: Esta propiedad define el tipo de información que va a ser almacenada en este atributo. Integranova soporta los principales tipos como son: auto numérico, blob, booleano, fecha, hora, caracteres, texto, enteros, reales, naturales.
- Tamaño: En el caso de que el tipo de dato sea cadenas de texto se habilita la propiedad de tamaño para limitar el número de caracteres permitidos.
- Valor por defecto: Permite que el campo tenga cierta información por defecto como sugerencia para el usuario.
- Requerido: Indica si el atributo es necesario al momento de crear una instancia nueva de la clase.
- Null: Esta propiedad indica si el atributo puede contener valores en blanco.
- Añadir al evento editar: Esta propiedad permite que el atributo se muestre únicamente cuando se edita y no en el momento de su creación.
- Mensaje de ayuda: Es un texto que se muestra al usuario para brindar información acerca del atributo.
- Comentarios: Útil para la documentación.

Derivaciones

Si un atributo es creado con el tipo derivado se muestra automáticamente en la ventana de derivaciones. Esta ventana permite obtener la información que se va a almacenar en el atributo.

Las derivaciones de un atributo tienen las siguientes propiedades:

- Atributo: Indica el atributo que contiene la información
- Formula de derivación: El mecanismo por el que se obtiene la información, está compuesto de condición y fórmula.
    - Condición: Permite definir una condición que se ejecuta antes de la fórmula, si la condición es verdadera pasará a ejecutar la fórmula.
    - Formula: Permite calcular el valor del atributo.
- Comentarios: Son utilizados para la documentación.

Como ejemplo se toma la clase Línea de Gastos, en esta clase se define un atributo “Total” y se selecciona de tipo Derivado. En la pantalla de derivaciones se selecciona el atributo y en la fórmula se indica que el valor de este atributo se calcula multiplicando el atributo precio por el valor que tenga en el atributo unidades. Se puede observar en la figura 45.

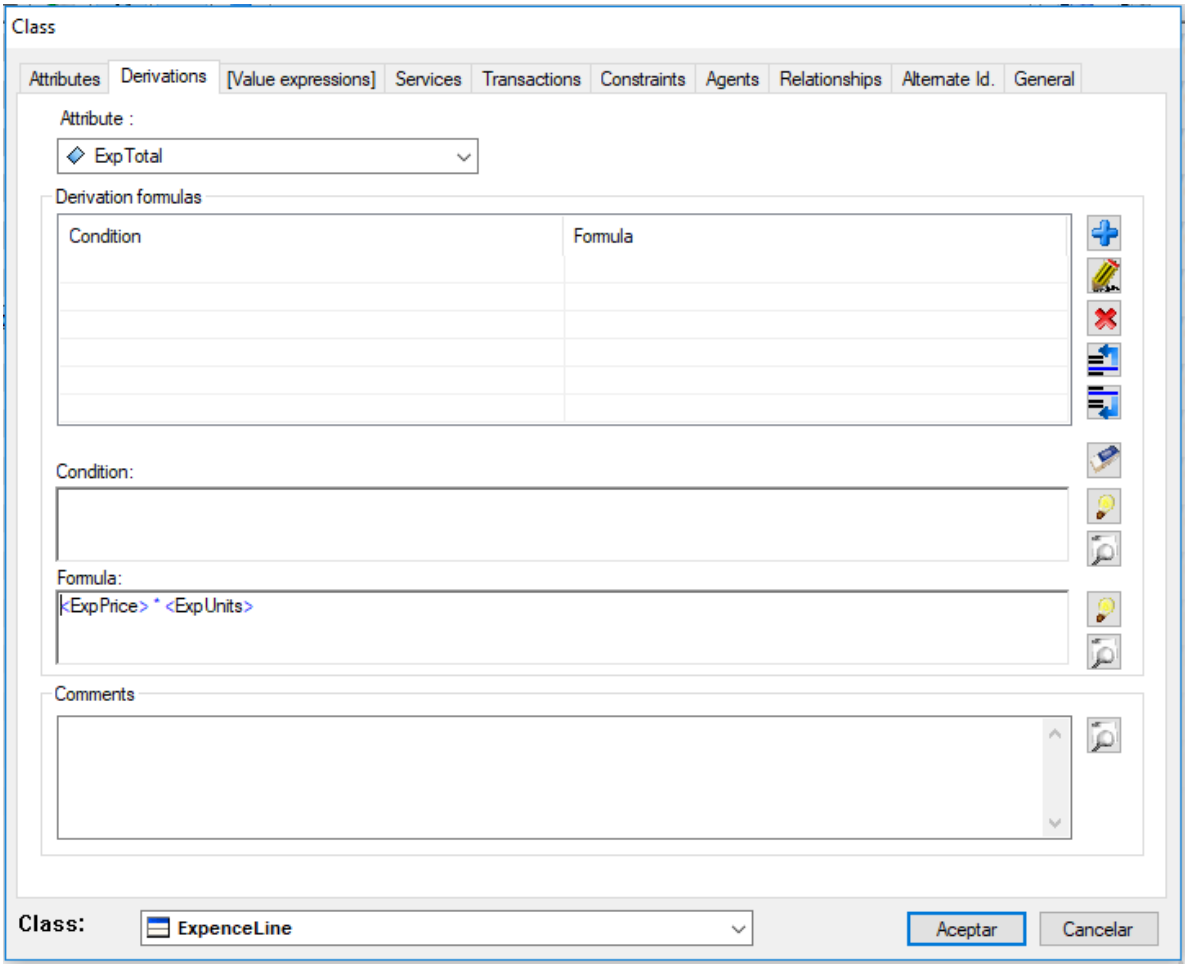

Figura 45. Derivación

Servicios

En cada clase que se genera Integranova permite incorporar servicios para definir su funcionalidad. Estos servicios son considerados como unidades de proceso, se distinguen dos tipos: atómicos (eventos) y moleculares (transacciones, operaciones) como se muestran en la figura 46.

Figura 46. Servicios Integranova

Eventos

Los eventos que están definidos en Integranova se muestran en la figura 47 y son: creación, modificación y eliminación.

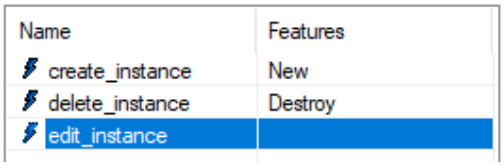

Figura 47. Eventos Integranova

- Crear: permite crear una nueva instancia de la clase
- Eliminar: Sirve para eliminar una instancia de la clase
- Editar: Permite modificar los datos almacenados en una clase

Los eventos tienen las siguientes propiedades:

- Nombre: Para identificar de manera única al evento
- Alias: Nombre nemotécnico para reconocer al evento
- Mensaje de Ayuda: Sirve para dar información mayor información relacionada al evento
- Comentarios: Utilizados para la documentación

Argumentos

Al momento de construir los servicios, Integranova ofrece la posibilidad del paso de argumentos necesarios para realizar una operación. Como ejemplo en la figura 48 se muestra el evento de creación de un nuevo empleado, este evento recibe como argumentos todos los atributos que forman parte de la clase empleado.

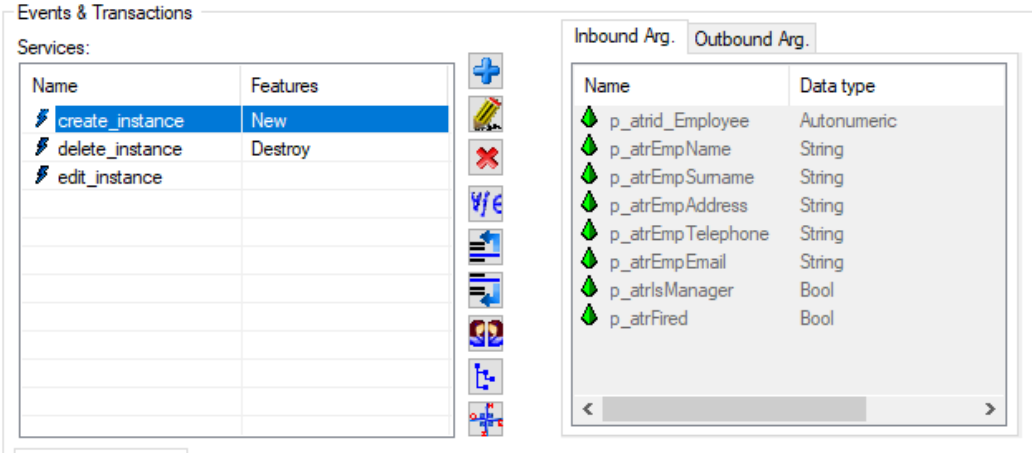

Figura 48. Argumentos

Se establecen dos tipos de argumentos:

- Entrada: información que se recibe y es utilizada para realizar el servicio
- Salida: información resultante que se obtiene después de la ejecución del servicio, este tipo de argumento permite devolver un valor.

La figura 49 muestra las propiedades de un argumento y son:

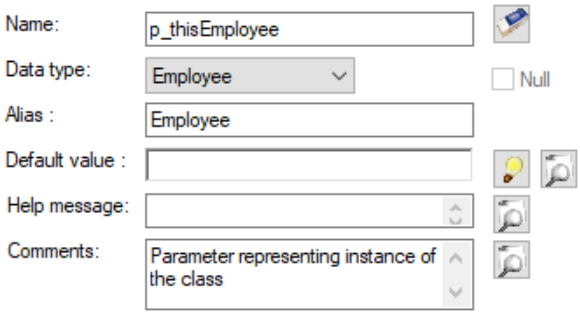

Figura 49. Propiedades del Argumento

- Nombre: Permite identificar al argumento
- Tipo de dato: Establece el tipo de información que contiene
- Nulo: Indica si puede venir en blanco.
- Alias: Nombre nemotécnico para identificarlo de una manera más fácil
- Valor por defecto: Valor con el que se inicializa
- Mensaje de ayuda: Contiene información relacionada al argumento
- Comentarios: Se utilizan para la documentación

Transacciones

En Integranova los eventos representan una unidad de ejecución. En ciertos casos es necesario agrupar varias unidades de ejecución, para esto se utilizan las transacciones locales. Por ejemplo un evento es eliminar un empleado pero si necesito eliminar un empleado y todos sus proyectos relacionados entonces necesito crear una transacción.

Integranova reconoce dos tipos de transacciones: locales y globales.

- Transacciones Locales: Puede incluir eventos y otras transacciones. Se ejecutan como una sola unidad de procesamiento. Se crean en la misma pestaña de los eventos como se muestra en la figura 50 y tienen las siguientes propiedades:

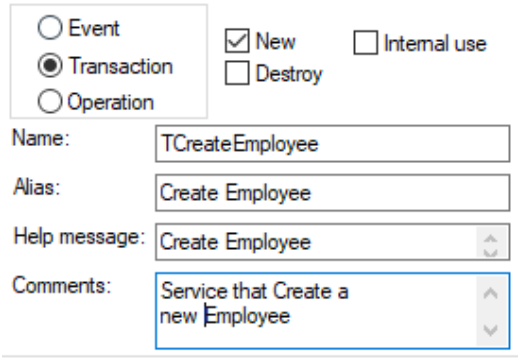

Figura 50. Transacción Local

  - Nombre: para identificar a la transacción
  - Alias: Nombre nemotécnico para identificar de mejor manera a la transacción
  - Mensaje de Ayuda: contiene información relacionada a la transacción
  - Comentarios: Utilizados para la documentación
  - Argumentos: información necesaria para la ejecución de la transacción
  - Formula de transacción: describe lo que debe hacer la transacción, en la figura 51 se observa la creación de un nuevo empleado.

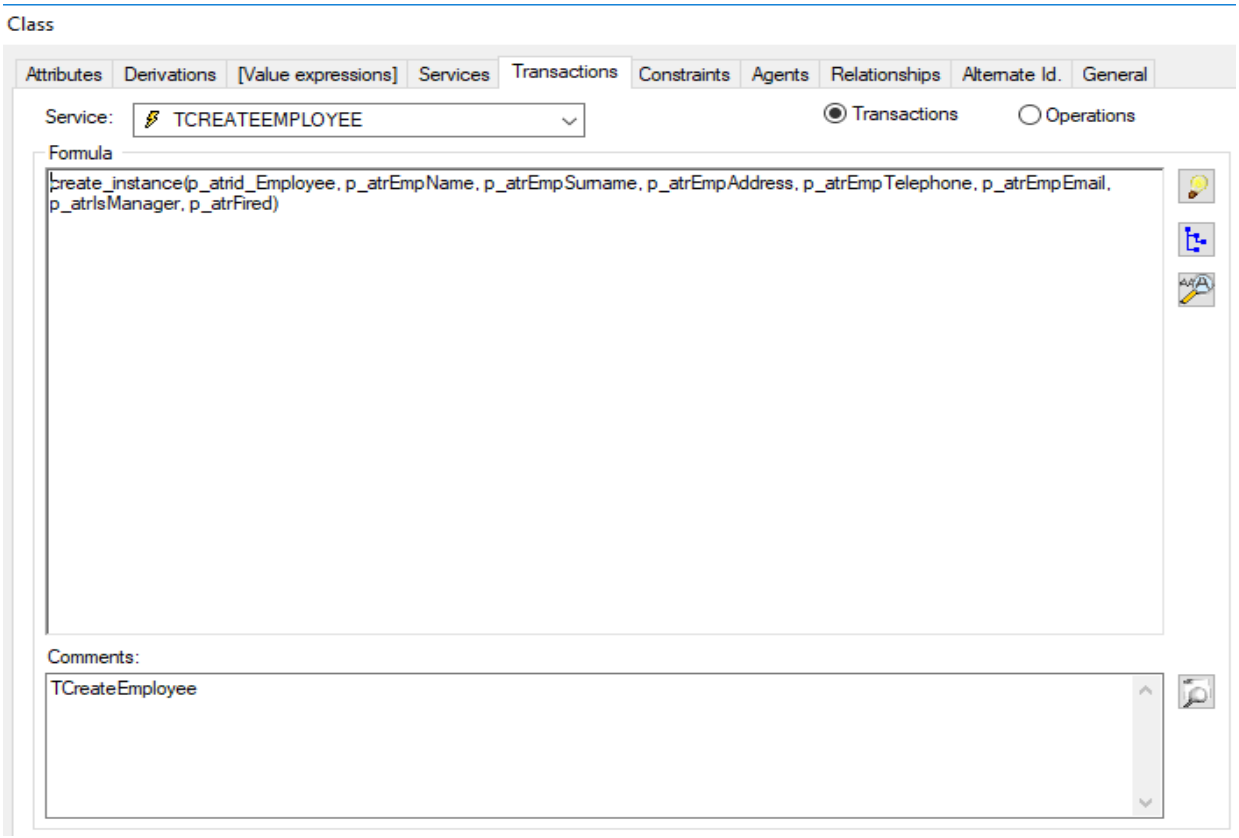

Figura 51. Formula de transacción

Para crear la fórmula de la transacción se puede integrar diversos componentes que se pueden observar en la figura 52, entre los principales están: atributos, otros servicios, funciones, operaciones, etc.

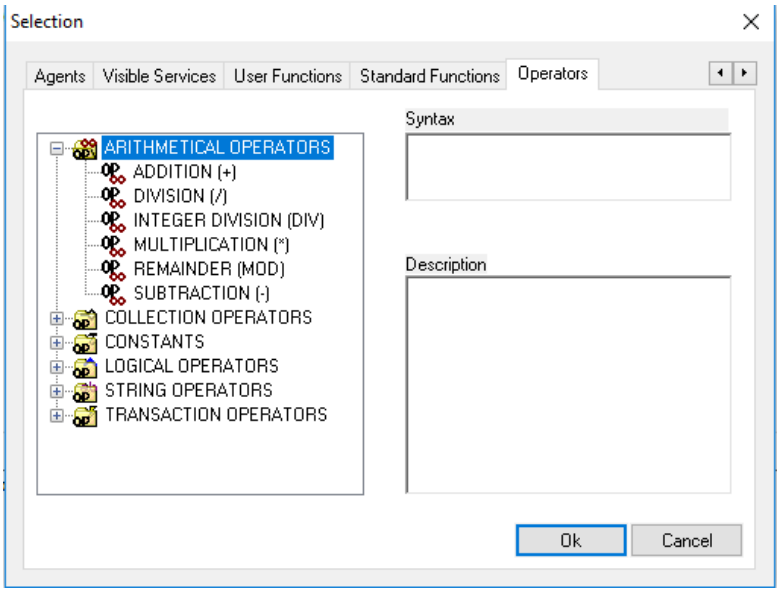

Figura 52. Operadores en Formula de la Transacción

- Transacciones globales: son similares a las locales pero no se limita a objetos relacionados entre sí, su alcance es a todo el modelo, estas transacciones necesitan una fórmula que incluye

cualquier del modelo, en Integranova se definen directamente desde el menú, como se muestra en la figura 53.

