Centro de InvestigaciÛn y de Estudios Avanzados
del Instituto PolitÈcnico Nacional

Departamento de Ingeniería Eléctrica

Sección de Computación

EveMac: Herramienta para el diseño de bases de datos

Tesis que presenta

Teresa Villegas Casas

para obtener el Grado de

Maestro en Ciencias

en la Especialidad de

Ingeniería Eléctrica

Director de la Tesis:

Dr. Sergio Victor Chapa Vergara

México, D.F. **Agosto del 2005**

Agradecimientos

En especial a **Dios**,

Por haberme permitido culminar esta etapa de mi vida.

A mi familia,

Por estar conmigo en mis proyectos.

A mis amigos.

Quienes me apoyaron con sus palabras de aliento en esos momentos cuando todo lo sentia perdido.

Y muy en especial a Anahí Ramiréz Hernandéz, Lorena Chavarría Baéz y Eduardo Martínez Paniagua por todo su apoyo y consejos que me otorgaron incondicionalmente.

Al Consejo Nacional de Ciencia y Tecnología CONACYT y al Centro de investigación y de Estudios Avanzados del IPN, CINVESTAV. Por el apoyo recibido para la realización de la maestria.

Resumen

El diseño de bases de datos es una etapa fundamental del desarrollo de los sistema de información para garantizar el mejo funcionamiento del mismo. Existe la concepción tradicional de almacenar información de un modelo de negocio limitando el uso de las bases de datos a un sólo nivel estructural. Sin embargo, existen actividades inherentes al ciclo de vida de los sistemas relativas al control y administración de proyectos las cuales se apoyan de herramientas de software adicional. De esta manera se plantea la necesidad de contar con una meta base de datos que representa un nivel superior, el cual describa la información del control y administración del proyecto de desarrollo de la base de datos la cuál sustentará al sistema de información.

Esta tesis presenta una metodología y herramienta que apoya el diseño de bases de datos basada en el modelo entidad vínculo extendido. Ofrece un marco de trabajo para el diseño en dos niveles. El primer nivel, llamado meta-diseño, facilita la configuración y administración del proyecto. Para esto, se definen los actores y los recursos de un proyecto para conformar una estructura general del mismo. El segundo nivel, llamado nivel de diseño, comprende el diseño conceptual de la base de datos correspondiente a la abstracción del negocio hacia la implantación de un sistema. Los dos niveles de diseño son abordados con la misma metodología y ambiente visual, en el cual se emplea una símbología simplificada para crear diagramas. La simbología ha sido enriquecida para facilitar la interpretación y revisión a los diagramas creados.

Nuetras herramienta fué desarrollada sobre la plataforma MacOSX en lenguaje C objetivo. Es un producto de software libre que permite visualizar la lista de atributos que compone a cada una de las entidades y relaciones del diagrama conceptual. Permite a su vez asociar un color a los íconos que representan entidades para caracterizarlas con base en el criterio definido por el diseñador, lo cual hace más legible la lectura del diagrama. La implantación de la base de datos del meta-diseño, permite conocer y hacer preguntas propias al proyecto del diseño (avances, tareas pendientes, responsables, retrasos). De esta manera, es posible conocer las tareas inherentes al control y administración de proyectos. El resultado es la implantación de dos bases de datos objeto relacional que pueden ser manejadas por la misma máquina de base de datos.

Abstract

The database design is a elementary stage on developing information systems to garantee their best performance. The traditional conception is to store information of a business model. In this way the usage of database is limited to just one structural level. However, there are inherent activities in the system life cycle related to the proyect control and management for which additional tools are used. So, the need of a meta-database of higer level is proposed for describing the control and management information of development of the database proyect which support the information sistem.

This thesis introduce a methodologie and tool to support the design of databases in EV extended model. Also, a framework for the design in two levels is offered. The first level, called meta-design, makes easy the project configuration and management. For this, the project actors and resourses are defined to shape a general structure of the project itself. The second level, called design level, covers the database conceptual design concerning to the business abstraction towards the system construction. Both levels are approached with the same methodologie and visual environment, which uses a simplify symbology to create charts. The symbology had been riched to make easy the interpretation and review of the charts made.

Our tool has been developed on the MACOSX platform using the C-objective programming language. It is a free software which allows visulize the attribut list, part of each entity and relationship in the conceptual chart. It also allows associate a color to the icons which symbolize entities to be characterized in base to the criteria defined for the designer, making the chart more legible. The construction of the meta-database, allows to know and to ask about the project design itself (progress, pending tasks, people in charge, delays). In this way, it is possible to know the inherent tasks to the project control and management.

The result is the implementation of two object-relational databases which can be enhanced by the same database machine.

Índice general

Índice de figuras

Capítulo 1

Introducción

Una de las etapas fundamentales en la construcción de un sistema de información que maneja una base de datos, es sin duda, el diseño mismo de la base de datos. Si la estructura no es definida apropiadamente se pueden tener demasiados problemas a la hora de ejecutar consultas a la base de datos para obtener algun tipo de información.

El diseño de bases de datos se realiza por lo general en tres fases. La primer fase, llamada diseño conceptual, donde se obtiene una primera representación de alto nivel, para capturar los requerimientos de datos del mundo real. Este diseño es generalemente empleado como una forma simple y significativa de entendimiento entre el diseñador y el usuario final. La segunda fase, llamada diseño lógico, transforma el esquema conceptual al modelo de datos que emplea el SMBD (jerárquico, de red y relacional). La tercera fase, llamada diseño físico, parte del esquema lógico y determina las estructuras de almacenamiento físico en dispositicos de almacenamiento secundario. describe tanto las estructuras de almacenamiento como los métodos usados para accesar efectivamente a los datos.

En este proyecto de tesis se ha desarrolado una herramienta que propociona nuevas caracteristicas que facilitan la construcción, interpretación y revisión de los esquemas conceptuales y además presenta un marco de trabajo para diseñar en dos niveles, meta-diseño y diseño.