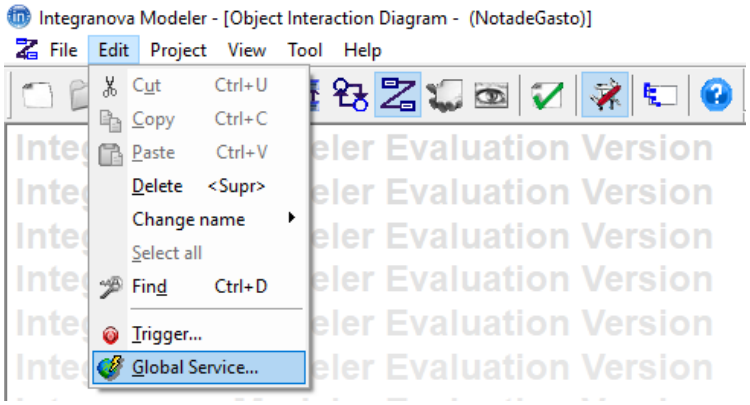

Figura 53. Transacciones globales

Las transacciones globales tienen las siguientes propiedades: nombre, alias, mensaje de ayuda, comentarios y argumentos. Al igual que en las transacciones locales es necesario establecer una fórmula que es lo que describe lo que hace la transacción.

Operaciones

Las operaciones son similares a las transacciones en cuanto a la definición como se muestra en la figura 54. Su diferencia está en la ejecución ya que si se produce un error durante la ejecución, la operación continúa con la siguiente parte de la formula en lugar de deshacer todos los cambios realizados. Una vez terminada la ejecución de la operación no se envía una respuesta de ejecución correcta o incorrecta

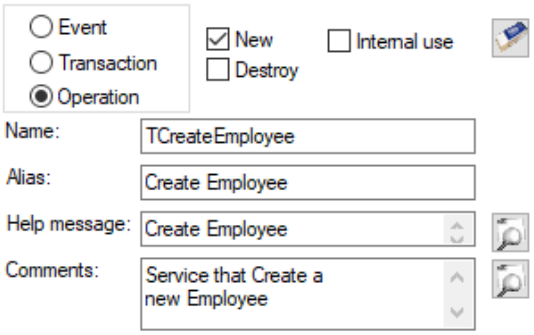

Figura 54. Operación

Restricciones de Integridad

Para establecer las restricciones del sistema Integranova tiene dos formas: Restricciones en términos del estado de un objeto y restricciones en términos de condiciones.

- Restricciones en términos del estado de un objeto: también conocidas como restricciones de Integridad, se analizan después de la ejecución de un servicio. Son utilizadas para restringir información por ejemplo en la figura 55, en la nota de gasto un empleado puede ver solo sus notas de gasto y no es correcto que se muestren notas que no le corresponden.

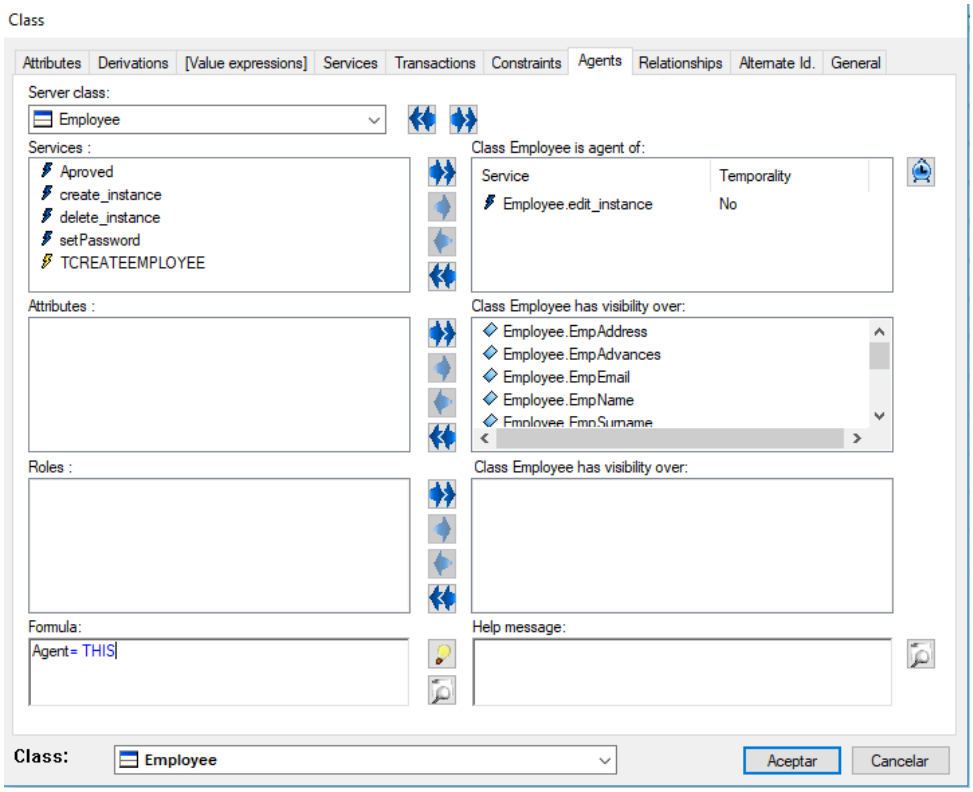

Figura 55. Restricciones de Integridad

Estas restricciones se definen en la visibilidad horizontal. Para este ejemplo la fórmula sería:

Agent = THIS

En esta fórmula, hay dos palabras reservadas, que significan:

- Agent: representa al empleado conectado en el sistema.
- THIS: representa la clase de servidor del gasto. La instancia que está ofreciendo los atributos, servicios y roles.

Entonces, se define la siguiente fórmula (Employee = THIS) con esto se informa al sistema que muestre la información que pertenece al cliente que está actualmente conectado en la aplicación.

- Restricciones en términos de condiciones: conocidas como precondiciones, deben aplicarse antes de ejecutar el servicio. Se pueden definir precondiciones para: eventos, transacciones u operaciones. Su alcance está limitado al servicio en el que fueron definidas. Las precondiciones consisten en una fórmula de condición bien formada que solo si se cumple se ejecuta el servicio caso contrario devuelve un error.

Relaciones entre clases

Para enlazar las diferentes clases Integranova permite generar relaciones de asociación y de herencia.

Relaciones de Asociación

Para definir las relaciones de asociación se distinguen tres tipos como muestra la figura 56: Asociación simple, Agregación y Composición.

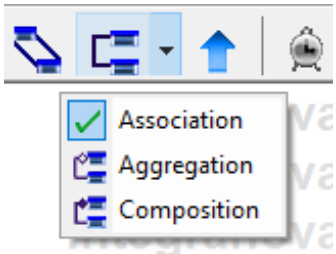

Figura 56. Relación de Asociación

Asociación

Son conexiones entre clase. La figura 57 muestra la relación entre empleados y departamentos estableciendo que un empleado pertenece a un departamento, mientras que un departamento puede tener 0 o más empleados.

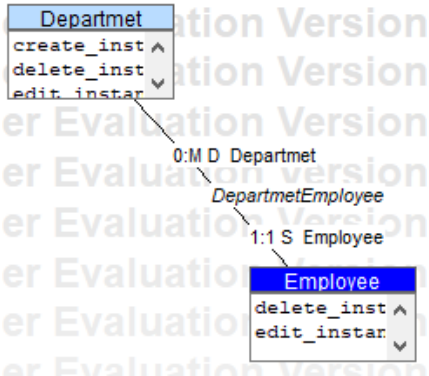

Figura 57. Relación de Asociación

Esta relación tiene las siguientes propiedades:

- Nombre: Sirve para identificar la relación
- Cardinalidades: Permite registras las cardinalidades mínimas y máximas para cada una de las clases relacionadas. Como ejemplo en la figura 58 el empleado puede pertenecer solo a un departamento mientras que el departamento puede tener muchos empleados.
- Nombres de Roles: Se refiere a los nombres de las clases que se relacionan
- Estática: significa que una vez que se crea la relación no se va a cambiar. Por ejemplo una vez que se crea el empleado va a pertenecer a un departamento en concreto si en el proceso este empleado se cambia de departamento es necesario que definirla como dinámica, si nunca se va a cambiar quedaría por defecto estática.
- Comentarios: Utilizados para la documentación

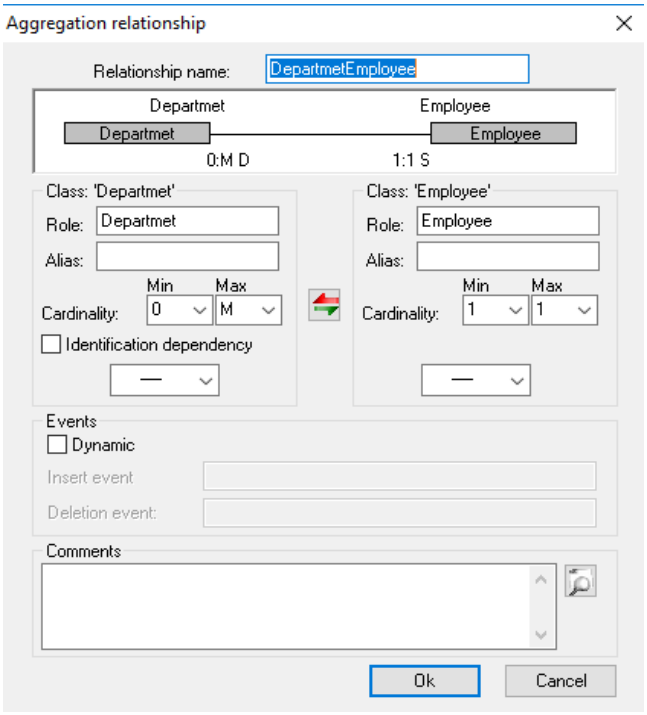

Figura 58. Propiedades de la Asociación

Para las relaciones de Agregación y Composición se sigue el mismo proceso que de una asociación simple. En ventana de propiedades se selecciona el tipo de dependencia como se muestra en la figura 59.

- Línea: Asociación simple
- Rombo Blanco: Agregación
- Rombo Negro: Composición

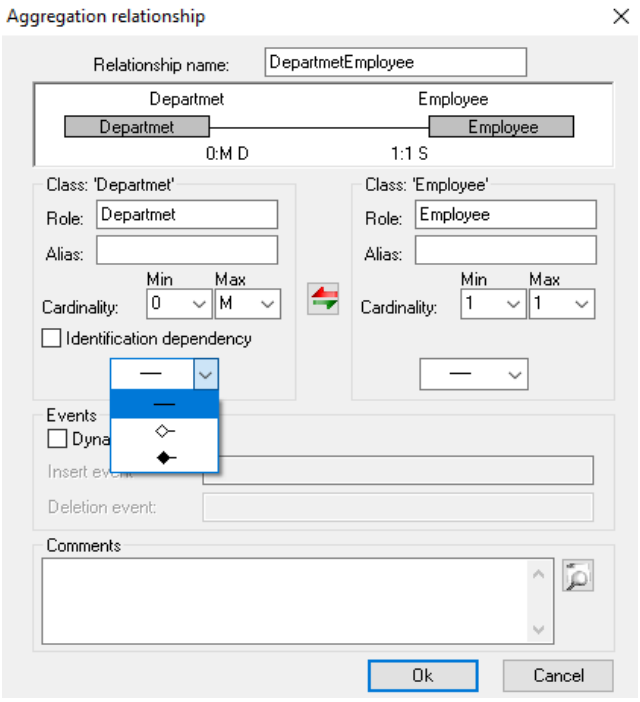

Figura 59. Tipos de Relaciones de Asociación

Agregación

Las relaciones de agregación son relaciones estructurales entre dos clases. La visibilidad de los atributos y servicios de cada clase con respecto a la otra es bidireccional. (Integranova S.A, 2011) Una clase forma parte de otra. Como ejemplo en la figura 60 se agrega la clase línea de gasto a la clase reporte de gasto.

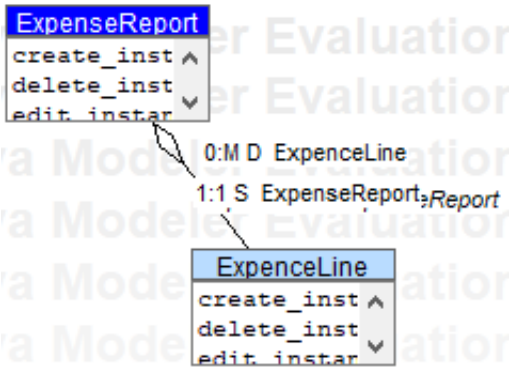

Figura 60**. Relación de Agregación**

Composición

La relación de composición está formada por una clase dependiente y otra independiente. La clase dependiente no puede existir sin la clase dependiente y la necesita para ser completada, pero la clase independiente puede existir por sí misma. La figura 61 muestra la relación de composición la clase jefe de departamento con la clase empleado.

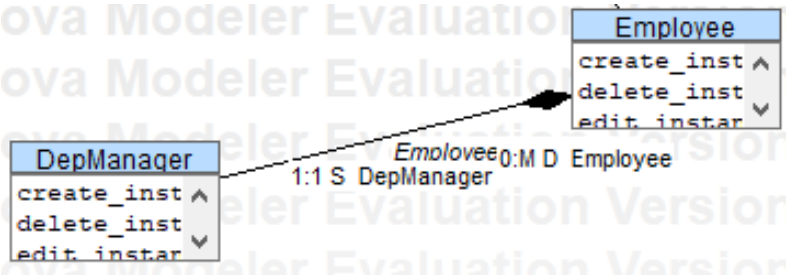

Figura 61. Relación de Composición

Relaciones de Herencia

En esta relación una clase hereda todos los atributos de la clase padre pero no viceversa. Se puede observar en la figura 62 en donde la clase departamentoH hereda de la clase departamento.

La relación de herencia tiene las siguientes propiedades:

- Clase: Indica la clase a la que se va a agregar la relación
- Clases Relacionadas: Se debe seleccionar la segunda clase para completar la relación
- Evento portador: Es el evento definido en la clase padre.
- Evento liberador. Es el evento o eventos definidos en la clase padre o hija.
- Tipo: Este es la propiedad que determina el tipo de herencia y puede ser de Especialización (descendiente) o de Generalización (Ascendiente)

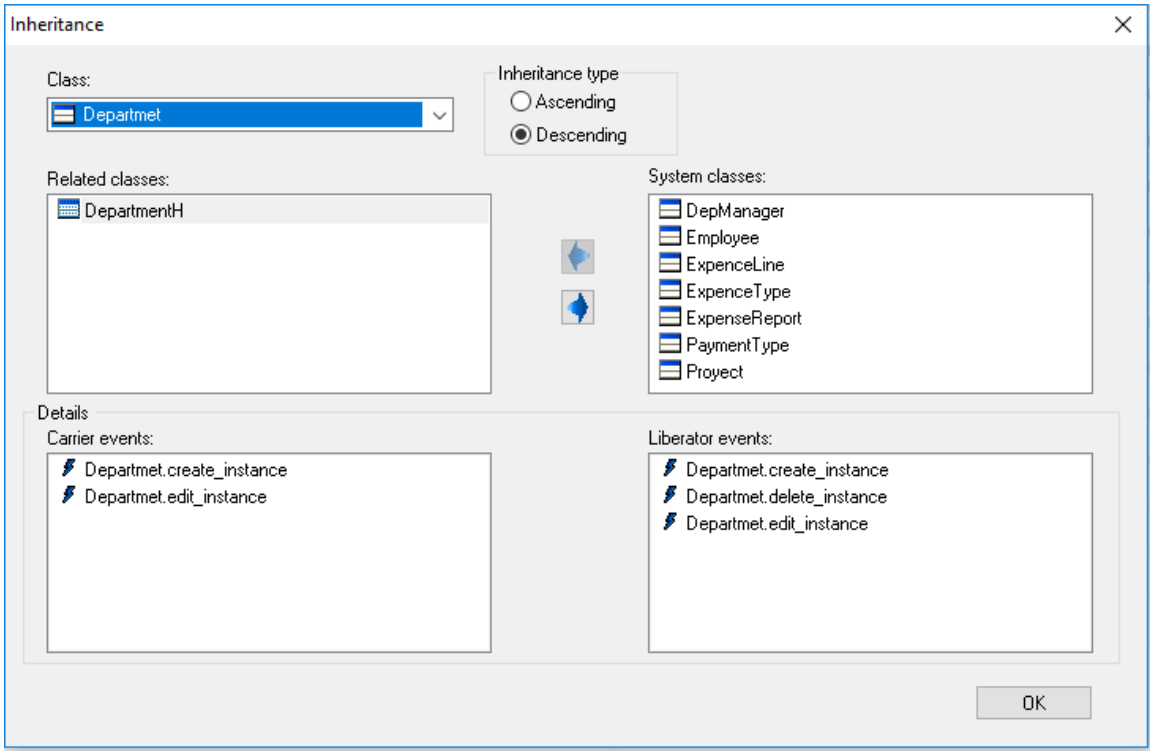

Figura 62. Relación de Herencia

Relación de Agentes

Integranova permite desarrollar un sistema cuyos procesos sean validados y utilizados por los agentes que están directamente conectados. Cuando un agente ingresa al sistema solo tendrá acceso a los servicios que pueda ejecutar y a los atributos que tenga permiso de visualizar. Entre las principales propiedades están:

- Clase: Determina la clase en la que está definiendo el agente
- Servicios: Indica los servicios que puede ejecutar el agente
- Atributos: Se selecciona los atributos que puede visualizar el agente
- Roles: Determina las relaciones de esa clase a las que puede acceder el agente
- Formula: Limita el acceso al agente

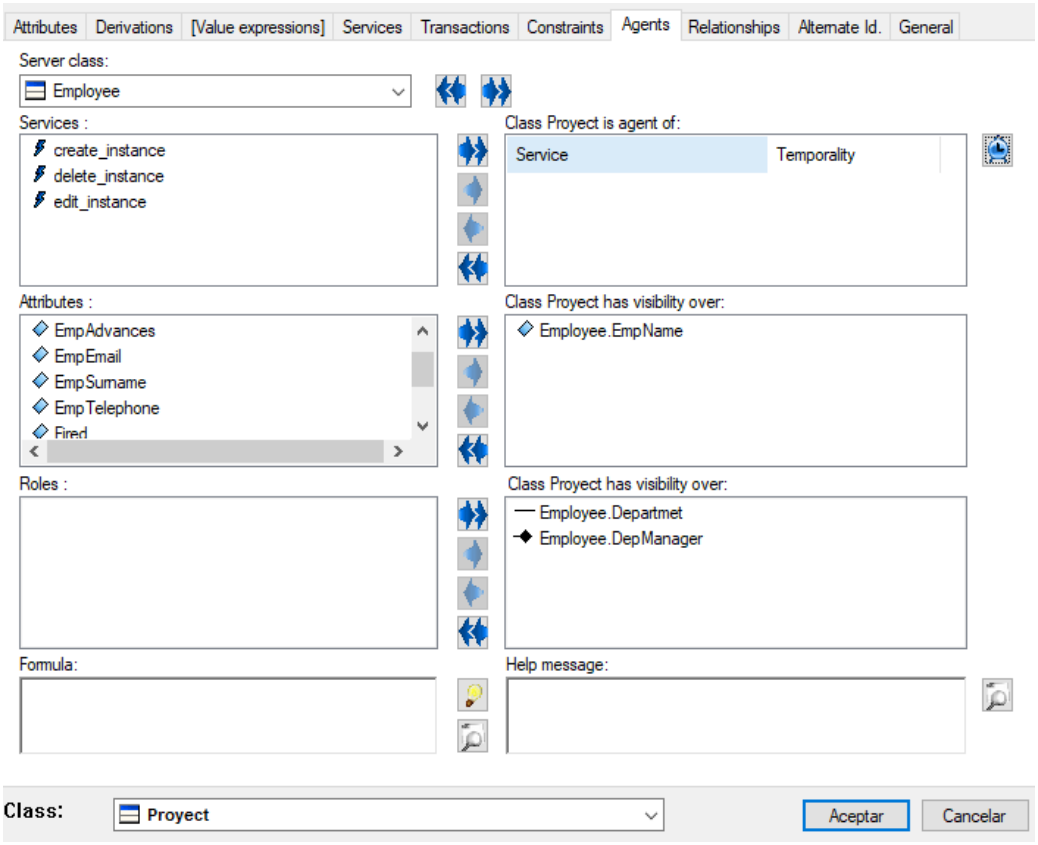

Figura 63. Relación de Agente

En la figura 63 se muestra la creación de una relación de agente en la tabla empleado, definiendo que únicamente estará visible el atributo nombre y tiene acceso a la relación del empleado con el departamento y el jefe del departamento.

5.1.2 Modelo dinámico

Para determinar la secuencia de los servicios definidos en una clase o las interacciones con otras clases Integranova incorpora los diagramas de transición de estados y de interacción de objetos desde su menú presentado en la figura 64.

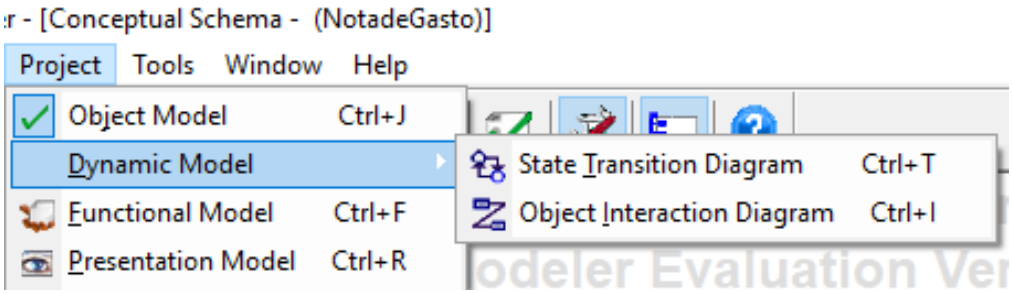

Figura 64. Modelo dinámico

Diagrama de Transición de estados

Integranova permite generar automáticamente el diagrama de transición de estados que permite obtener la secuencia en la que se ejecutan los servicios que definidos en una clase. En la figura 65 se muestra este diagrama para la clase empleado, con sus servicios de creación, eliminación y modificación.

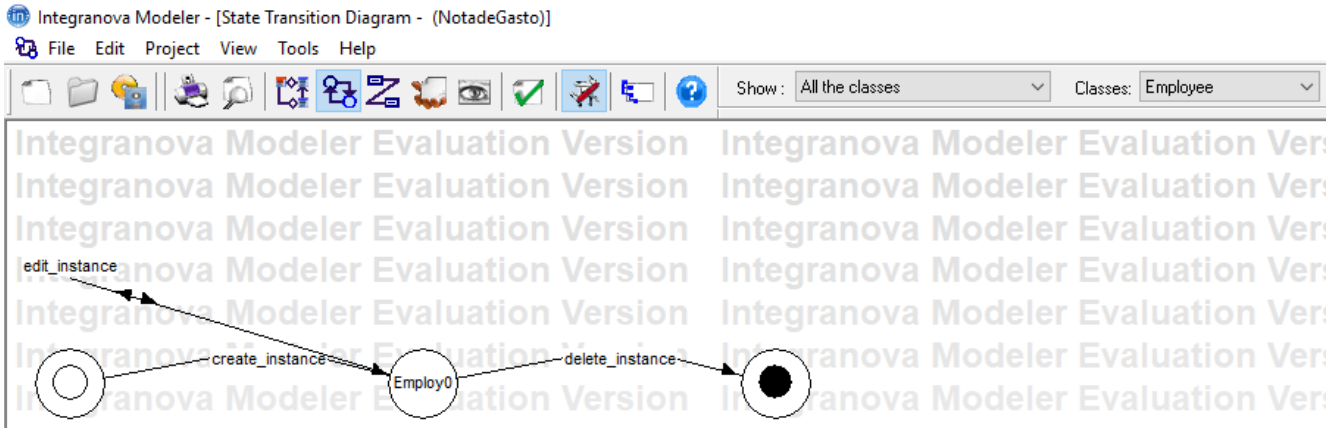

Figura 65. Diagrama de Transición de Estados

Diagrama de interacción entre objetos

En este diagrama se muestran todos los triggers y operaciones globales creadas para cada una de las clases. La figura 66 muestra un disparador creado para la clase línea de gasto.

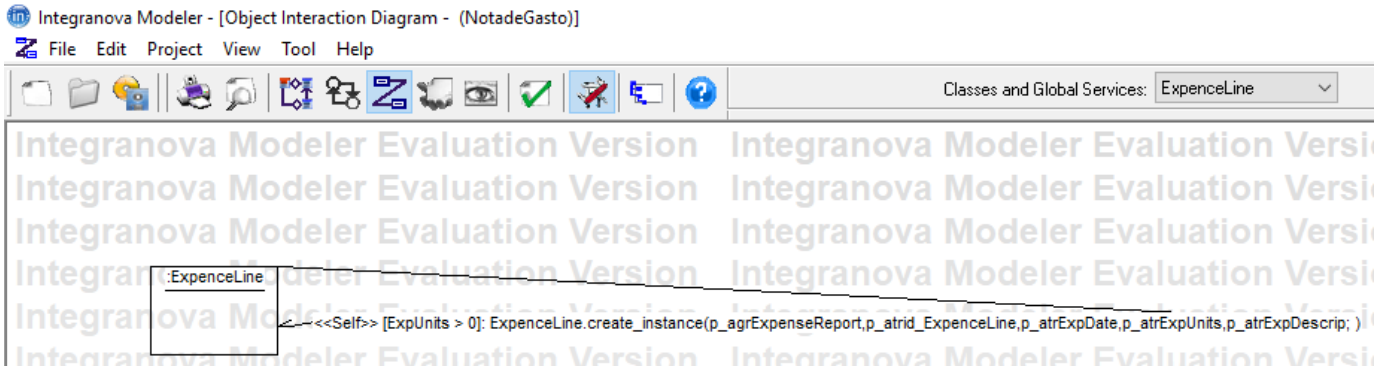

Figura 66. Diagrama de Interacción de Objetos

5.1.3 Modelo Funcional

Integranova ofrece varias opciones para expresar invariantes y utiliza constructos básicos presentes en cualquier lenguaje de programación: secuencia, iteración, decisión y recursión.

Para esto se integra el modelo funcional mostrado en la figura 67, que permite una definición detallada del comportamiento de las operaciones.

Entre sus principales propiedades están las categorías que permite definir para la evaluación:

- Estado: Se utiliza para para dar un valor al atributo después de la ejecución de un evento sin depender del estado que haya tenido anteriormente.
- Cardinal: Permite que el atributo tenga un nuevo valor después de un evento pero este nuevo valor depende del valor que tenía el atributo antes de la ejecución del evento.
- Situación: En este caso el atributo toma un valor de un conjunto de valores definidos.

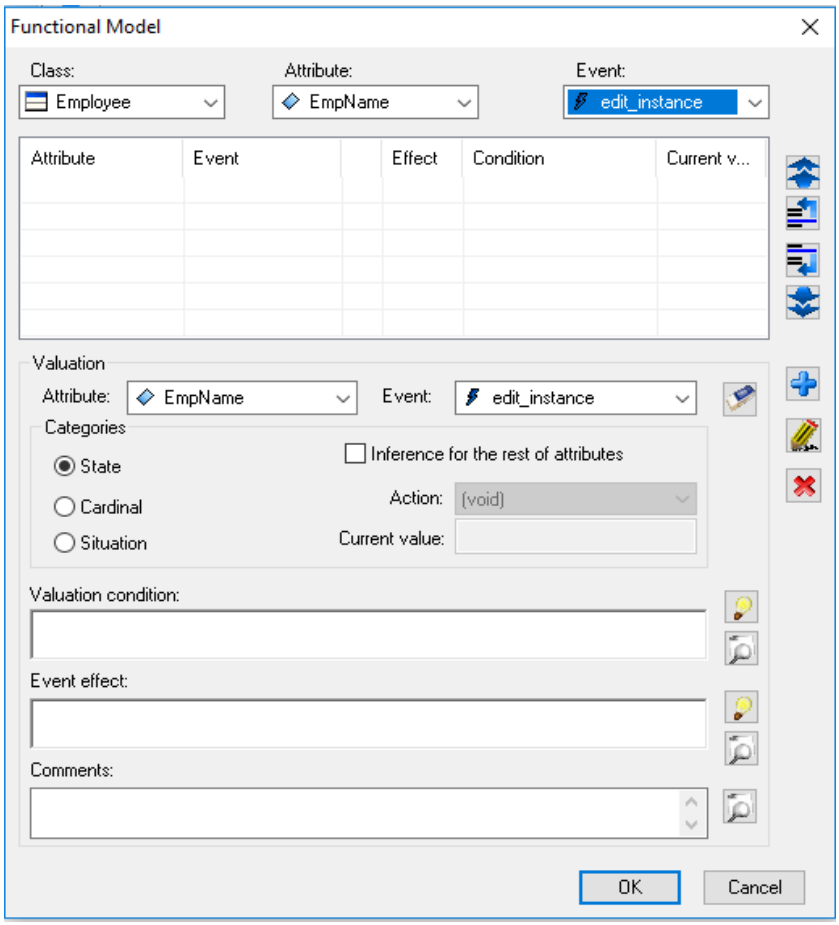

Figura 67. Modelo Funcional

5.1.4 Modelo de Presentación

Para establecer la relación del sistema con el usuario Integranova define un modelo de presentación que cubre todos los aspectos de navegación e iteración, se puede acceder a este modelo desde el menú como se indica en la figura 68.

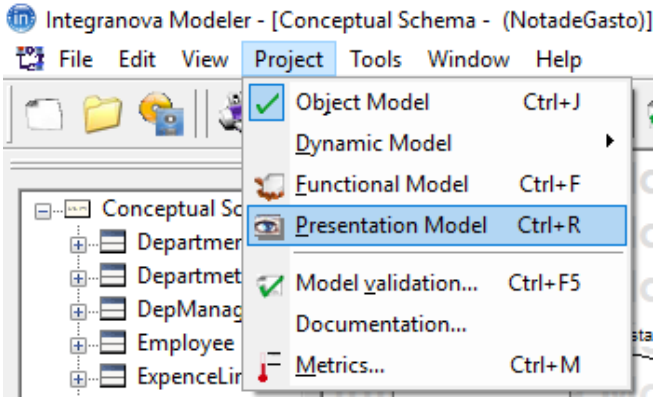

Figura 68. Modelo de Presentación

Elementos básicos

Entrada

Permite capturar la información que ingresa el usuario. En la figura 69 se crea un campo para el ingreso del nombre del proyecto.

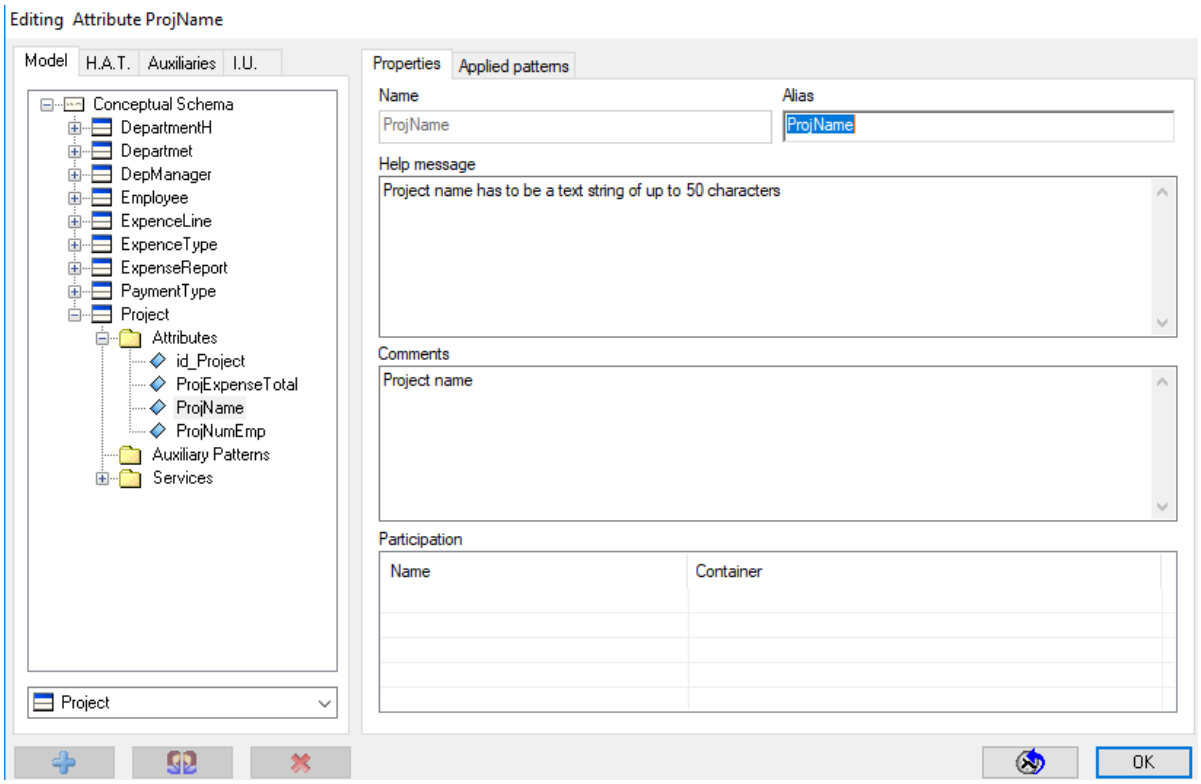

Figura 69. Entrada

Para cada atributo de las clases, Integranova permite incorporar máscaras y establecer un rango de valores que pueden tomar dependiendo de las necesidades. De igual manera permite agregar mensajes de ayuda para una mejor interacción con el usuario.