Las caracteristicas que sobresalen de esta implementación son:

1. La presentación de un marco de trabajo para diseñar en dos niveles, meta-diseño y el de diseño . El primer nivel, es el nuevo concepto que se adiciona al diseño tradicional. Este meta-diseño es descrito con el mismo formalismo y metodologia, obteniendo como producto final la estructura de una metabase de la base de datos. Esta metabase permite obtener información útil para el control y administración de proyectos ya que aquí se describen los conceptos básicos y funciones que se requieren para el desarrollo de un proyecto, es decir, se define la estructura del proyecto: actores, actividades,

roles, asociaciones, etc. El segundo nivel, es el diseño tradicional de la base de datos que corresponde a la abstracción del negocio que se esta modelando.

2. La propuesta de una símbología para crear los diagramas EVE.

3. La facilidad de asociar colores a los íconos que representan las entidades, interreralciones, y gerarquias de generalización y especialización, permitiendo caracterizarlas con base en un criterio definido por el diseñador.

Nuestra herramienta se ha desarrollado sobre la plataforma MacOSX en lenguaje C-Objetivo, con la finalidad de mantener un estandar en el software de desarrollo del área de investigación. Con el lenguaje C-Objetivo es posible construir interfaces de usuario de alta calidad al utilizar eficientemente la tecnología Macintosh, además permite que se hagan buenos diseños orientados a objetos con alto rendimiento.

La presente tesis esta dividida en 6 capitulos, el capítulo 2 presenta el proceso de diseño de las bases de datos asi como la descripción sintética del problema de tesis y el impacto en las nueva técnologias. El capítulo 3 comienza con la metodología de diseño de las bases de datos ubicando en que parte de esa metodología se emplea nuestra herramienta. Continua con una sección dedicada a la abstracción y modelado de objetos, dando la importancia que esta actividad cobra en el diseño de bases de datos. Así mismo, se describe el método entidad vínculo y sus extenciones, y se realiza un analisis comparativo de la simbología de las diferentes familias que has surgido de este método. Culmina con una descripción general de la herramienta EVEMac. El capítulo 4 presenta el diseño e implementación de la herramienta EVEMac, haciendo distinción entre el diccionario de atributos y el editor de diagramas EVE. El capitulo 5 trata dos casos de estudio para aplicar la metodología basada en el modelo entidad vínculo extendido y apoyada con nuestra herramienta. En el capítulo 6 se resumen las conclusiones obtenidas y el trabajo a futuro.

Capítulo 2

Proceso de Diseño de la Base de Datos

2.1. El planteamiento del problema de diseño

El diseño de una base de datos es un problema principalmente de índole conceptual el cual se traduce en procedimientos pragmáticos de implementación de software. La metodología inicia con la etapa de recopilación y análisis de requerimientos en donde se establece el modelo de empresa, pasando posteriormente por etapas de construcción de co-diseño y refinamiento a través de los distintos niveles y diferentes orientaciones.

El objetivo de un buen diseño es una solución que permita producir la manufactura de un software a la medida: *Base de datos y Sistema de Información.*

Éste deberá cumplir con las expectativas siguientes:

i) Sintácticamente correcto y semánticamente válido.

ii) Suficientemente flexible de implantaciones y facilidades de reingeniería.

iii) Robusto.

iv) Con un buen desempeño para responder eficientemente a problemas que se plantean al sistema.

2.1.1. Metas en el diseño de la base de datos

La meta fundamental de diseñar base de datos es que a partir de una recolección de requerimientos se construya un *Modelo Conceptual de la Base de Datos* que permita llevarlo a un diseño lógico, y que este ˙ltimo a su vez conduzca a su instalación a un esquema físico en un cierto *Sistema Manejador de Base de Datos*. Lo anterior es con el fin mantener de toda la información requerida por el usuario para que el comportamiento de un *Sistema de Información* sea efeciente y adecuado.

Las metas implícitas de diseño de una base de datos son:

1. Encontrar toda la información de requerimientos de toda la gama de usuarios en una área de aplicación.

2. Proporcionar de manera natural y fácil de entender la estructura del contenido de la información.

3. Guardar el diseño de toda su información semántica para posibles procesos de rediseño posteriores.

4. Archivar todos los requerimientos de proceso y también sus parámetros de grado de eficiencia.

5. Archivar independencia lógica de consultas y formulación de transacciones.

2.1.2. Extensión del problema de diseño

La extensión del problema de diseño se relaciona directamente con dos aspectos básicos: el número de etapas de la metodología y los modelos formales de computación que intervienen en el proceso de diseño. Acerca del primero los métodos son los que establecen las etapas que se suceden a través del ciclo de vida del software. Para el segundo, el planteamiento de los modelos formales permite manejar en cada una de las etapas una sintaxis y formas de evaluar la semántica que permita validar la información. Con esto, se tiene cada vez un mayor refinamiento de las tareas con el fin de obtener la mejor base de datos posible.

La manufactura de una base de datos debe considerar continuas actualizaciones y cambios, así como, la incorporación de nuevas aplicaciones durante su ciclo de vida. Por consiguiente, el proceso de diseño conceptual de una base de datos, ha llegado a la necesidad de cubrir varios aspectos observando amplias extensiones al problema de diseño de una base de datos.

El problema inicia de manera difusa con las tareas de recolección de requerimientos que hasta cierto punto son informales. Posteriormente, en cada etapa el propósito es decodificar la información y, mediante modelos formales, el objetivo es mantener la correctitud y coherencia entre los distintos niveles de desarrollo. Es importante la validez semántica de la información en cada uno de ellos. En su conjunto se pretende cumplir con el mejor diseño y la más eficiente implantación de la base de datos.

Justificación en extensiones

- Los proyectos grandes requieren de una base de datos del proyecto. Con el mismo modelo entidad-vínculo podemos definir la estructura del proyecto: actores, documentos, versiones, actividades, entidades de proyectos, asociaciones y roles. Las entidades y esquemas son sujetos para la administración del proyecto de diseño y rediseño de la base de datos.

- Los sistemas de base de datos necesitan tener disponible la mayor información en sus diferentes orientaciones. La metodología contempla una visión completa del diseño y desarrollo basado en: estructura, operacional y comportamiento.
- La recolección de requerimientos se mantiene como un talón de Aquiles para el desarrollo de sistemas. La adquisición difusa de la información requiere de mejores métodos que capturen mayor semántica. Descripción de la información que apunte a una decodificación correcta.
- Se requiere captura de información que contemple mantener coherencia entre los distintos niveles de desarrollo: conceptual, lógico, físico.
- Un adecuado análisis de requerimientos permite codificar la estructura y semántica de acuerdo a su aplicación y comportamiento. Las operaciones actúan sobre los datos y los eventos activan operaciones. El análisis funcional aplicado a los datos da lugar a una serie de esquemas que deberán ser almacenados y usados. Las reglas deberán ser almacenadas como triggers para disparar aplicaciones que usan datos a su vez.
- El nivel de meta-datos contempla las restricciones de integridad estáticas y dinámicas.
- Es necesaria la creación de interfaces de usuarios homogéneas y amigables con estándares en grupos de trabajo. Este es un requerimiento necesario para la automatización del diseño de bases de datos.

2.1.3. Complejidad del problema de diseño

La principal complejidad del proceso de diseño está determinada por la complejidad y el número de términos incluídos en la implementación de los diversos esquemas que intervienen en la implementación de la base de datos. Por otro lado, la semántica inherente en la información se declara en la estructura de la base de datos y las operaciones, proporcionarán una semántica de comportamiento importante para su almacenamiento.

Aplicaciones como: bases de datos en internet, almacenes de datos y bases de datos multidimensionales; requieren información adicional referida al comportamiento de automatización con consideración a interfaces de usuario. En este contexto, los usuarios se constriñen a: diferentes semánticas, diferentes estilos de trabajo y diferente contextos, que se llevan a un registro de los datos vistos como *microdatos* hasta *macrodatos*.

Con respecto a la complejidad hay algunos términos que debemos destacar:

- **Complejidad computacional**.- El comportamiento de la base de datos depende de las operaciones que se usan para la consulta y actualización de la bases de datos. Se considera la frecuencia de utilización de las diversas consultas. La complejidad computacional es influenciada por la cantidad de datos que se necesitan gestionar. Además, el comportamiento depende de la complejidad del mantenimiento para las diferentes operaciones de la

base de datos. Si las interfaces de usuario son consideradas en el sistema, entonces la complejidad de su generación deberá tomarse en cuenta.

- **Complejidad de almacenamiento**.- En general, las bases de datos son muy grandes, por lo que la redundancia en los datos puede generar un problema de computación y almacenamiento. La redundancia de los datos es uno de los principales temas durante la normalización. La reducción de datos requiere algunas acciones de acuerdo a su mantenimiento y consistencia. Las interfaces de usuario con frecuencia despliegan agregación de datos. Si su generación resulta demasiado compleja, entonces la materialización de vistas parece ser una solución adecuada. Este enfoque incrementa la complejidad de almacenamiento.
- **Comprensibilidad**.- Una característica principal de las bases de datos es su largo período de vida, que se refleja en su proceso de diseño. Por tal motivo requerimos de un diseño de esquemas simples y entendibles que permitan una fácil y correcta implementación, para realizar posteriormente el mantenimiento de forma sencilla.
- **Claridad**.- Los esquemas de bases de datos requieren ser especificados claramente y documentados cuidadosamente, de tal forma que puedan ser mantenidos por otras personas. Si un esquema es simple y entendible, entonces resulta fácil de describir. Con una buena documentación, las modificaciones al esquema original pueden ser llevadas a cabo por otras personas que no es el diseñador original.

2.2. Descripción sintética

De lo expresado anteriormente se desprende que el *Diseño de las bases de Datos*, enfrenta fuertes problemas debidos a tres causas: su definición estructural (sobre todo cuando se tienen datos complejos) los objetivos de los sistemas y la extensión del problema.

En esta tesis se expone una metodología y el sistema para ayudar al diseño de bases de datos. Basándonos en la metodología entidad-vínculo-extendido, se propone un desarrollo de diseño desde el nivel alto de meta-diseño hasta el diseño e implementación de una base de datos. Su aplicación y desarrollo se lleva a cabo a través de un ambiente visual general que es parte del sistema de software CADBD que tiene la finalidad de automatizar todo el proceso.

2.2.1. La metodología y los dos niveles de diseño

Desde tiempo atrás se ha expuesto la idea de la incorporación de nuevos niveles de estructuras de datos que permitan una descripción de niveles inferiores de la información[1].

[1]El mecanismo original implementado fue el llamado descriptor de archivos [9] y [10]. A través de dicha estructura de almacenamiento, se puede determinar el agregado de información: archivos, registros,

El objetivo de tener estos niveles de almacenamiento de información es que plantean en su generalidad una meta base de datos que permita una descripción semántica de la base de datos.

Durante el proceso de maduración del concepto algunas preguntas han sido elaboradas:

- ¿Tenemos una metodología que permita extraer la información de meta-datos en la recolección de requerimientos?
- ¿Podemos plantear un modelo conceptual el cual sirva para una estructura lógica?
- ¿Qué tan sencillo es llevar a cabo actualizaciones a este nivel?
- ¿Cómo llevar a cabo las implementaciones de manera flexible y automática?

En este proyecto se contemplan dos niveles principales: meta-diseño y diseño conceptual; enfrentados con una misma metodología y ambiente visual, a través de la construcción de una herramienta de software que ayude al diseño de la base de datos. El proceso de diseño se lleva a cabo a través de tres orientaciones complementarias: estructural/semántica, operaciones y eventos.

- **Nivel de meta-diseño**.- En este se definen los actores y los recursos definiendo una estructura general del proyecto en base a su metodología. Se obtiene una base de datos de meta-diseño en el modelo relacional. Sobre ésta podemos aplicar los mismos lenguajes visuales del sistema propuesto, con el fin de hacer consultas para el control del proyecto y reingeniería. En esta etapa se establece la base de meta-datos.
- **Nivel de diseño**.- Del nivel anterior recurrimos a los recursos generados por el proceso para iniciar un diseño conceptual mediante la misma metodología y herramienta. Se obtienen modelos lógicos y esquemas físicos sobre la misma máquina de base de datos. En esta etapa se obtienen: una base de datos y base de reglas.

El proyecto contempla inclusión de modelos formales y técnicas de ingeniería de software, con el propósito de lograr la base de datos con el mejor desempeño y calidad. El producto es un software que está sujeto a una gran cantidad de cambios durante todo su ciclo de vida, lo cual requiere de una amplia documentación y métodos adecuados para el rediseño de la base de datos.