Para agregar un nuevo patrón se debe tener en cuenta:

- Nombre: Es el identificador del patrón

- Mensaje de ayuda: Brinda información al usuario
- Comentarios: Se utilizan para la documentación
- Participación: Indica el atributo al que se asigna el patrón y la clase al que pertenece
- Tipo de dato: Define el tipo de dato permitido para el ingreso del atributo desde la pantalla
- Formato: Establece el formato que debe tener la información ingresada por el usuario
- Mensaje de validación: Este mensaje se muestra al usuario si ingresa información que no cumple con el formato requerido

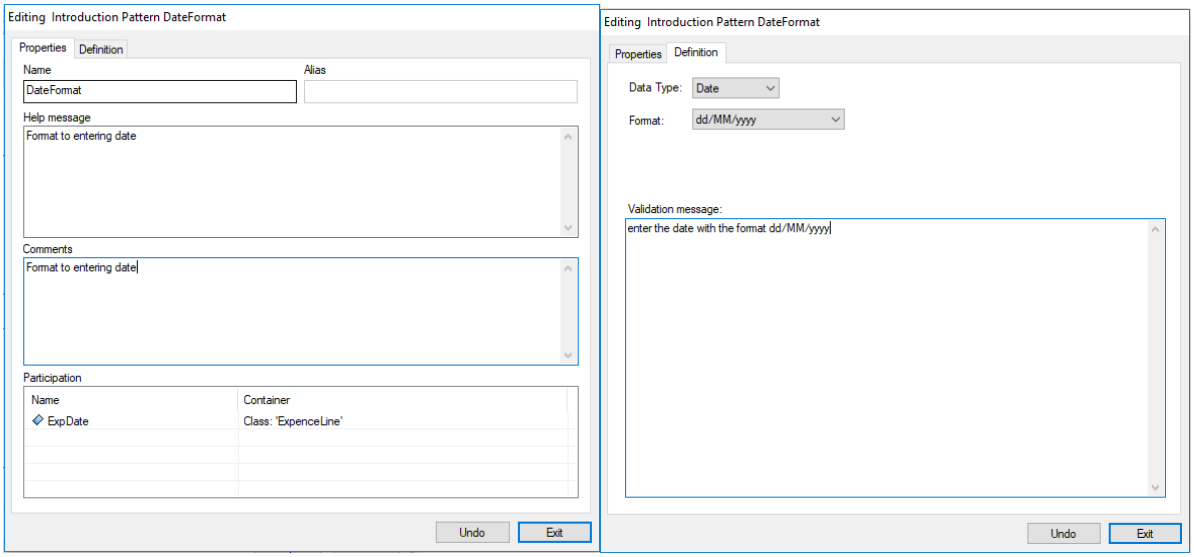

Figura 70. Crear Patrones

La figura 70 muestra la creación de una máscara para el campo fecha, se indica el formato que va a tener este campo “dd/MM/yyyy”

Selección definida

Es utilizado para enumerar valores válidos para un atributo. Por ejemplo en el caso de la nota de gasto para mostrar el estado se necesita que esté en una forma entendible para el usuario y no con el código almacenado internamente en el sistema. Para esto es necesario un patrón de selección definida que se asigna al atributo cada valor correspondiente como en la figura 71.

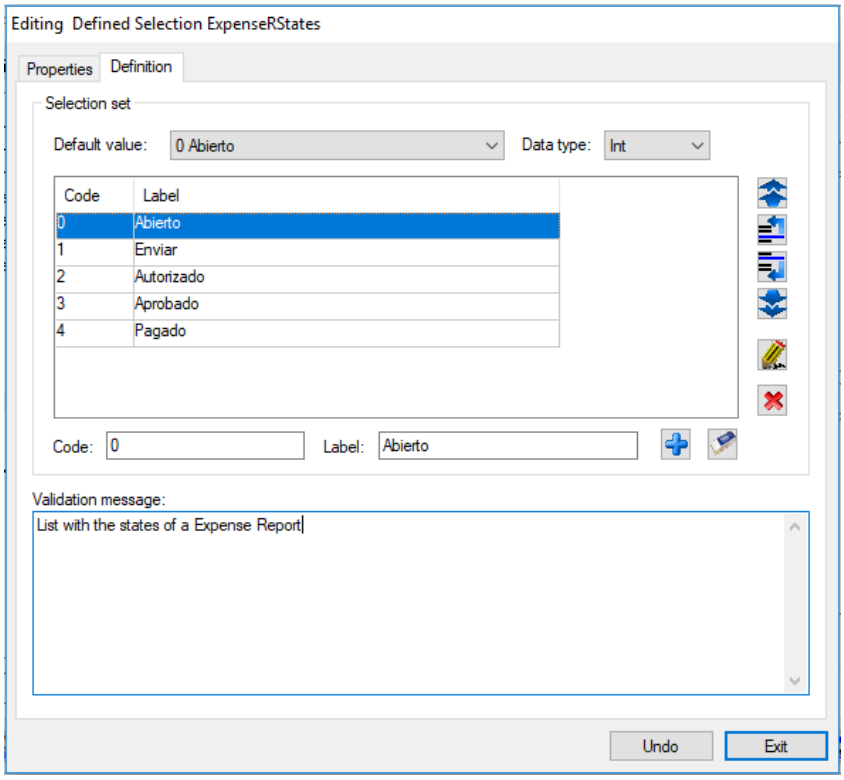

Figura 71. Patrón de Selección Definida

Entre las propiedades principales al crear este patrón están:

- Valor por defecto: Este valor se muestra seleccionado al momento de cargar la lista
- Tipo de dato: Es importante que el tipo de dato que se selecciona en esta propiedad sea igual al tipo de dato que tiene el atributo
- Lista de valores: Está conformado por un conjunto de pares en donde el primer valor es el que se almacena internamente en el sistema y el segundo valor es el que muestra al usuario
- Mensaje de validación: Muestra al usuario información sobre este patrón

Agrupación de Argumentos

Integranova organiza los argumentos creados en los servicios con relación a su orden de creación. Es decir los argumentos que fueron creados primero se enumeran antes de los que se crearon al final. También da la posibilidad de modificar sus posiciones y agruparlos

En la figura 72 como ejemplo se ha agrupado algunos de los argumentos utilizados para crear un nuevo empleado. Se ha creado un grupo para la información relacionada a la cuenta, en el que se agregan los atributos de contraseña y correo. Luego se crea un segundo grupo para la información de contacto en el que incluyen los atributos de teléfono y dirección.

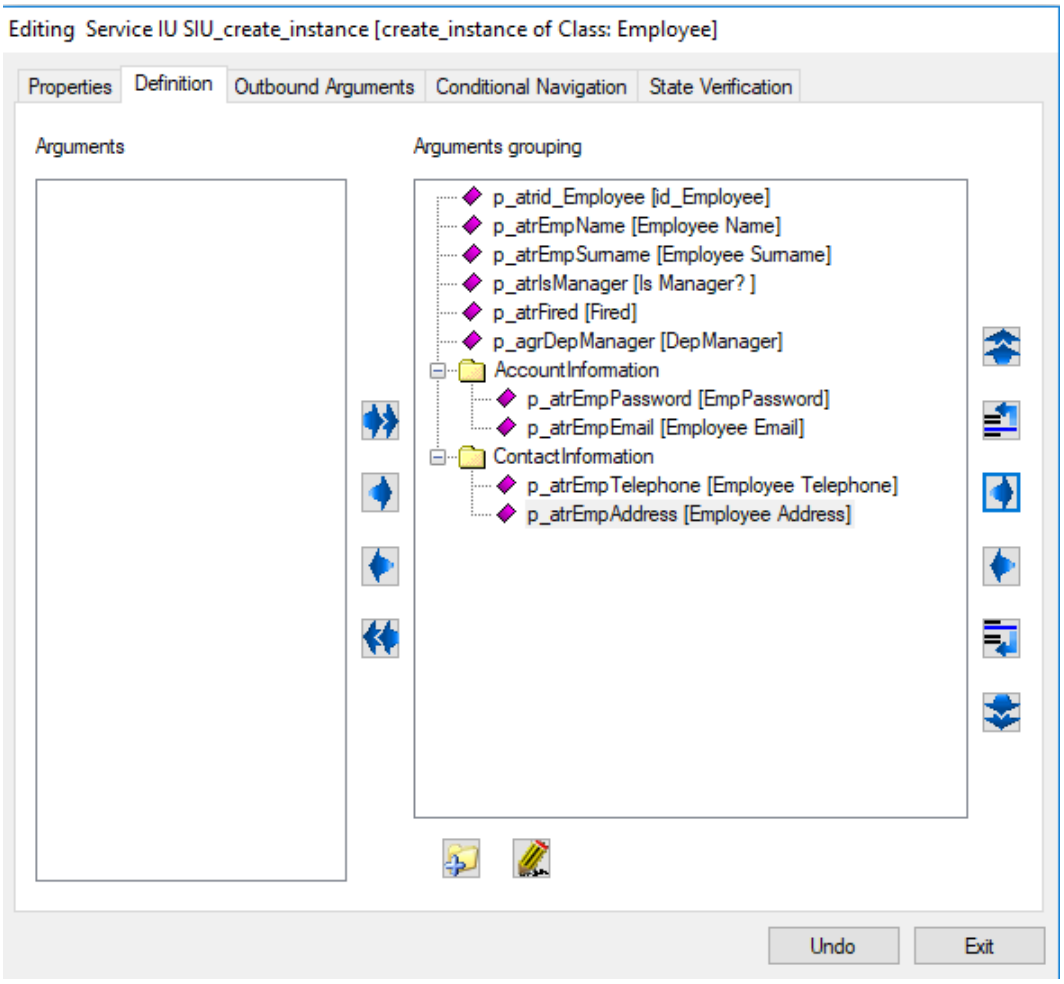

Figura 72. Argumentos agrupados

Filtro

Cuando una clase se relaciona con otra, es necesario mostrar las instancias de la otra clase, sin embargo si la cantidad de instancias crece se puede crear filtros para esa información. Integranova permite crear patrones de filtro para estos casos.

Entre las principales propiedades que tienen los filtros están:

- Variables: Almacenan los criterios que se establecen para extraer la información.
- Fórmula: Son las condiciones que debe cumplir la información para ser mostrada.

En la figura 73 se ha creado un filtro en la clase empleado para obtener solo los empleados que su nombre coincida con el criterio de búsqueda y que no ha sido despedido. Para esto se crea una variable que almacena el nombre del empleado que se va a buscar, en la formula se compara el nombre con todas las instancias de la clase, para extraer solo los que coincidan. Se agrega también una condición para el atributo despedido indicando que su valor sea falso. Se realiza este proceso para mostrar únicamente aquellos empleados que no han sido despedidos.

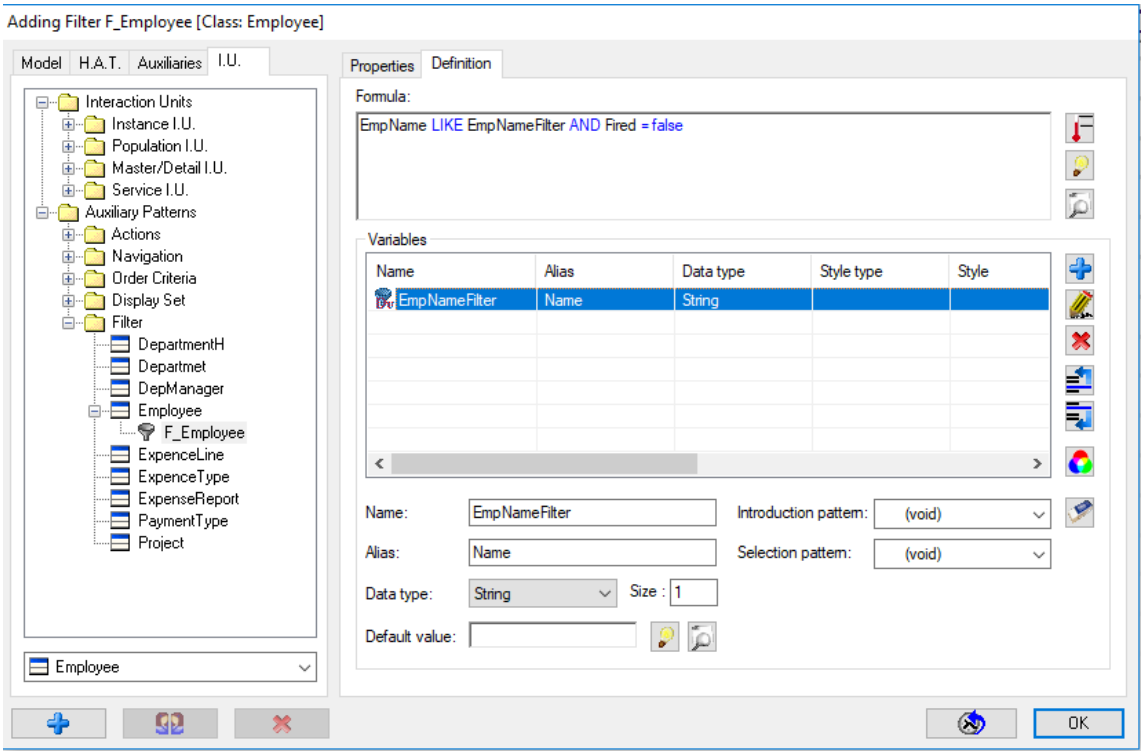

Figura 73. Filtro de Empleado

Criterio de Orden

Integranova permite definir el orden de la información que se recupera, este orden se define por medio de sus atributos. También se puede indicar el sentido de ordenación en términos de ascendente o descendente.

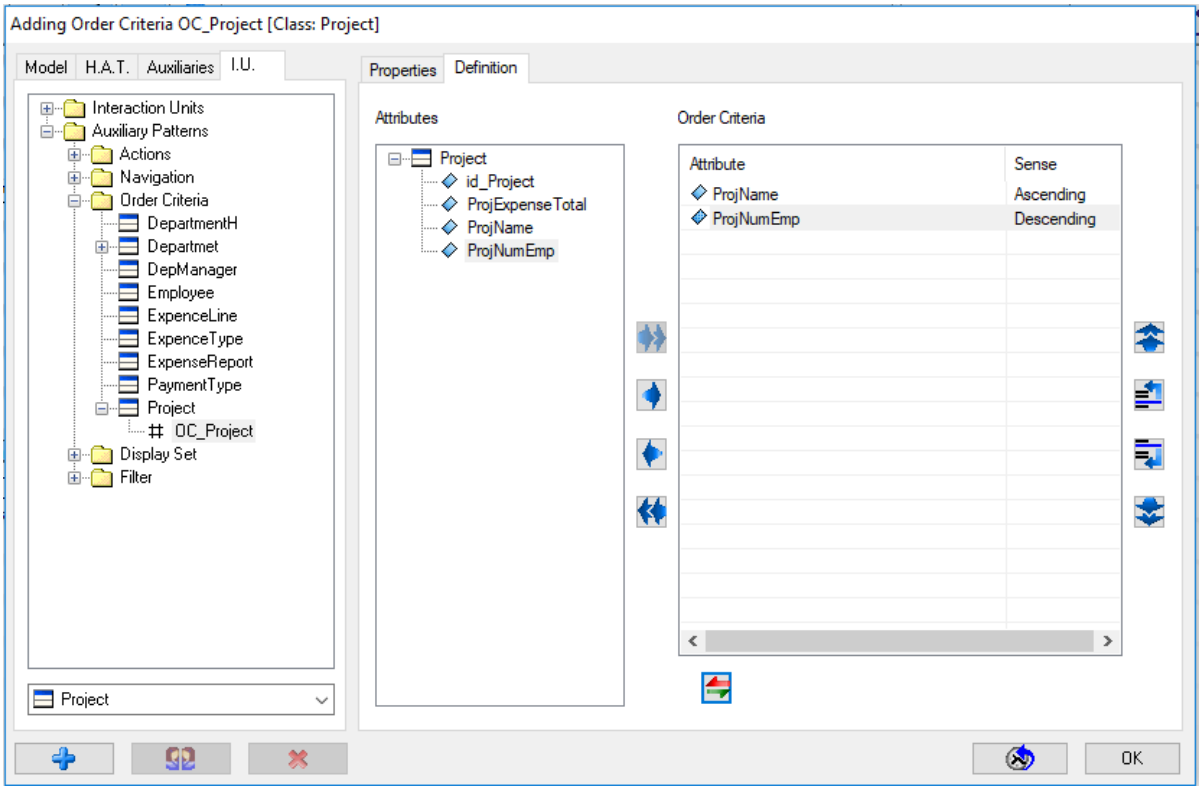

Figura 74. Criterio de Orden

Se puede crear criterios de orden para cada clase y con los atributos que se crea necesarios. En la figura 74 se ha creado un orden para la clase proyecto, sus elementos se ordenan por el nombre de forma ascendente y el segundo criterio es el número de empleados asignados a ese proyecto pero en este caso se ordena de forma descendente, es decir se muestra primero los que tengan un mayor número de empleados asignados.