2.2.2. El modelo Entidad-Vínculo-Extendido

El modelo se basa en representaciones gráficas o pictográficas de todos los conceptos abstractos, lo cual conduce a un modelo de fácil entendimiento y uso. Una ventaja importante es la gran flexibilidad para la extensión de los sistemas e integración de vistas

campos y datos. El concepto sirvió como mecanismos de acceso a base de datos, en construcción de un sistema manejador en el CINVESTAV

que ofrece al diseñador un panorama completo de la base de datos mediante un diagrama completo.[2]

Por otro lado, la estructura misma del modelo, con un enfoque de edición dirigida por la sintaxis, permite llevar a cabo implementaciones seguras y correctas a un modelo lógico, lo que ofrece una característica de ser sólido el modelo. Actualmente, su enfoque como metodología permite un uso amplio en sistemas de información, bases de datos e ingeniería de software.

2.2.3. La herramienta visual de diseño

- LIDA (**Lenguaje Iconográfico para el Desarrollo de Aplicaciones**). Es un lenguaje visual de flujo de datos para desarrollar aplicaciones sobre una base de datos. Su programación visual se basa en gráficas de arcos e iconos llamadas flujogramas. Mediante un flujograma es posible plantear una consulta a una base relacional o desarrollar una aplicación en donde se incluyan algunos procesos.
- EVEX (**Entidad-Vínculo-Extendido**). Es un sistema visual para el diseño conceptual de base de datos basado en el modelo entidad-vínculo de P. Chen. EVEX usa el paradigma de T. Teorey, que permite extensiones a vínculos ternarios con jerarquías de especialización y generalización. El resultado es un modelo semántico extendido con una inmediata orientación al modelo de objetos. Mediante EVEX es posible editar diagramas entidad-vínculo-extendido y traduce los esquemas conceptuales a esquemas lógicos relacionales normalizados con sus dependencias funcionales.
- Petra (**Ambiente de simulación y análisis de redes de Petri**). Es una aplicación, para Mac OS X que permite editar y ejecutar redes de Petri.

2.3. El proceso de diseño y su impacto en nuevas tecnologías

Actualmente, uno de los problemas de diseño de base de datos es que no se encuentra alineada a las nuevas tecnologías de bases de datos. Aunque los sistemas relacionales de datos son los mejor posicionados, con estandares bien establecidos; el surgimiento de una nueva generación de aplicaciones propícia la creación de infraestructuras de tecnología de la información, la cual envuelve a las metodologías y tecnologías de diseño. Un principio, importante que hemos considerado siempre es el uso de tecnología abierta, con una sólida fundamentación y el desarrollo de nueva herramienta como contribución en el campo, como puede observarse en la generalidad del proyecto CADBD.

Algunos problemas abiertos, donde se requiere de mejores tecnologías de diseño, se presentan en seguida.

[2]En este proyecto estamos considerando la implementación de EVEX como ambiente de diseño y LIDA (Lenguaje Iconográfico para el Desarrollo de Aplicaciones) para las consultas y aplicaciones en una base de datos

2.3.1. Estructura de la base de datos y problemas complejos de consulta

En base de datos la manufactura del software es conducida por la vertiente de estructura y aplicaciones, las cuales se encuentran interrelacionadas. De esta forma, los objetos que intervienen en las bases de datos están determinados por la combinación de sus tres enfoques: estructural de datos, operacional que actua sobre los datos y el comportamiento, dado este último por los eventos que activan los procesos.

Un problema actual es la ampliación cada vez mayor en la aplicación de las áreas, generado por estructuras de datos y procesos más complejos que deberá ser activados automáticamente.

Consultas dinámicas

"Queries" dinámicos y de multiresolución son nuevas aplicaciones de bases de datos de imagenes y un sistema dinámico de consulta, requiere de cierta investigación de lenguajes de consulta. Por un lado existen sistemas experimentales donde las consultas se llevan a cabo a través de la imágen (por ejemplo, señalamiento de un rio en un mapa), produciendo una búsqueda a través de la base de datos. Este tipo de problemas en general son determinados como búsquedas a través del contexto de imagen. El segundo, se refiere, a llevar al cabo una serie de consultas guiadas mediante la modificación dinámica de ciertos parámetros. Cuando este sistema esta acoplado a un sistema de visualización y computación numérica nos ofrece un ambiente magnífico para el *análisis exploratorio de datos*.

Lenguajes y ambientes visuales

Señalando de manera precisa podemos observar algunos aspectos importantes:

- Ambientes visuales para la administración del proyecto del proceso de diseño.
- Lenguajes que permitan especificaciones de manera interactiva.
- Lenguajes para la especificación automática de procesos.
- Un ambiente unificado y estandar que pueda integrar la mayoría de los lenguajes.
- Lenguajes simples que puedan llevar al cabo eficientes implementaciones de conceptos y conocimiento.

2.3.2. Nuevas tecnologías en bases de datos

Actualmente, uno de los problemas de diseño de base de datos es que no se encuentra alineada a las nuevas tecnologías de bases de datos. Aunque los sistemas relacionales de datos son los mejor posicionados, con estandares bien establecidos; el surgimiento de una

nueva generación de aplicaciones propícia la creación de infraestructuras tecnológica en la cual envuelve a las metodologías y tecnologías de diseño. Un principio, importante que hemos considerado siempre es el uso de tecnología abierta, con una sólida fundamentación y el desarrollo de nueva herramienta como contribución en el campo, como puede observarse en la generalidad del proyecto CADBD.

La creación de las nuevas tecnologías de base de datos estan muy de acuerdo con sus aplicaciones: biomédicas, ingeniería y espacios temporales (sistemas geográficos).

Bases de datos biomédicas

El problema de base de datos en biología y medicina corre casi en la misma dirección de las oportunidades de investigación que tienen las bases de datos para este siglo 21. La naturaleza del campo de la biología requiere de *modelos formales de nuevos tipos de datos*, que vayan de acuerdo a la complejidad semántica de los datos. Siendo éste, uno de los problemas más importantes y delicados de las bases de datos.