Conjunto de visualización

Cuando se muestra al usuario un conjunto de instancias de una clase, es necesario modificar los nombres de los atributos, ocultar algunos o agregar nuevos con información de atributos de clases relacionadas. Para esto en Integranova se crea un conjunto de visualización.

En la figura 75 se crea un conjunto de visualización para la clase Empleado, en donde se muestra el nombre, apellido, correo electrónico, dirección, teléfono y el nombre del departamento que es un atributo de la clase departamento con la cual está relacionada. Estos atributos también se pueden ordenar dependiendo de los requisitos.

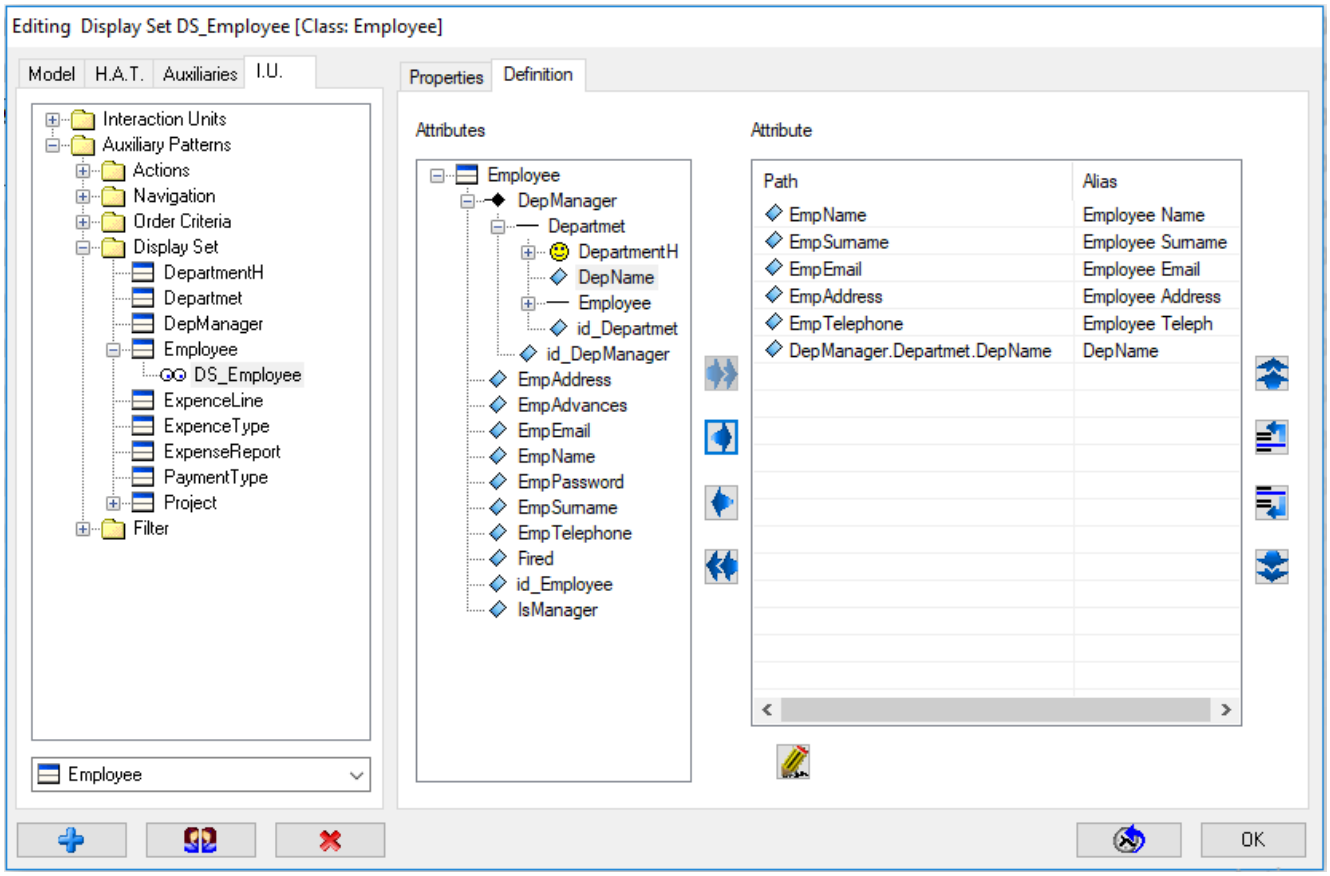

Figura 75. Conjunto de visualización de Empleados

Navegación

Permite el acceso a instancias relacionadas en el escenario actual, al igual que las acciones por defecto todas las clases son accesibles por medio de roles. Se puede crear patrones de navegación para controlar las navegaciones disponibles.

En la figura 76 se ha creado un patrón de navegación para el empleado. En este caso el empleado puede navegar por el departamento pero no puede navegar por la información del gerente del departamento.

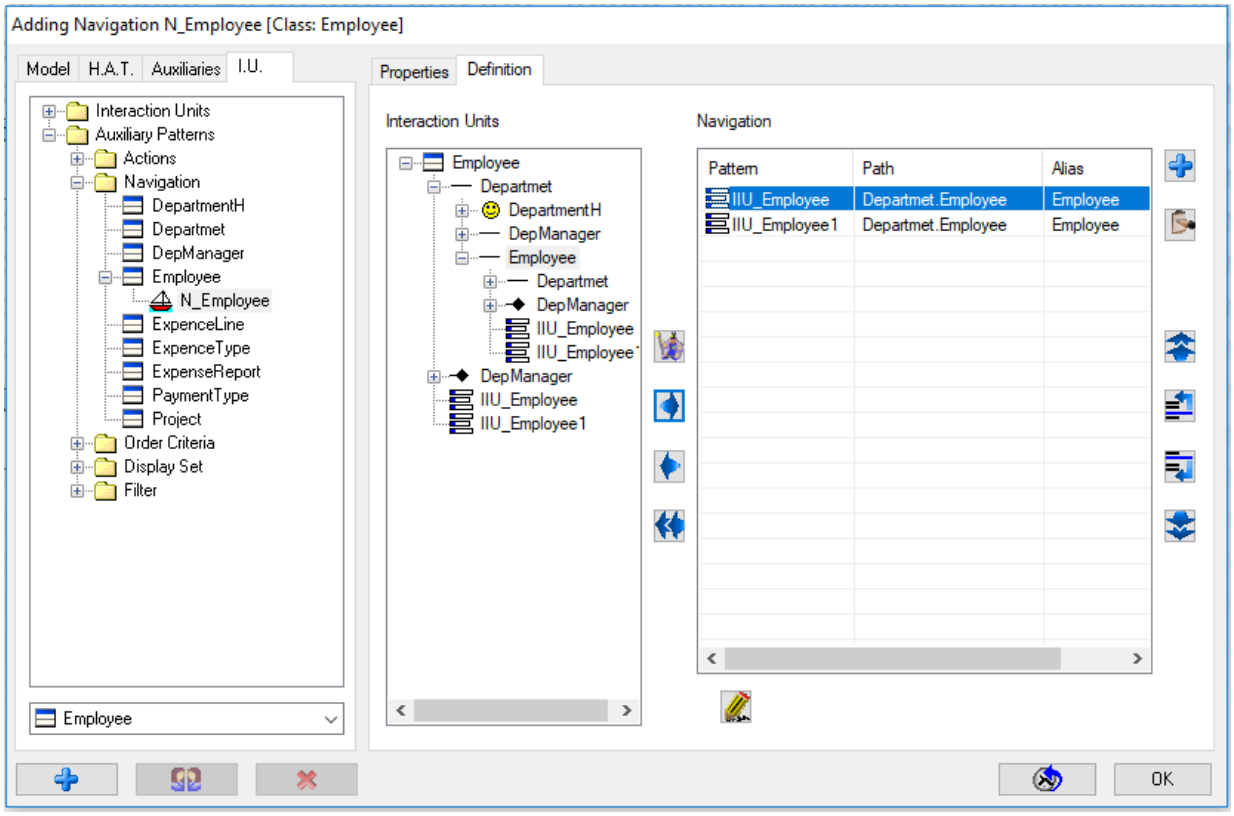

Figura 76. Patrón de Navegación

Acciones

Determina un conjunto de acciones que se ofrecen al usuario. Por defecto todas las acciones que se crean están disponibles, pero Integranova permite crear patrones de acciones para seleccionar solo ciertas acciones como disponibles.

Por ejemplo un empleado puede crear un reporte de gastos y modificarlo, pero no puede eliminarlo entonces únicamente se habilitan las acciones de crear y modificar como se muestra en la figura 77.

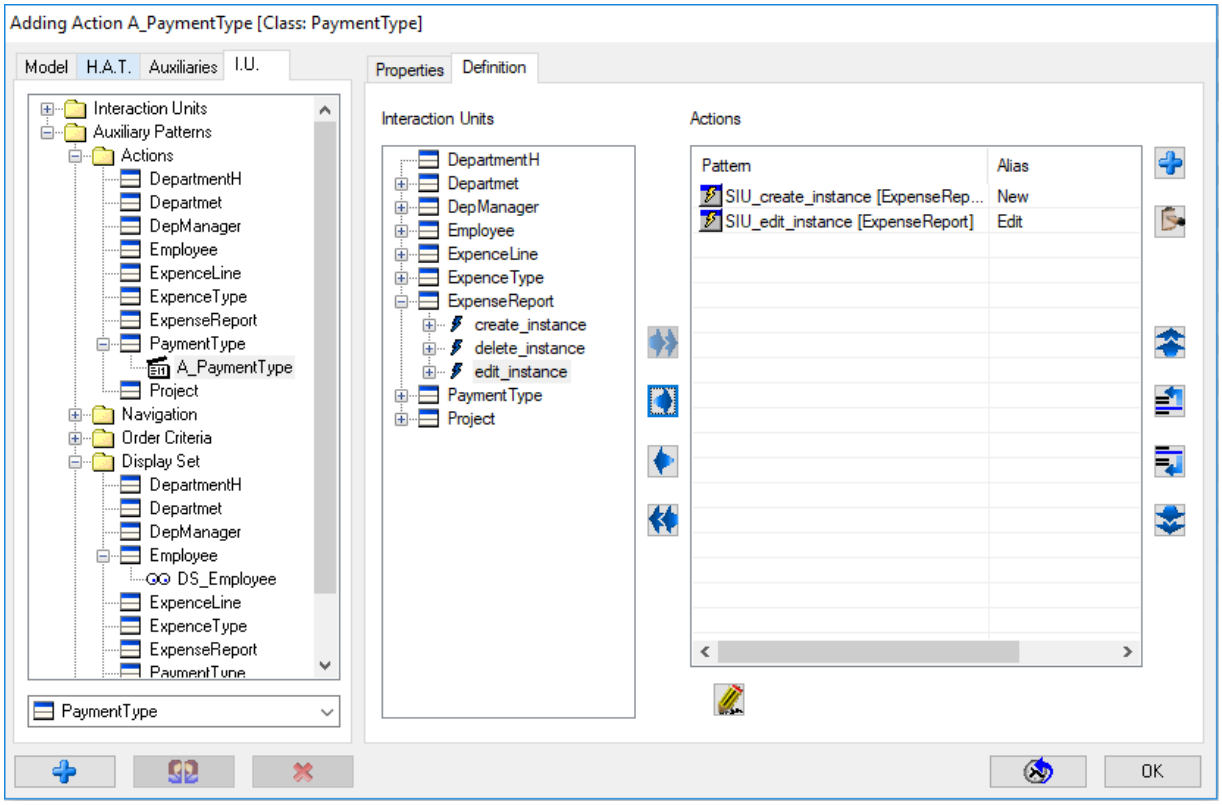

Figura 77. Patrón de Acciones

Unidades de Interacción

Son mecanismos para presentar la información de una manera estructurada, permitiendo al usuario realizar tareas específicas. Integranova incorpora las siguientes unidades de interacción que se pueden ver en la figura 78: Instancia, Población, Maestro/Detalle y Servicio.

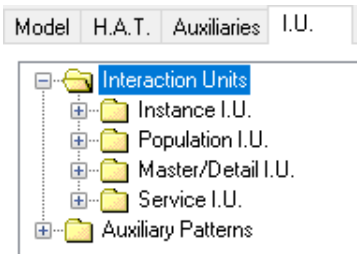

Figura 78. Unidades de Interacción

- Instancia: Este tipo de escenarios muestran al usuario de la aplicación final la instancia de la clase actual en la que se define la Unidad de interacción.
- Población: Integranova ofrece un mecanismo para mostrar una lista de objetos del mismo escenario en el que se usa un objeto. El conjunto de instancias queda restringido a las que el usuario pueda ver.
- Maestro Detalle: Los atributos que se muestran y el tipo de acciones que se ejecutan dependen del tipo de agente
- Servicio: Integranova crea automáticamente una unidad de interacción para cada servicio, pero permite generar nuevas.

Árbol de Jerarquía de acciones

Es un filtro de navegación que permite obtener la población de una clase dependiendo de una condición. Integranova permite aplicar un filtro en el destino cuando se accede directamente a través del menú principal, esto se crea por medio del árbol de Jerarquía de acciones (H.A.T.)

Integranova brinda la posibilidad de crear un H.A.T. por medio de un asistente como se muestra en la figura 79, con eso se obtiene un menú creado automáticamente.

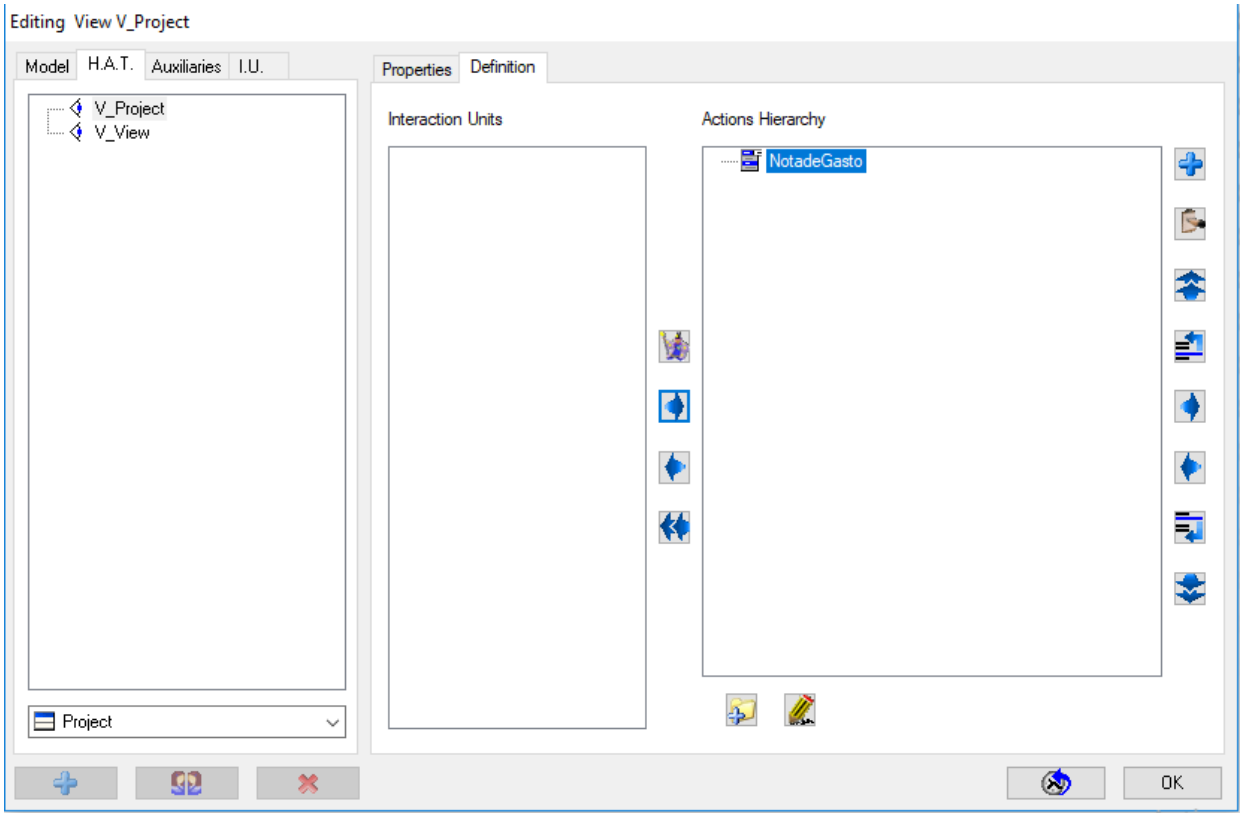

Figura 79. Árbol de Jerarquía de acciones

CAPITULO 6

Comparativa de herramientas WebRatio-Integranova

En el capítulo 3 se definieron las primitivas necesarias que las herramientas de modelado conceptual deben tener y en los dos capítulos anteriores mediante el desarrollo de un caso de estudio en concreto que es la gestión de notas de gasto, se verificó el soporte de dichas primitivas en las herramientas WebRatio e Integranova respectivamente. El siguiente capítulo presenta el resultado de una comparativa, entre estas dos herramientas en donde se analiza cada uno de los modelos planteados en el marco ontológico.

6.1 Modelo de objetos

En este modelo la base fundamental es la posibilidad de definir objetos, que son estructuras de datos con comportamientos y relaciones que determinan la arquitectura del sistema, por lo tanto se ha planteado como primitiva principal la clase y sus relaciones.

Tanto WebRatio como Integranova incorporan un modelo de objetos con clases, atributos y relaciones. Los dos utilizan una arquitectura tres capas, por lo que separan claramente el esquema de la base de datos del resto de la aplicación.

Integranova presenta una capa de persistencia en la que se deposita toda la información de las clases y atributos que definen la estructura del sistema, incorporando también las relaciones que existen entre estos. Por lo tanto es independiente del gestor de base de datos a utilizar, sin embargo solo soporta conexiones con: Oracle, SQL Server, DB2, MySQL.

Mientras que WebRatio utiliza Hibernate para convertir el modelo de objetos de una aplicación en una base de datos relacional, esto lo realiza mediante archivos XML. Al utilizar esta herramienta es posible que el modelo se pueda integrar con cualquier gestor de base de datos, incluso se puede adaptar a bases de datos ya existentes.

Clases

Las clases se forman a partir de una abstracción del mundo real. Estas agrupan objetos con características iguales, lo que determina los atributos que tienen junto con su comportamiento, que son las bases ontológicas planteadas en esta sección.

WebRatio permite representar las clases por medio de Entidades, en las cuales se pueden definir atributos. Como propiedades particulares tiene la capacidad de crear entidades temporales que no se almacenen en la bases de datos y también permite utilizar datos que se originan en otras tablas que no pertenecen al modelo en construcción.

Por su lado Integranova también permite una definición sencilla para crear clases junto con sus atributos, relaciones, eventos, transacciones y validaciones.

La principal diferencia entre estas dos herramientas es la separación clara al momento de crear una clase. WebRatio solo permite generar la clase con sus atributos, y se reserva una sección adicional para el comportamiento, mientras que Integranova permite crear la clase con todos sus componentes en una sola sección. Sin embargo ambas herramientas representan claramente las ontologías planteadas.

Atributos

Las principales propiedades que un atributo debe tener son: nombre, tipo, tipo de dato, tamaño, valor predeterminado, si es requerido o no y un indicador si puede ser nulo.

WebRatio al igual que Integranova establecen por defecto un atributo como llave primaria al momento de crear la clase. Permiten ingresar un nombre y el tipo de dato, para el cual ofrece una amplia gama: condicional, entero, cadenas de texto, fechas, etc.

WebRatio crea por defecto un atributo simple pero si se escoge derivación, entonces tiene opciones de: constante, derivado, importado y calculado, mientras que Integranova especifica tres tipos: constante, variable y derivado. A diferencia de Integranova el tamaño y valores por defecto, el indicador si es nulo o es requerido y otras validaciones en WebRatio se realizan directamente en el modelo de presentación.

Servicios

Es necesario que toda herramienta permita definir la funcionalidad, es decir las tareas que puede realizar un objeto, para esto se distinguen los siguientes componentes: eventos, argumentos, transacciones y operaciones.

Integranova adiciona por defecto el evento de creación, modificación y eliminación al momento de crear una clase, también permite el paso de argumentos y la generación de transacciones y operaciones. Todo esto se puede realizar al momento de crear la clase.

Por su parte WebRatio no incluye por defecto estos eventos pero permite su definición, el paso de argumentos es algo principal al momento de establecer el comportamiento por lo que WebRatio lo integra de buena manera. También incorpora un apartado de módulos que permite dividir los proyectos en partes que se puedan reutilizar varias veces, esto es útil en proyectos extensos. Lo que no cuenta es con la definición de operaciones.

Restricciones de integridad

Es importante que toda herramienta incorpore condicionales que deban cumplirse para que un objeto pueda o no cambiar de estado, estas son llamadas restricciones.

En WebRatio las restricciones se agrupan en: condiciones de selectores y condiciones de visibilidad. Estas permiten recuperar datos de acuerdo a criterios y mostrar u ocultar elementos de las páginas creadas. Por lo tanto lo que permite representar son condiciones estáticas y pre-condiciones, no existe una definición clara para condiciones dinámicas y post-condiciones.

Integranova en cambio reconoce las condiciones como: restricciones en términos del estado de un objeto, que se analizan después de la ejecución de un servicio y restringen información para los usuarios y restricciones en términos de condiciones o también llamadas pre-condiciones, que se analizan antes de la ejecución de un evento.

Relaciones entre clases

Este punto determina el comportamiento del sistema y define como se comunican los objetos entre sí mediante las relaciones que tienen. Se determina como principales primitivas las relaciones de asociación (asociación simple, agregación, composición), herencia (generalización, especialización) y relación de agentes.

Las relaciones que se pueden definir en WebRatio son de asociación simple, y herencia (especialización), quedando reducido el nivel que tiene para representar la comunicación entre clases. Como características principales es posible definir la cardinalidad en las relaciones que puede ser: de uno a uno, de uno a muchos y de muchos a muchos. Además cuenta con una propiedad para definir si la eliminación se realiza en cascada, esto es útil en la herencia.

Integranova en cambio aborda claramente todas las relaciones establecidas en las primitivas, junto con su respectiva cardinalidad, lo que no incluye es la parte eliminación en cascada que tiene WebRatio.

Relación de Agentes

Por medio de esta relación es posible definir agentes en el sistema, que tiene con accesos a diferentes secciones o funcionalidades. Se debe controlar el acceso a las clases, atributos y relaciones.

Los agentes son los encargados de activar servicios en determinadas clases. Con esto se establece la seguridad en el sistema.

WebRatio crea por defecto las entidades de Modulo, Usuario y Grupo para controlar el acceso a partes específicas de la aplicación acorde al rol del usuario que ingresa al sistema. Sin embargo el nivel de control no es tan detallado puesto que no se puede restringir el acceso por atributo o relación. Queda limitado únicamente a clases y eventos. WebRatio incorpora el acceso directamente en las páginas de la aplicación.

Para definir la relación de agentes en Integranova se requiere de un proceso largo y detallado puesto que se da acceso a cada una de las clases, atributos y relaciones de ser necesario. Pero una vez que se configura se cuenta con un sistema cuyos procesos sean validados y utilizados por los agentes que están directamente conectados. Cuando un agente ingresa al sistema solo tiene acceso a los servicios que pueda ejecutar y a los atributos que tenga permiso de visualizar.

6.2 Modelo Dinámico

Para determinar la secuencia de los eventos en un objeto o las interacciones entre objetos se especifica un conjunto de primitivas que se representan mediante dos diagramas: el Diagrama de Transición de Estado para describir las secuencias de eventos en una clase y el Diagrama de Interacción para describir la comunicación e interacción entre los objetos.

El Diagrama de Transición de Estados sirve para describir la secuencia de eventos válidos para un objeto. Definiendo como estado, la situación en la que se encuentra un objeto y transiciones los cambios de estado que se producen, se debe tener en cuenta que una transición es una respuesta a un evento que satisface una condición. Este diagrama indica cómo se comportaría el sistema en tiempo real si existiera la tecnología perfecta.

Integranova permite generar automáticamente este diagrama en el cuál se puede visualizar la secuencia en la que se ejecutan los servicios que se definen en una clase, mientras que en WebRatio este diagrama no se puede generar.

En el Diagrama de Interacción la comunicación entre objetos puede ser representada mediante disparadores (triggers) que son procesos que se ejecutan automáticamente cuando ocurre un evento en particular y operaciones globales que son unidades de ejecución compuestas por varios servicios.

De Igual manera Integranova tiene la opción para visualizar este diagrama automáticamente, y se forma con los triggers y operaciones globales que se crean en el sistema, WebRatio no muestra tampoco este diagrama, sin embargo permite la creación de disparadores que permitan desencadenar eventos específicos y también la creación de módulos que engloban varias clases y eventos y son capaces de recibir argumentos tanto de entrada como de salida.

6.3 Modelo Funcional

Este modelo se utiliza para determinar en qué manera un evento afecta al estado de un objeto para esto se necesitan especificaciones que muestren de forma precisa los cambios de estado en un determinado objeto cuando se ejecuta un evento por lo tanto se plantean las primitivas de evaluaciones usando condiciones y categorías.

Integranova define el comportamiento de las operaciones y se distinguen tres tipos: estado, cardinal y de situación. Ofrece varias opciones para expresar invariantes y utiliza constructos básicos presentes en cualquier lenguaje de programación: secuencia, iteración, decisión y recursión. Además permite que un atributo pueda tener más de una evaluación definida para el mismo evento.

WebRatio en cambio utiliza el lenguaje de modelado web WebML para definir la funcionalidad de la aplicación por medio de sus páginas, operaciones, servicios, etc. Cuenta con componentes que determinan ciertas funcionalidades de la aplicación pero también tiene la posibilidad de personalizar dichos componentes para adaptarlos a funcionalidades especiales.

WebRatio permite determinar las actividades que puede realizar cada usuario para especificar los requisitos funcionales a los que tiene acceso, representando claramente las primitivas de condiciones y categorías planteadas.

6.4 Modelo de Presentación

Es indispensable que la herramienta permita interactuar al usuario con el sistema, capturando datos de entrada y mostrando información procesada, esta interacción debe ser independiente del diseño. Las herramientas deben ser capaces de capturar cada parte del problema y transformarlo en una representación de software adecuada para la solución. Como primitivas conceptuales se plantea: Elementos básicos, unidades de interacción y estructuras de acceso al sistema.

WebRatio incorpora una serie de componentes llamados componentes de vista, los cuales se pueden configurar y adaptarse así a las necesidades del sistema en cuanto a interacción. Mientras que Integranova destina un modelo de presentación en el cuál se establecen todos los aspectos de interacción con el usuario y navegación en el sistema.

Las dos herramientas tienen un extenso conjunto de componentes para representar diversos aspectos del sistema, sin embargo WebRatio ofrece mayor versatilidad a la hora de configurar y realizar componentes con especificaciones y validaciones más detalladas.

Elementos básicos

Describen aspectos concretos de la interacción del usuario con el sistema, se distinguen los siguientes tipos: entrada para capturar datos del usuario, selección definida para mostrar al usuario listas de valores seleccionables, agrupación de argumentos para definir el orden de los argumentos que se muestran al usuario, argumentos de dependencia para definir los estados entre dos argumentos y filtros.

WebRatio permite la creación de formularios para el ingreso de información del usuario, en este formulario se pueden agregar diversos componentes como cuadros de texto, listas desplegables y también crear diversas reglas de validación para la información que se ingresa. Incluso se puede configurar máscaras para dar un formato apropiado a cada campo. Para mostrar la información es posible incluir filtros y criterios de ordenamiento.

De igual manera para la interacción con el usuario Integranova dispone también de diversos componentes en los que se puede configurar máscaras, rangos de valores permitidos y mensajes de error. Cuenta también con componentes para configurar listados de manera personalizada, incorporando filtros y criterios de orden. Además es posible incorporar la navegación entre clases por medio de sus relaciones.

Los componentes que tienen ambas herramientas buscan ofrecer al usuario una mejor experiencia a la hora de interactuar con el sistema.

Unidades de Interacción

Permite especificar escenarios para que un usuario pueda realizar tareas específicas, formando interfaces de usuario. Se plantean las unidades de interacción de servicio en la que el usuario ejecuta un servicio, de objeto para la organización de objetos individuales, de una colección de objetos para manipular un conjunto de objetos y de maestro/detalle para las colecciones de varias clases relacionadas entre sí.

Integranova incorpora las cuatro unidades de interacción planteadas de forma muy clara, mientras que WebRatio utiliza contenedores de vista para organizar la aplicación, mostrar contenido e interactuar con el usuario y así modelar la estructura deseada. Los componentes que permite utilizar son vistas de sitio, áreas y páginas.

Árbol de Jerarquía de acciones

Es necesario definir un menú por medio del cual el usuario puede acceder a las diferentes funcionalidades del sistema, estas funcionalidades deben agruparse en forma de árbol para mostrarlas al usuario de forma ordenada. Debe ser posible construir diferentes arboles de jerarquía de acuerdo al tipo de usuarios que se creen.

En Integranova es posible crear este árbol por medio de un asistente del sistema, lo que facilita su implementación. Además emplea los agentes creados para determinar los accesos a diferentes propiedades del sistema.

En WebRatio se pueden crear rutas de navegación predefinidas para guiar al usuario en el uso de la aplicación web, y también permite realizar personalizaciones a un menú por medio de hojas de estilo. De igual manera se pueden generar rutas de navegación de acuerdo al rol asignado al usuario.

En la tabla 2 se muestra un resumen de la comparativa realizada entre las dos herramientas frente a las primitivas planteadas en el capítulo 3.

Primitiva	**WebRatio**	**Integranova**
Modelo de Objetos	Utiliza el modelo de Entidad-Relación para producir un esquema del sistema que se convierte en la base de datos relacional. Para ello emplea Hibernate, que se integra con todas las bases de datos relaciones disponibles. Por lo tanto está en la capacidad de integrarse con sistemas ya existentes en las empresas	La capa de persistencia de Integranova contiene toda la información de las entidades y los atributos que se definen en el sistema. Permite una conexión con los siguientes gestores de base de datos: Oracle, SQL Server, DB2, MySQL

Clases	Representa las clases por medio de Entidades compuestas por atributos que pueden ser de diversos tipos. También permite generar entidades que no se almacenan en la base de datos, la información de la entidad permanece activa solamente durante la sesión o solo si está habilitada la aplicación, esto se logra por medio de la propiedad de persistencia	Permite la creación de clases formadas por un conjunto de atributos que pueden ser de diversos tipos. Al definir la clase también se pueden establecer sus relaciones, eventos, transacciones y validaciones
Atributos	Al crear la entidad se crea un atributo por efecto como llave primaria, pero se puede modificar. No hace falta definir si el atributo es requerido o puede ser nulo, si son necesarias estas u otras validaciones se las puede realizar directamente en el modelo de presentación. Los atributos tienen la propiedad de derivación que permite establecer valores constantes, importar datos u obtener valores calculados	Cuando se crea la clase también se crea automáticamente un atributo de tipo autonumérico como clave principal, el cuál puede ser modificado. Al crear atributos se puede definir el tipo de dato, tamaño, valor por defecto, si es requerido o si puede ser nulo, además permite agregar atributos constantes, variables o derivados
Servicios	Existe un apartado para definir todos los servicios, esto hace posible incorporar una gran variedad de eventos desde simples hasta más complejos, que involucran varias entidades. En cuanto al paso de argumentos se establecen por medio de "Parameters Binding" que permite el paso de varios parámetros a la vez. También es posible definir transacciones que se crean como módulos independientes y pueden ser utilizados en cualquier parte del sistema	Al crear una clase se generan automáticamente los servicios de creación, eliminación y modificación de una instancia. Integranova permite crear nuevos eventos, transacciones u operaciones, todos ellos con la posibilidad de incorporar argumentos de entrada o de salida estableciendo el tipo de dato y un valor por defecto
Restricciones de Integridad	Permite configurar restricciones mediante condiciones de selectores o directamente en el modelo de presentación. Esto permite que los atributos tomen ciertos valores de acuerdo a las restricciones del sistema. Se puede configurar restricciones a la llave primaria, a los atributos o a las relaciones	Es posible condicionar la ejecución de los servicios mediante la incorporación de pre-condiciones. Esta condición se establece mediante una fórmula que incluye constantes, funciones y atributos