Tener un buen proceso de diseño con sus métodos y herramienta que fácilite su automatización permite incorporar información valida que se traducirá en la confiabilidad de la información y obtención de resultados válidos.[1]

Base de datos para ingeniería

En aplicaciones como CAD/CAM. Los objetos a tratas son muy complejos. Las bases de datos almacenan un número pequeño de objetos de objetos complejos, los cuales requieren de definiciones funcionales diferentes para casi cualquier objeto. En este caso se requiere de la especificación de lenguajes específicos que permita el mapeo de especificaciones del objeto a eficientes estructuras de almacenamiento.

Base de datos geográficas

Bases de datos geográficas con aplicaciones a modelos dinámicos de ecosistemas, es el ejemplo clásico de bases de datos espacio-temporales. Este tipo de base de datos incorpora una serie de objetos complejos y clases heterogéneas las cuales interrelacionan dos tipos principales: información nominal e información geográfica. Esta interdependencia da lugar a una gran cantidad de contenido semántico el cual se refleja en un problema en el diseño conceptual.

Los programas de los Sistemas de Información Geográfica (SIG), a diferencia a la de negocios, deben proveen las funciones y las herramientas necesarias para almacenar, analizar y desplegar la información geográfica. El punto importante es la diferenciación entre representación y semántica. Los objetos geométricos como punto, linea, polígono; tienen asociada su semántica a un objeto geográfico el cual puede estar determinada a algún

[1] En el desarrollo de la base de datos de la *Colección Nacional de Microorganismos*, el proyecto fue sometido ha un escrupuloso escrutinio, por parte de la CONABIO, acerca de la calidad de los datos. Esto significó el desarrollo de un buen diseño de base de datos.

tema. Su implementación requirere de un buen modelo a nivel conceptual que permita un panorama interpretativo substancial.

2.3.3. Proceso de diseño integral y sus mecanismos de almacenamiento

Para las nuevas tecnologías de bases de datos mencionadas anteriormente se requiere mecanismos de almacenamiento que deben ser considerados en el proceso de diseño:

- Mecanismos para la adquisición y especificación de datos complejos: documentos, formas, fuentes geográficas.
- Modelos de datos objeto-relacionales.
- Mecanismos para almacenar metadatos. Estructuras de almacenamiento independientes entre la base de datos y metadatos.
- Procedimiento de gestión: consultas y aplicaciones; que respondan al tipo de base de datos.

De acuerdo con los proyectos que se han venido realizando y tendencias que concuerdan con nuevas direcciones de investigación en base de datos tenemos lo siguiente.

Tratamiento de nuevos tipos de datos y almacenamiento terciario

Computación en datos biológicos, médicos, científicos y aplicaciones a la industria de gran escala, requiere del manejo de información de multimedia. Es necesario la implantación de computación de alto desempño, requiriendo de un tipo de almacenamiento terciario donde se tengan las herramientas de software que puedean manejar diferentes niveles de almacenamiento de datos. Los distintos, niveles estarán determinados por su tecnología de almacenamiento y requieren de un modelo que permita su organización e integración.

Repositorios

Muy de acuerdo con el punto de almacenamiento terciario, se encuentra una gran clase de aplicación denominada de *Repositorios*. Estos se caracterízan por almacenar y manejar datos y metadatos (descripción semántica y estructural de los datos). El proyecto CADBD, esta dirigido en parte, a la construcción de los repositorios teniendo como objetivo un mismo ambiente (modelo y tecnología), en donde los diferentes niveles puedan ser trabajados desde un enfoque común, visto como un *Sistema Gestor de Repositorios*. Las principales tareas de los repositorios son:

- Deberán mantener un desarollado conjunto de representaciones de la misma o información similar.
- Deberá soportar la información acerca de las versiones y configuraciones que se llevan al cabo a través del tiempo.

- Deberá mantener los cambios en la estructura de la base de datos y la limpieza de los metadatos.

2.3.4. Ingeniería y calidad de datos

El problema de validación de los datos de una base de datos requiere de mecanismos confiables para la entrada de los datos y herramienta que con cierta frecuencia este validando los mismos en la base de datos.[3]

En las bases de datos se incluye una combinación de información que proviene de distintas fuentes y, por consecuencia distintas confiabilidades. Por consiguiente, se requiere de herramienta que pueda evaluar la confiabilidad de la información y, que la mantenga válida en las bases de datos. Además, se requiere los sistemas de consulta contemplen la confiabilidad mediante los aspectos de *linaje de la información*.

Integración de datos y conversiones

La semántica de la información es uno de los puntos más difíciles que enfrentan las bases de datos. El punto anterior, referido tanto a la sintaxis como a la semántica, toma mayor importancia cuando incluimos una interconexión de fuentes de información con una variedad de formatos y modelos. Cada fuente esta envuelta por una componente que traduce entre el punto de vista de la fuente y el que comparte el punto de vista global. En este sentido, sólo pensar en la correctitud de los datos no es suficiente. Ahora es necesario incluir herramientas de *Diccionarios Semánticos de Datos*, los cuales permitan llevar a cabo traducciones correctas entre las distintas entidades. Este es un *problema de ontologías*, que se presenta primordialmente en base de datos abiertas, por ejemplo los *Open GIS*.

[3]Desde la instalación de la base de datos CDBB-500 de microbiología, una de las principales preocupaciones ha sido la calidad de los datos.

Capítulo 3

La Metodología, el Modelo y la Herramienta de Diseño

El diseño de bases de datos es una actividad de vital importancia dentro del ciclo de vida de los sistemas de información. Y el objetivo es crear la estructura de una base de datos confiable con el mejor desempeño. Para lograrlo es necesario seguir una serie de pasos establecidos que nos auxilien en esa labor. En este capítulo presentamos una metodología basada en uno de los modelos conceptuales más ampliamente utilizados el cual es adoptado por la herramienta de diseño de bases de datos creada en este proyecto de tesis.

3.1. Metodología de diseño de base de datos.

3.1.1. Estructura general de la metodología.

El primer nivel de la estructura general esta determinado; por un lado por la interfaz que existe entre la estructura organizacional y la distribución de la base de datos, y por el otro, la estructura de los objetos de la base de datos. El resultado es una concepción alta de diseño con un enfoque de interacción entre análisis funcional y diseño conceptual de datos.

Clasificación de requerimientos.