Relaciones entre clases	Las relaciones están compuestas por atributos entre los cuales destacan la cardinalidad que permite: de uno a uno, de uno a muchos y de muchos a muchos. También es posible definir si la eliminación se realiza en cascada con los atributos derivados y se puede establecer una relación de asociación o de herencia	Permite generar relaciones de asociación, agregación y composición y para representar la herencia tiene especialización y generalización, en todas las relaciones se puede establecer la cardinalidad
Relación de agentes	Para otorgar el acceso a partes específicas de la aplicación de acuerdo con el usuario que ingresa y su respectivo rol, WebRatio integra automáticamente en el sistema las entidades de Modulo, Usuario y Grupo	Permite desarrollar un sistema cuyos procesos sean validados y utilizados por los agentes que están directamente conectados. Cuando ingresa un agente al sistema solo tiene acceso a los servicios que pueda ejecutar y a los atributos que tenga permiso de visualizar
Modelo Dinámico	Permite realizar formularios en los cuales mediante una interacción con el usuario se puedan desencadenar eventos específicos, esto se logra mediante el uso de “Ajax”. También permite realizar servicios globales ejecutando varios servicios de diferentes clases, mediante la creación de módulos que reciben argumentos de entrada y son capaces de enviar datos de salida, indicando si la operación se realizó con éxito o no, sin embargo no es posible visualizar un diagrama de transición entre estados ni de interacción de objetos.	Genera automáticamente el diagrama de transición de estados para esto selecciona todos los servicios definidos en la entidad y se visualiza el comportamiento del sistema. Por otro lado también es posible generar el diagrama de interacción de objetos, esto se forma con los triggers y servicios globales que se definen en el modelo
Modelo Funcional	Usa WebML en la definición de cada aspecto funcional de la aplicación (páginas, operaciones, vistas, servicios, etc.). Permite la utilización de componentes personalizados para definir las funcionalidades de una aplicación, un componente personalizado está escrito en Java y realiza una tarea definida que no está incluida en las funcionalidades del componente estándar de WebRatio	Tiene un apartado de modelo funcional para detallar el comportamiento de las operaciones, se distinguen tres tipos: estado, cardinal y de situación. Ofrece varias opciones para expresar invariantes y utiliza constructos básicos presentes en cualquier lenguaje de programación: secuencia, iteración, decisión y recursión
Modelo de Presentación	Para mejorar la interacción del software con el usuario tiene una serie de componentes que se pueden configurar y así facilitar la transferencia de la información, estos se denominan componentes de vista	Para definir la relación del sistema con el usuario plantea un modelo de presentación que cubre todos los aspectos de navegación e iteración

Elementos básicos	Permite la creación de formularios para capturar los datos del usuario. En los campos de los formularios se pueden agregar reglas de validación y configurar máscaras. Para presentar listados utiliza componentes específicos en los que se puede crear condiciones a la lista de objetos, configurar el orden de los objetos en la lista y determinar los atributos que se van a mostrar al usuario. WebRatio también permite definir el flujo de navegación que sigue cuando se produce un evento en un formulario	Presenta un conjunto amplio de elementos y propiedades, que permiten interactuar de mejor manera con el usuario. En los atributos simples permite configurar máscaras, rango de valores y mensajes de error. Si se trata de listados se puede incorporar patrones de filtro y establecer criterios de orden (ascendente, descendente). Se puede también ocultar ciertos atributos de la clase y modificar los nombres de otros, creando un conjunto de visualización apropiado para cada requerimiento. Si la clase tiene relaciones, también permite el acceso a las instancias relacionadas
Unidades de Interacción	Permite identificar perfiles para los accesos dependiendo de las necesidades de cada usuario, especificando interacciones entre los usuarios y los perfiles. Para organizar la aplicación utiliza contenedores de vista que muestran el contenido e interactúan con el usuario, con estos contenedores se puede modelar la estructura deseada de la aplicación	Tiene mecanismos para presentar la información de una manera estructurada, permitiendo al usuario realizar tareas específicas. Incorpora las unidades de interacción: Instancia, Población, Maestro/Detalle y Servicio
Árbol de jerarquía de acciones	La opción más común para crear un menú es usar los puntos de referencia definidos en las unidades de interacción, aunque también se puede modelar un menú estructurado con varios niveles para ello permite integrar hojas de estilo	Integra un árbol de Jerarquía de acciones para acceder directamente a través de un menú principal y navegar por la aplicación, esto se puede crear por medio de un asistente

Tabla 2. Comparativa

CAPITULO 7

Conclusiones y Trabajos futuros

En este capítulo se describe las conclusiones finales de la investigación realizada y se plantean trabajos que pueden ser abordados en futuras investigaciones.

7.1 Conclusiones

Se concluye que utilizar el Método OO como escenario para determinar el conjunto de primitivas conceptuales esenciales ha sido de gran utilidad puesto que asocia cada primitiva con una representación correspondiente en el software, gracias a ello ha sido posible el planteamiento del conjunto de primitivas necesarias. Esto permitirá un desarrollo de software en el que no se dependa del código fuente sino en el que el desarrollador se centre únicamente en realizar un modelo correcto del problema a solucionar. Otro beneficio que se logra es un desarrollo más independiente, en donde otros programadores sean capaces de entender el sistema fácilmente. Ahorrando esfuerzo y costos en el desarrollo.

La capacidad que tiene cada herramienta para soportar base ontológica planteada junto con el nivel que tiene para representar dicha ontología es lo que diferencia a una herramienta de otra y es lo que determina el grado de efectividad y eficiencia en el desarrollo de sistemas a partir de modelos conceptuales.

En este trabajo se ha realizado un marco experimental que incluye la evaluación de las primitivas planteadas, donde se ha evaluado la herramienta Integranova y WebRatio. Para determinar el soporte ontológico se realizó un caso de estudio en concreto que es la gestión de notas de gasto, mediante la cual se pudo comprobar el grado que tienen las dos herramientas para representar las ontologías definidas.

En el análisis realizado en cada una de las herramientas a través del caso de estudio planteado se puede concluir que aunque las dos herramientas se basan en el desarrollo dirigido por modelos, cada una representa de distinta manera en sus modelos conceptuales las primitivas planteadas. En Integranova al basarse en el modelo OO se pudo identificar de una manera más clara la representación de cada primitiva, mientras que en WebRatio se tuvo que analizar más a detalle la manera en la que representaba dicha primitiva, teniendo que revisar de cada una de las características que presenta la herramienta. Integranova separa claramente todos los modelos definidos en el soporte ontológico, mientras que en WebRatio se pueden distinguir claramente el modelo de objetos y el modelo de presentación, los modelos dinámico y funcional se encuentran implícitos entre las diversas propiedades que la herramienta posee.

Por medio del desarrollo del caso de estudio planteado también se pudo analizar que en general las dos herramientas incorporan varias características que toda herramienta MDD tiene, como son: la representación de objetos del mundo real junto con sus atributos y relaciones, definir la secuencia que tiene un servicio en un objeto, diversos procesos y operaciones que se deban generar para cumplir con los requerimientos planteados y la interacción del usuario con el sistema, sin embargo se podría agregar todavía más expresividad en cada herramienta, por ejemplo en WebRatio no se puede definir relaciones de agregación y composición entre clases.

Al finalizar el trabajo mediante la comparativa se concluye que Integranova es una herramienta confiable que cubre en gran porcentaje las ontologías planteadas por lo tanto está en la capacidad de crear aplicaciones robustas. Sin embargo existen problemas en términos a su usabilidad ya su diseño con relación a herramientas más actuales no es muy comprensible. Por ejemplo al agregar atributos a una clase después de añadir el nombre y tipo de atributo, por defecto se pulsa “Aceptar” y se pierde la información, el usuario cree que se ha guardado y en realidad no lo está.

WebRatio por su parte es una herramienta nueva que está en constante crecimiento y evolución, soporta muy bien las ontologías planteadas y se adapta a los constantes cambios del desarrollo dirigido por modelos. La comparación ha demostrado la ventaja del diseño visual presentado por WebRatio en donde los programadores pueden captar rápidamente las ideas del diseño de MVC, sin la necesidad de un conocimiento previo de programación avanzada.

El desarrollo de este trabajo brinda un aporte importante para mejorar el desarrollo de software dirigido por modelos, por lo que puede llevar a desarrollos más entendibles, independientes y sobre todo robustos, es por esto que considero apropiado el tema investigado.

Se concluye también que hoy en día existe una importante actividad en esta área, tanto a nivel académico como industrial. Se desarrollan conferencias internaciones con trabajos relevantes de profesionales e investigadores, como ER (International Conference on Conceptual Modeling) (ER, 2017), CAiSE (International Conference on Advanced Information System Engineering) (CAiSE, 2018), MoDELS (International Conference on Model Driven Engineering Languages and Systems) (MoDELS, 2014), para presentar tanto ponencias científicas de investigaciones y técnicas innovadoras sobre la ingeniería dirigida por modelos en los procesos de desarrollo, como ponencias sobre experiencias reales en proyectos concretos.

También existen cada vez más herramientas que se basan en esta metodología como es el caso de IBM Rational Rhapsody family (IBM Developer, s.f.), Jamda (Jamda, s.f.), AndroMDA (AndroMDA, s.f.). Que proveen de un soporte integral para el diseño y el desarrollo dirigido por modelos.

7.2 Trabajos Futuros

La investigación realizada constituye una base en el desarrollo dirigido por modelos a partir de la cual se pueden plantear diversos estudios ampliando así los resultados obtenidos.

Como trabajos futuros, surgen dos líneas de investigación inmediata que me propongo abordar en el futuro:

1. Comparar la ontología fundacional presentada en esta Tesis con otras existentes (como UFO [] (Guizzardi, Wagner, Almeida, & Guizzardi, 2015), Dolce (Guarino, 2009)) con el objetivo de determinar posibles extensiones conceptuales de interés y su implicación en las herramientas analizadas.
2. Incluir en el estudio otras herramientas MDD con el objetivo de crear una taxonomías que las clasifique de forme rigurosa, en el marco de una ingeniería del software dirigida por ontologías.

Referencias

AndroMDA. (s.f.). *AndroMDA*. Recuperado el 20 de 08 de 2018, de AndroMDA: http://www.andromda.org/

Bordonaro, L. (24 de 10 de 2011). *WebRataio.* Recuperado el 1 de 08 de 2018, de https://my.webratio.com/learn/learningobject/getting-started-with-visibility-conditions#toc1

Bruno, M. (24 de 10 de 2011). *WebRatio.* Recuperado el 30 de 07 de 2018, de https://my.webratio.com/learn/learningobject/getting-started-with-selector-conditions

CAiSE. (11-15 de Junio de 2018). Advanced Information Systems Engineering. *30th International Conference on Advanced Information Systems Engineering*. Tallin, Estonia.

ER. (6-9 de Nov de 2017). Conceptual Modeling. *The 36th International Conference on Conceptual Modeling*. Valencia, Valencia, España.

Frigerio, M. (31 de 10 de 2017). *WebRatio.* Recuperado el 01 de 08 de 2018, de https://my.webratio.com/learn/learningobject/how-to-model-rich-forms-v-72?inu1k.current.att1u=418&link=ln231x

Guarino, N. (2009). The Ontological Level: Revisiting 30 Years of Knowledge Representation. En N. Guarino, *Conceptual Modeling: Foundations and Aplications* (págs. 52-67). Berlin: Springer.

Guizzardi, G., Wagner, G., Almeida, J. P., & Guizzardi, R. S. (2015). Towards ontological foundations for conceptual modeling: The unified foundational ontology (UFO) story. *IOS Press Content Libray, 103-4*, 259-271.

IBM Developer. (s.f.). *IBM*. Recuperado el 20 de 08 de 2018, de IBM: https://www.ibm.com/developerworks/downloads/r/rhapsodydeveloper/index.html

Integranova. (2016). *Integranova*. (Integranova) Recuperado el 03 de 08 de 2018, de http://www.integranova.com/es/integranova-m-e-s/

Integranova S.A. (2011). *Integranova.* Recuperado el 04 de 08 de 2018, de http://www.care-t.com/evaluation/downloads/Tutorial_02/Tutorial_02_Creating_static_relationships.pdf

Jamda. (s.f.). *The Jamda Project*. Recuperado el 20 de 08 de 2018, de The Jamda Project: http://jamda.sourceforge.net/

Lindgreen, P. (Mayo 1990). A Framework of Information Sustem Concepts Interim Report. *IFIP WG 8.1 FRISCO*.

MoDELS. (28-03 de Septiembre-Octubre de 2014). Model Driven Engineering Languages and Systems. *17th Intl. Conf. on Model Driven Engineering Languages and Systems*. Valencia, Valencia, España.

Naamad, H. y. (1996). *The STATEMATE semantics of statecharts.ACM Trans SoftwareEng Methodo.* TOSEM.

Pastor López, O. (04 de abril de 1992). El Modelo OO. *Diseño y Desarrollo de un Entorno de Producción Automática de Software basado en el Modelo Orientado a Objetos*, 39-42. Valencia, Valencia, España.

Pastor, O. (4 de Abril de 1992). Metodología de produccion automática de software. *Diseño y Desarrollo de un Entorno de Producción Automática de Software basado en el Modelo Orientado a Objetos*. Valencia, Valencia, España.

Pastor, O. (04 de Abril de 1992). Oasis: Un lenguaje de especificación OO. *Entorno de Producción automática de Software basado en el Modelo Orientado a Objetos*. Valencia, Valencia, España: Servicio de Publicaciones UNIVERSITAT POLITÈCNICA DE VALÈNCIA.

Pastor, O., & Molina, J. (2007). Object-Oriented Modelling as the Starting Point. En O. Pastor, & J. C. Molina, *Model-Driven Architecture inPractice* (págs. 19-21). Berlin: Springer-Verlag Berlin Heidelberg.

Pastor, O., & Molina, J. C. (2007). Conceptual Modelling Primitives. En O. Pastor, & J. C. Molina, *Model-Driven Architecture in Practice* (págs. 49-52). Berlin: Springer.

Pastor, O., & Molina, J. C. (2007). Object Model. En O. Pastor, & J. C. Molina, *Model-DrivenArchitecturein Practice* (págs. 55-11). Berlin: Springer.

Pastor, O., & Molina, J. C. (2007). The OO-Method. En O. Pastor, & J. C. Molina, *Model-Driven, Architecture in Practice* (págs. 39-45). Berlin: Springer.

UML. (2004). *The Object Management Group, Unified Modeling Language: Superstructure, version 2.0.* OMG doc formal/05-07-04.

Wand, Y., & Weber, R. (1990). An Ontological Model of an Information System. *IEEE Transactions on Software Engineering*, Vo. 16, N.11.

Weber, R. (Spring, 1987). Toward a Theory of Artifacts: A Paradigmatic Basis for Information Systems Research. *Journal of Information Systems, Volume 2*, 3-19.

WebRatio. (31 de 10 de 2014). *webratio*. (webratio) Recuperado el 20 de 07 de 2018, de www.webratio.com

WebRatio. (21 de 03 de 2014). *WebRatio.* Recuperado el 30 de 07 de 2018, de https://my.webratio.com/learn/learningobject/reusable-models-modules-module-definitions-v-72#toc9

WebRatio. (2014). WebRatio Platform Modeler. *Sudent-Guide*, pág. 19.

Anexos

Anexo 1 – Ejercicio planteado (Gestión de notas de gasto)

Nota de Gasto. Parte 1

Introducción

El sistema de gestión de informes de gastos es una herramienta que ayuda al proceso de informes de gastos a seguir. Utilizando esta herramienta, es fácil seguir el ciclo de vida de los informes de gastos, desde su creación hasta que se paga al empleado.

Los empleados dan en sus informes de gastos cuando todos los billetes relacionados con viajes de negocios o un almuerzo de negocios hayan sido ingresados.

Si el empleado recibió un pago por adelantado, éste debe ser ingresado cuando se crea el informe de gastos. Además, cada gasto se especificará en una línea. El pago anticipado debe ser restado del gasto total que se pagará al empleado.

Cuando se emite un Reporte de Gastos, éste debe ser revisado por el Gerente de Departamento. El mismo decidirá si el informe de gastos está autorizado y continúa su flujo. Pero puede denegar el informe y especificar el motivo por el cual no autorizó el informe de gastos.

Antes de pagar un informe de gastos, el pago debe ser aprobado por el Gerente de Cuentas. También puede negar el pago, pero en este caso, debe indicar la razón por la cual se le niega.

Si un Gerente de Departamento o Gerente de Cuentas niega el informe de gastos, entonces el empleado debe modificarlo, dependiendo de la razón dada por los gerentes.

Después de que el empleado modifique el informe de gastos, debe volver a emitirlo para iniciar el proceso.