El diseño implícitamente debe satisfacer un conjunto limitado de requerimientos:

- Requerimientos básicos y generales relacionados a ingeniería de software:
  - correctitud
  - desempeño en el tiempo y el espacio.
  - compatibilidad con las restricciones organizacionales.
- Requerimientos especiales de bases de datos:

  - disponibilidad
  - confiabilidad,
  - Contar con un administrador de datos.
  - compatibilidad con las restricciones semánticas.

- El objetivo del proceso de diseño es el resolver el problema mencionado anteriormente. Por consiguiente cada etapa de este proceso produce una cierta porción de especificaciones.
- Sin importar la metodología usada, un proceso de diseño deberá resolver los problemas mediante un conjunto limitado de tareas.
- Cada proceso de diseño se basa en conceptos específicos, técnicas y modelos formales.
- Cada etapa del proceso de diseño de la base de datos puede ser vista como una transformación, en donde es necesario mantener correctitud sintáctica y preservar la validez semántica.

La perspectiva de diseño

El diseño de software y base de datos puede ser llevado a través de tres perspectivas distinguibles:

1. La perspectiva orientada a estructura se enfoca a la descripción estructural de los datos. En general, todos los enfoques de diseño de base de datos reconocidos se basan en la perspectiva orientada a estructura, uniéndose con mucha frecuencia con una perspectiva semántica. En este caso el diseño de la estructura es combinado con el diseño de restricciones de integridad estáticas. Existe una taxonomía importante en las estrategias de diseño orientado a estructura, la cual puede ser tomada en cuenta para una etapa de meta-diseño.
2. La perspectiva orientada a comportamiento, corresponde con tratamiento de los eventos y las acciones que se llevan a cabo en una base de datos durante su ciclo de vida. Los modelos formales pueden ser los siguientes: reglas de evento, condición y acción. redes de Petri, y sistemas de transición de predicados.
3. La perspectiva orientada a procesos, corresponde con las operaciones de la base de datos.

Por lo tanto, de acuerdo a resolver el diseño general se requiere de un modelo basado en una teoría genérica de las etapas de diseño. Este modelo puede ser usado también como reingeniería para bases de datos existentes. En este caso se sugiere buscar la descripción de las bases de datos para las trazas de cada uno de los procesos específicos de diseño. A partir de los requerimientos de aplicaciones y datos que se presentan en el mundo real el problema de diseño de una base de datos se plantea mediante una serie de etapas que se presenta en la figura de estructura general de metodología.

3.1.2. Recolección y análisis de requerimientos.

El problema en general

La recolección y análisis de requerimientos inicia a partir de la especificación de los objetivos y la misión a la cual se designó el Sistema de Información. El objetivo es encontrar y coleccionar toda la información requerida por toda la gama de usuarios los cuales se encuentran ubicados en diversos ambientes de la organización (vistas). Cada uno de ellos producen sus propias especificaciones de requerimientos dentro de una área de aplicación. Las distintas necesidades informáticas relativas a cada uno de los ambientes de una organización requiere de un profundo y cuidadoso análisis de requerimientos, procurando resolver de la mejor manera los posibles conflictos entre usuarios, con el objetivo de obtener un diseño óptimo de una base de datos. Los requerimientos pueden ser clasificados en tres categorías generales de información:

1. Restricciones corporativas son las descritas como reglas de una organización y políticas operacionales. Así también como restricciones: temporales y espaciales, financieras y legales.

2. Requerimientos de procesamiento son las que describen la planeación, control y la operación de los sistemas.

3. Requerimientos de información son los que describen las estructuras de datos, entidades, asociaciones y restricciones de integridad semántica de los objetos.

El análisis, la especificación y el modelado de esas tres categorías de información son etapas esenciales en el diseño de bases de datos. Debido al enfoque de análisis funcional y estructura, se tiene una interrelación muy estrecha entre las categorías de información que llegan a producir conflictos. La primera categoría puede ser considerada como una clase especial de conocimiento de sentido común. Desde el punto de vista de la inteligencia artificial., nosotros podemos diferenciar entre entre las distintas clases de conocimiento que se usan durante el proceso de diseño. En principio, la teoría que sustenta los modelos y herramientas podrán ser considerado s como conocimiento de estrategia o meta-conocimiento del proceso de diseño. El soporte algorítmico, es considerado como el judgemental knowledge, del proceso de diseño debido a que describe el razonamiento del proceso de diseño. Y por último la información usada por el diseñador durante el diseño es considerada como conocimiento factual, que a su vez visto como sistema basado en conocimiento tiene la categoría extensional e intencional.

La adquisición de información.

El procedimiento inicial de recolección de la información fundamentalmente consiste en entrevistas y comunicación oral, enfrentando el problema de la informalidad. Las tareas hasta cierto punto son difusas ocasionando con frecuencia ambigüedades en la información adquirida y problemas semánticos. Un objetivo primordial es encontrar la forma más natural y método con sintaxis dirigida, el cual permita extraer el mayor contenido semántico de

la misma. Con ésto, se persigue obtener los sistemas de información que mejor cumplan con la misión para la cual fueron pensados. En esta etapa el método inicia con la construcción esquemas globales para cada uno de ambientes. Sobre esta información es necesario aplicar filtros de ambigüedades y la partición del discurso en conjuntos de sentencias homogéneas, con el propósito de: reducir expresiones repetidas, obtener clases de expresiones que son sinónimos y la introducción de substitutos apropiados. De este análisis se desprenden conjuntos de sentencias las cuales son expresadas en formas de requerimientos de: datos, procesos y eventos.

En el modelado de las bases de datos se pueden identificar tres perspectivas:

1. La orientada a estructura, es un enfoque de perspectiva semántica

2. La orientada a procesos, es la perspectiva que concierne a los procesos o actividades llevadas a cabo en una área de aplicación.

3. La orientación a comportamiento, es la que corresponde a los eventos que suceden en el mundo real y disparan acciones en los sistemas de bases de datos.

Análisis de requerimientos.

Las sentencias de datos expresan el conocimiento de entidades y sus propiedades. Así como, las asociaciones que existen entre ellas. Las sentencias de operaciones expresan el conocimiento de las operaciones que actúan sobre los datos: modificación, transformación, o presentación de ellos; además, de la frecuencia que ocurre cada operación y la manera de como se realiza. Finalmente, las sentencias de eventos expresan el conocimiento de cómo ciertas condiciones se deben satisfacer para efectuar ciertas operaciones. Las sentencias pueden incluir varios eventos, que determinan varias condiciones y diferentes operaciones que se ejecutan concurrentemente o alternativamente. La obtención de una especificación precisa de los requerimientos, se realiza mediante la trascripción o decodificación de la información: en datos, operaciones y eventos; en sus respectivos glosarios.

3.1.3. Diseño conceptual de vistas.

En esta etapa se toman los resultados del análisis de los requerimientos para llevar a cabo la construcción de esquemas de datos, operaciones y eventos mediante un modelo conceptual. El objetivo es proceder hacia especificaciones formales de las diversas vistas del sistema. El modelo conceptual se basa en un modelo específico que usualmente se elige de los modelos semánticos. Estos modelos representan explícitamente la semántica de los datos, describiendo apropiadamente las propiedades dinámicas y su evolución de acuerdo a la ocurrencia de los eventos que se suceden en la vista. El esquema de datos permite especificar formalmente la información relativa a un ambiente de la organización y el proceso es la creación de tantos esquemas, de datos como vistas existan en la organización. El esquema de operaciones es un subconjunto de esquemas de datos que se requieren para realizar la operación referida. Existen tantos esquemas de éstos como operaciones en el

ambiente existan, usando la misma especificación formal del modelo de datos. El esquema de eventos es una especificación formal de los eventos relevantes a todas las funciones del ambiente, con una, especificación formal basada en redes de Petri.

3.1.4. Integración de vistas

Esta etapa tiene como insumo todas las vistas conceptuales definidas independientemente durante la etapa anterior de diseño. Ahora el objetivo es combinar en una vista global única de la aplicación, obteniendo lo que se conoce como esquema global conceptual. Los diferentes enfoques que tienen los usuarios acerca de la aplicación y las diversas representaciones que son equivalentes. requieren de un proceso complejo de integración de vistas. En proceso de integración se toman en cuenta la integración de esquemas de datos, los de operaciones y finalmente los de eventos.

El proceso anterior tiene como aspectos importantes los siguientes:

- Integración del esquema de datos.
- Análisis de conflictos.
- Unión y análisis de redundancias.

Archivar independientemente los esquemas conceptuales, lógicos y físicos, con el objetivo primordial de tener independencia en base de datos. Crear una base de metadatos, con el objetivo de la creación de esquemas de análisis semántico de la información para sistemas de verificación automática de la integridad de los datos.

El modelo que comunmente es utilizado para llevar a cabo esta etapa, es el entidad vínculo que estudiaremos ampliamente en la sección 3.3.

3.1.5. El diseño lógico.

El objetivo del diseño lógico es la conversión del esquema global conceptual a un esquema lógico que será usado como entrada a un sistema manejador de bases de datos. La traducción del esquema conceptual global al modelo lógico destino es un proceso de refinamiento el cual debe garantizar un diseño aceptable orientado a minimizar los accesos lógicos que puede requerir un proceso de desnormalización. La entrada a esta etapa son el esquema global conceptual y los subesquemas de operación, con el objeto de:

- Simplificación del esquema conceptual.
- Refinamiento del esquema simplificado
- Generación de la descripción relacional.

Simplificación del esquema conceptual.

Algunas estructuras conceptuales tienenun mapeo inmediato a las estructuras del modelo relacional. Sin embargo, existen otras relaciones complejas las cuales el mapeo no es de forma inmediata, requiriendo de esta forma de un proceso de transformación o de rediseño, por ejemplo:

- Rediseño de relaciones circulares.
- Conversión de relaciones complejas.
- Selección de identificadores principales.

Refinamiento del esquema simplificado.

La tarea de refinamiento consiste en reestructurar el esquema simplificado con el fin de obtener un nuevo esquema, con el objetivo de optimizar la ejecución de las operaciones más importantes y complejas:

- Partición de entidades.
- Replicación de atributos.

Generación de la descripción relacional.

Finalmente, a lo que se quiere llegar es a una descripción del modelo relacional de datos normalizado y con la introducción de nuevas operaciones para preservar las restricciones de integridad. Los esquemas relacionales servirán de entrada para generar los esquemas físicos de implementación a un sistema manejador de bases de datos.

3.2. Abstracción y modelado de Objetos

En los primeros pasos de la metodología analizada en la sección anterior, vimos que una de las tareas es analizar los requerimientos para posteriormente realizar el diseño conceptual de vistas. Entre estas dos etapas se lleva a cabo un proceso de abstracción de la realidad cuyo objetivo es elaborar un esquema conceptual. En este esquema se modelan aquellos aspectos relevantes de la realidad de un problema, para luego incluirlos en una base de datos que servirá de base para un sistema de información que automatizará los procesos que requieran los usuarios. Por tanto, decimos que de la *realidad* pasamos al *modelo* aquello que es importante para el uso que se le va a dar al modelo y desechar o ignorar los detalles innecesarios. Es necesario contar con mecanismos para poder destacar lo importante y desechar lo irrelevante de una realidad. A estos mecanismos se les llama, *abstracciones*. Las abstracciones son un proceso mental que se aplica al seleccionar algunas características y propiedades de un conjunto de objetos y excluir otras no pertienentes [3]. Las abstacciones ayudan al diseñador y al usuario a entender, clasificar, modelar y usar el conocimiento. Según [11] existen diferentes tipos de abstacciones:

1. La abstracción del concepto, de la estructura del concepto, o de la construcción, son usados para clasificar, agregar y generalizar conceptos:

   - *Clasificación/Instanciación.* Es un proceso mediante el cual un conjunto de la realidad, con propiedades comunes, se agrupan en una sola clase, considerado como un concepto de alto nivel. La clasificación puede ser organizada dentro de una jerarquía de niveles de conceptos.
   - *Agregación/Descomposición* es una forma de abstracción en la cual una relación entre n componentes es considerado como un concepto agregado simple de alto nivel. En otras palabras, una agregación define una nueva clase a partir de un conjunto de otras clases, que representan a sus partes componentes. Cada clase componente forma parte de la nueva clase. Hay varias formas que puede adoptar la agregación:
     - Una clase de objetos físicos está compuesta de otras clases de objetos físicos, por ejemplo una automovil que esta compuesto de llantas, rines, chasis, asientos, etc.
     - Un concepto y los atributos que lo describen, por ejemplo, una persona esta descrita por su nombre, RFC y dirección.
     - Un concepto y los conceptos q lo definen, un ejemplo de esta variante es cuando un propietario se define como la persona que es dueña de un edificio, por tanto propietario se define como la agregación de persona y edificio.