Finalmente, cuando se aprueba un informe de gastos, un usuario de Cuentas o el Administrador de Cuentas puede pagarlo. A continuación, el estado del informe de gastos se "pagará" y se mantendrá la fecha de pago.

Definición de datos del problema

Reporte de gastos

- Encabezado y pie de página: información general sobre los gastos. Recoge todos los gastos relacionados para un proyecto y un empleado.
- Identificador del informe. Debe ser único.
- Fecha de creación del informe de gastos.
- Empleado que entrega el informe
- Proyecto relacionado con estos gastos
- Breve descripción del motivo de los gastos.

- Pago por adelantado
- Total de gastos: suma de todos los gastos
- Saldo (Gastos totales - Anticipo). Si el saldo es superior a 0 y el tipo de pago contiene un coste adicional, este valor se restará. En caso de que después de restar el coste adicional, el resultado de la operación sea negativo, entonces el balance será cero.
- Estado actual del informe de gastos. Puede ser:
    - 0-Abierto
    - 1-Enviar
    - 2-Autorizado
    - 3-Aprobado
    - 4-Pagado
- Fecha de aprobación.
- Fecha de pago y tipo de pago.
- Razón por la cual el informe de gastos no está autorizado (opcional).
- Razón por la cual el informe de gastos no es aprobado (opcional).
- Comentarios relacionados con el pago (opcional).

Línea de Gastos

Cada gasto se especifica en una línea. Datos a almacenar:

- Número de línea: cada informe de gastos tendrá su propio número de lista
- Fecha de gastos
- Tipo de gastos: (Kilómetros, alquiler de coches, billete de avión, factura de almuerzo, etc.)
- Unidades: Número de elementos definidos como gasto. Por ejemplo, kilómetros, número de factura del almuerzo, etc. Su valor predeterminado es 1
- Precio: Representa el precio por unidad. Se asigna en función del tipo de gasto.
- Descripción de gastos: una breve descripción sobre el gasto. Por ejemplo: "Billete de avión de Barcelona a Munich" o "Ticket de almuerzo en el Restaurante Gourmet con clientes (CHG Company)
- Gasto total: precio * unidades

Tipo de Gasto

Un tipo de gasto contiene todo tipo de gastos posibles que se pueden agregar a un informe de gastos. Por ejemplo, alquilar un coche, billete de gasolina, etc.

- Código de tipo de gasto. Longitud máxima 4 caracteres.
- Breve descripción sobre el tipo de gasto
- ¿Es un precio fijo de la empresa?
- Precio: fijo / recomendado.

Tipo de Pago

Contiene todas las formas posibles con las que la empresa pueda devolver el dinero gastado por la misma. El tipo de pago se puede cambiar durante el proceso de informe de gastos.

No es obligatorio vincular un tipo de pago cuando se crea un informe de gastos. Se puede vincular durante el proceso de informe de gastos. Pero antes de pagar un informe de gastos, debe estar relacionado con un tipo de pago.

La información relacionada con un tipo de pago es:

- Código de tipo de gasto: Debe ser único. Longitud máxima 4 caracteres.
- Breve descripción sobre el tipo de gasto
- ¿Cargo adicional?: Algunos tipos de pago tienen cargos adicionales que disminuyen la cantidad que se debe pagar al empleado. De forma predeterminada, la cantidad es cero.
- Porcentaje de cargo adicional: Este campo indica si la cantidad como cargo adicional es una cantidad simple o un porcentaje del monto total del informe de gastos

Departamento

Un empleado puede ser un miembro de un solo departamento. Cada departamento tiene un gerente que es un empleado de este departamento.

- Código de departamento: Debe ser único. Longitud máxima 4 caracteres.
- Nombre de Departamento
- Gerente de departamento
- Número de empleados activos: Que pertenecen a este departamento

Proyecto

Un proyecto puede ser desarrollado por un grupo de empleados. De la misma manera, un empleado puede trabajar en varios proyectos. Un proyecto es administrado por el Gerente del Departamento y puede agregar o remover empleados del grupo de trabajo.

- Código del proyecto: Debe ser único. Longitud máxima 4 caracteres.
- Nombre del proyecto
- Número de empleados: Que se encuentran trabajando en el proyecto
- Total de informes de gastos: Que estén relacionados con el proyecto

Empleado

Este es el conjunto de todos los empleados pertenecientes a la empresa.

- Código del empleado: Debe ser único. Longitud máxima 4 caracteres.
- Nombre y apellido
- Dirección, teléfono y correo electrónico
- Departamento
- ¿Es gerente de departamento?
- Despedido (ahora, él no está trabajando para esta empresa)

Gerente de Departamento

Un empleado puede ser un Gerente de Departamento y tendrá una funcionalidad adicional. Para cada departamento, no es posible tener más de un gerente. Cuando un empleado que es Gerente de Departamento es trasladado a otro departamento, él será un empleado normal en el nuevo Departamento.

- Código del Empleado: Debe ser único. Longitud máxima 4 caracteres. Se recomienda usar las iniciales del empleado.
- Departamento que es el Gerente.

Usuarios

Los usuarios no pertenecen al grupo de empleados.

- Código de usuario: Debe ser único. Longitud máxima 4 caracteres.
- Nombre y apellido
- Email (opcional)

Dependiendo del nivel de acceso, un usuario puede ser especificado como:

- Usuario de Cuentas
- Gerente de Cuentas
- Administrador

Nota de Gasto. Parte 2

Introducción

La funcionalidad de gestión de informes de gastos se incluirá en esta práctica. Esta funcionalidad se caracteriza por:

- Proceso de informe de gastos (proceso básico y principal de la aplicación)
- Ciclo de vida de un empleado en la empresa
- Gestión de proyectos
- Gestión del departamento
- Modo de pago y tipo de gestión de gastos
- Gestión de usuarios

Proceso del Reporte de Gastos

El sistema se utiliza una vez que el empleado reúne toda la documentación (billetes, boletos, etc.) relacionada con sus gastos que se han realizado durante una actividad laboral (viajes, comida, compras...) y quiere que se les reembolse.

El empleado puede haber pedido algo de dinero por adelantado antes de hacer cualquier pago. Esta cantidad de dinero dada por adelantado se reflejará en el informe de gastos en el momento de la creación.

Una vez que se ha creado el informe de gastos, se agregará una línea de gastos por cada gasto que el empleado haya realizado. También será necesario un comentario explicando cada gasto.

Cuando se agrega una línea de gastos, su precio dependerá del tipo de gasto relacionado. Algún tipo de gastos tendrá un precio fijo y su importe se calculará sobre la cantidad de elementos (por ejemplo: € * km, dieta * días, etc.). El resto de los gastos no tendrá un precio fijo por lo que su cantidad dependerá de otros factores. En estos casos, un precio recomendado puede ser dado de manera

predeterminada, aunque puede ser modificado por el empleado, lo que le permite introducir el precio exacto del gasto realizado (por ejemplo: un billete de tren).

Una vez introducida la información, el sistema asignará al informe de gastos el estado de "Abrir". Al estar en este estado, el informe de gastos puede ser modificado, puede tener nuevas líneas añadidas o eliminadas, incluso se puede eliminar por completo. Sin embargo, cuando el empleado emite el informe de gastos, el estado de los gastos se convertirá en "Emitido" y no podrá ser modificado en absoluto. En ese momento el informe de gastos está pendiente de ser autorizado.

Un Gerente del Departamento tiene que verificar que los Informes de Gastos son consistentes con el trabajo que un Empleado ha sido asignado. Por lo tanto, cuando se expide un Informe de Gastos, el Gerente de Departamento tendrá que autorizarlo para que el proceso pueda ser seguido. Después de esta autorización, los boletos y las facturas proporcionados por el empleado deberán ser verificados por el Gerente de Cuenta para que el Informe de Gastos pueda ser aprobado y listo para recibir el pago.

Si el informe de gastos está autorizado, su estado se convierte en "Autorizado", guardando la fecha del sistema en este momento. Una vez alcanzado este estado el informe de gastos estaría pendiente de ser aprobado. Si el informe de gastos no está autorizado, debe darse una razón de denegación y el estado de los gastos se convertirá en "No Autorizado". Con este estado el informe de gastos se puede modificar como lo hace cuando su estado es "Abierto". Cuando el empleado hace los cambios apropiados para hacer que el informe de los gastos sea autorizado, él / ella tendrá que emitir el reporte de gastos de nuevo.

Una vez aprobado el pago, el informe de gastos se establecerá como "Aprobado". En caso de que algo esté mal, se convertirá en "Desaprobado". En este estado el informe de gastos se puede modificar como lo hace cuando su estado es "Abierto". Cuando se hacen los cambios apropiados, el informe de gastos debe ser emitido de nuevo para seguir el proceso.

Cuando el informe de gastos es aprobado, puede ser pagado. Una vez realizado el pago, el estado de los gastos se convierte en "Pago", ahorrando la fecha de pago y forma de pago (efectivo, transacción bancaria).

Cuando se paga un informe de gastos y el dinero avanzado excede los gastos totales, se agregará una sentencia notificando al empleado que tiene 15 días para devolver la diferencia entre el dinero avanzado y los gastos totales en los comentarios del Informe de gastos.

Ciclo de vida del empleado en la empresa

Posibilidad de crear nuevos empleados ingresando toda la información requerida.

Todos los datos de los empleados pueden ser modificados por el mismo servicio.

Si un empleado deja la empresa, se marcará como "Despedido". Él / ella será removido del proyecto / s que él / ella perteneció a, sin embargo el empleado no será borrado. Si el empleado era un departamento responsable, él / ella no será más el departamento responsable.

Un empleado puede ser cambiado de un departamento. Sin embargo, si el empleado era responsable de un departamento, no podrá ser cambiado a menos que él / ella fue despedido previamente.

Debe estar disponible un servicio para eliminar empleados. Si se elimina un empleado, solo si todos sus informes de gastos tienen un año de antigüedad, podrían eliminarse con su Empleado, de lo contrario, el Empleado no podría ser eliminado.

Un empleado puede ser promovido como responsable de un departamento. Si el mismo departamento tuviera ya un responsable, él / ella tendría que ser automáticamente despedido y entonces el otro empleado sería promovido.

Director de Proyectos

Posibilidad de crear nuevos proyectos.

Posibilidad de asignar empleados a proyectos existentes.

Director de Departamento

Pueden ser creados, modificados o eliminados por el administrador del sistema

Un Administrador puede cambiar el Gerente de Departamento.

Modo de pago y tipo de gestión de gastos

Ambos modos de pago y tipo de gastos pueden ser creados, modificados o borrados en el sistema.

Manejo de Usuarios

Posibilidad de agregar nuevos usuarios de cuentas, responsables de cuentas y administradores de sistemas. Posibilidad de modificar o eliminar datos del sistema.

Definición de Perfiles

Auditor

Se debe permitir que un Auditor de sesión inicie sesión en la aplicación sin tener definido un usuario ni una contraseña.

Un auditor debe tener visibilidad sobre toda la aplicación para acceder a toda la información necesaria sin modificar ningún dato.

Líneas de Gastos y Manejo de Reportes

Cuando un empleado hace un informe de gastos, este informe se asignará automáticamente al empleado y no se le pedirá al empleado como argumento entrante.

El Administrador para que el Departamento de Gerentes pueda crear Informes de Gastos, pero, en este caso, tendrá que agregar el Empleado a quien se le ha asignado, como argumento de entrada.

Un empleado puede crear, modificar y eliminar sus informes siempre y cuando estén en estado "abierto".

El Gerente del Departamento será la persona encargada de autorizar o rechazar los informes de gastos.

El Gerente de Cuentas será quien aprueba o rechaza el pago del reporte de gastos.

Los Informes de gastos pueden ser pagados por los Usuarios y los Administradores de Cuentas (aunque hay casos especiales para los usuarios: consulte la sección de restricciones).

Manejo de Empleados

La gestión de los empleados sólo puede ser llevada a cabo por el Gerente del Departamento o el Administrador del Sistema.

Si un Gerente de Departamento crea un empleado, será asignado automáticamente al departamento donde pertenece el Gerente de Departamento. Pero en el caso de que el Administrador cree al empleado, el departamento podrá ser elegido.

Sólo el Administrador podrá:

- Cambiar a un empleado del departamento.
- Promover o despedir a un empleado de / hacia el Gerente del Departamento.
- Elimine un empleado.

Manejo de Proyectos

Sólo pueden ser administrados por un Administrador o un Gerente de Departamento.

Los empleados sólo pueden ver los proyectos en los que participa.

Manejo de Departamentos

La administración del departamento es llevada a cabo por los Administradores.

La información de un departamento puede ser visualizada por sus empleados, su Gerente de Departamento, todos los usuarios y todos los Gerentes de Cuenta.

Tipos de Gasto y formas de pago

El Administrador los crea, modifica o elimina, pero son visibles para todos los usuarios del sistema.

Manejo de Usuarios

La gestión de usuarios de Cuentas es llevada a cabo por los Gestores de Cuentas o el Administrador.

La información sobre los usuarios de Cuentas sólo puede mostrarse a los Administradores de Cuentas, a los Administradores y al Usuario de Cuentas.

La información sobre los administradores de cuentas sólo se puede mostrar al administrador de cuentas mismo o los administradores. Los Administradores de Cuentas y la Administración de Administradores sólo pueden ser realizados por un Administrador. Pero un administrador de cuentas podrá modificar sus propios datos.

Restricciones de la Aplicación

Un empleado no puede tener más de 2000 € en pagos anticipados en todos sus Informes de Gastos que no se pagan, los Informes de Gastos que exceden esta cantidad de dinero en pagos anticipados no pueden ser creados. (Tal vez un nuevo atributo es necesario para cumplir este requisito).

Una Línea de Gastos nunca tendrá un valor negativo. Si un informe de gastos no está en estado "abierto", no puede modificarse o eliminarse (pero si el agente es el administrador).

Un informe de gastos puede ser "Emitido" si está en estado "Abierto".

Un informe de gastos sólo puede ser autorizado o no si se encuentra en estado "Emitido".

Un informe de gastos sólo puede aprobarse o no si se encuentra en estado "Autorizado".

Un usuario de cuentas puede pagar el informe, siempre y cuando el informe sea aprobado y el saldo no exceda de 3000 €.

Si un Empleado es Gerente de Departamento, el Empleado no puede cambiar de Departamento a menos que deje de ser Gerente de Departamento.

Al promocionar un Empleado al Gerente de Departamento, si hay un Gerente de Departamento anterior para este Departamento, es necesario despedir a este Gerente de Departamento antes de promover el nuevo.

Si un empleado ha sido despedido, no será posible asignar nuevos informes de gastos a este empleado.

Si un empleado ha sido despedido, no puede ser promovido a Gerente ni cambiado de departamento.

Ciclo de Vida del Reporte de Gastos

A continuación, se explica el proceso que sigue el informe de gastos:

Cuando se introducen datos, el sistema asigna al informe de gastos el estado "Abierto". Mientras que el estado es "Abierto", el informe de gastos puede ser modificado, y las nuevas líneas se pueden agregar o eliminar el informe de gastos con todas sus líneas. Sin embargo, cuando el empleado emite el informe de gastos, no puede ser modificado; Cambia al estado "Emitido". Ahora el informe de gastos está esperando la autorización.

El Gerente del Departamento se encarga de autorizar o no los informes de gastos. Si se autoriza un informe de gastos, se convierte en el estado "Autorizado" y estará a la espera de ser aprobado. Si no es así, el informe de gastos se convierte en "Abrir". En este estado, el informe de gastos puede modificarse de nuevo. Cuando un empleado ha realizado los cambios necesarios para obtener el informe de gastos autorizado, vuelve a "Emitir" el informe de gastos, por lo que comienza de nuevo su proceso desde el estado "Emitido".

El Gerente de Cuentas será el que autorice o no los pagos de los informes de gastos. Cuando se aprueba el pago, el informe de gastos se verifica como "Aprobado". Si hay algo que no está siguiendo la normativa, el gerente de Cuentas no aprobará el informe de gastos y el estado del informe de gastos será "Abierto". En este nuevo estado, el informe de gastos puede modificarse de nuevo. Cuando los cambios necesarios para ser aprobados son realizados por el empleado o por el Gerente del Departamento, emitir el informe de gastos de nuevo por lo que comienza su proceso desde el estado "Emitido".

Cuando el informe de gastos ha sido aprobado por el Gerente de Cuentas, cualquier miembro del departamento de Cuentas puede pagar el informe de gastos, pero sólo si el saldo de gastos es inferior a 3000 €. En este caso, sólo el administrador de cuentas puede pagarlo. Cuando se ha pagado el pago, el informe de gastos se marca como "Pago".

Cuando se ha pagado un informe de gastos, sólo el administrador puede eliminar y si y sólo si ha pasado un año después de la fecha de pago.