     En su forma más general, una agregación es una correspondencia que se establece entre clases, e indica que los miembros de las clases componentes forman parte de los miembros de la clase agregada.
   - *Generalización/Especialización* es una forma de abstracción en la cual un subconjunto de relaciones entre dos clases concepto es considerado como un concepto generico de alto nivel. Es decir, define una interrelación de subconjunto entre los elementos de dos o más clases.

2. Abstracción de Localización o abstracción de contexto. Esta forma de abstracción sirve para "factorizar", conceptos de patrones repetitivos, compartidos o locales y funcionalidad de conceptos individuales dentro de un ambiente de aplicacion de base de datos compartido. Los objetos pueden considerarse dentro de diferentes contextos tal como: tiempo, espacio, usuarios, ambientes y vistas. Por lo tanto, la globalización de objetos tambien tiene que estar basada en contextos que constituyen la dificultad principal de la intregración de vistas y sistemas de bases de datos distribuidos o federados. El nombramiento es un mecanismo basico para lograr la localización. La parametrización puede ser usada para la abstracción en descripciones parciales de objetos. El Binding y el em mapping son usadas para relacionar conceptos locales a otros conceptos. La abstracción de la localización puede ser usada en una parametrización estructurada en árbol tree-estructured. En los fundamentos de aspectos la

abstracción usa una parametrización tree-structured, dónde la raíz sólo espedifica la interfaz de entrada y salida.

3. Abstracción de Implementación o modularización. Es unada para selectivamente conservar información acerca de estructuras, semantica y comportamiento de conceptos definidos por los dos mecanismos previos de abstracción. La abstracción de implementación es una generalización de *encapsulación* y *alcance*. Este provee *independencia de datos* por la implementación. Permitiendo cambiar la parte privada de un concepto sin afectar otros conceptos que usan ese concepto. Hay dos métodos principales para aplicar la abstracción de implementación:

   - *Ocultar:* Algunos compones deben ser públicos, mientras que otros son privados o protegidos (visibles a sunconceptos).
   - *Mapear:* Hay funciones que definen un mapeo entre los componentes publicos, privados y protegidos.

El poder de los principios de la abstracción proviene de su ortogonalidad y su universalidad. Todos esos mecanismos pueden ser combinados incluso con más mecanismos tales como la excepción y reglas por defecto para poder abstraer la realidad.

Es necesario la abstración para concevir la semnatica. La adquisición de la semantica de la información es similar a la adquisición del conocimiento. Podemos distinguir diferentes tipos de diseño de conocimiento:

1. **Conocimiento del dominio** es la información específica acerca del área de diseño, Esta puede ser dividida en conocimiento efectivo y de control.

   - *Información efectiva* describe objetos tangibles o abstractos y sus propiedades tangibles o abstractas, respectivamente.
   - *Reglas* son usadas para la representación del uso de conocimiento efectivo.
   - *Información de control* representa conocimiento acerca de la aplicabilidad de las reglas y la selección apropiada de hechos.

2. **Conocimiento del sentido común** es información acerca de la estructura del mundo externo que se adquiriere y aplica sin un esfuerzo concentrado de algún humano. Esta información no es frecuentemente representada durante el proceso de diseño.

3.3. El modelo entidad-vínculo

El modelo entidad vínculo es un esquema conceptual que nos permite describir globalmente una base de datos, sin mostrar detalles del las estructuras físicas. En esta sección se presentan primeramente los conceptos básicos del modelo estructural, continuando con las extensiones asi como los mecanismos de abstracción que se utilizan especificamente en este modelo.

3.3.1. El modelo estructural

El modelo entidad-vínculo fué propuesto para modelar conceptualemente bases de datos y fué introducido por Peter Chen en 1976 [7]. El modelo es fácil de usar y de entender, debido a esas carácteristicas ha sido muy popular. Muestra claramente todos los tipos de abstracciones de conceptos, víclos, mapea restricciones y cardinalidades. Su versión básica trataba con propiedade estructurales más estáticas como son: entidades, atributos y vínculos.

Entidad. Es considerada una "cosa" del mundo real con existencia independiente. Una entidad puede ser un objeto con existencia física como una persona, un automovil, una casa o un empleado; o un objeto con existencia conceptual, como una compañia, un puesto de trabajo o un curso escolar. Las entidades se representan gráficamente por medio de un rectangulo como se muestra en la figura 3.1a.

Atributos. Representan las propiedades específicas que describen a las entidades o a los vínculos. Por ejemplo, una entidad empleado puede describirse por su nombre, su edad, su dirección, su salario, y su puesto de trabajo. Al atributo o conjunto de atributos que distinguen de manera única una instancia de entidad y se le denomina indetificador. Cuando el identificador se compone de más de un atributo es posible que en el modelado se este omitiendo algún vínculo con la entidad del identificador. Si esto no es así, y si el identificador está compuesto de tres o más atributos se le suele crear un atributo artificial numérico simple y corto. A los atributos que no forman parte del idenficador, y por tanto no distinguen de manera única una instancia de entidad, se les denomina descriptores.

Vínculo/Interrelación o relación. Es una asociación entre entidades y pueden contener atributos descriptivos inherentes, estos describen características del vínculo y no de alguna de las entidades participantes de la relación. Los vínculos son representados gráficamente con rombos, como se aprecia en la figura 3.1b.

Además de las representaciones básicas mencionadas, existen características de los vínculos que también tienen una representación grafica: el grado y la conectividad de los vínculos.

Grado de los vínculos.

Se le conoce como grado de un relación al número de entidades participantes involucradas en la asociación.

Una relación de **grado uno** o **unaria** es aquella en la que se conecta una entidad consigo misma. Por ejemplo, la relación *Dirige* de la figura 3.2 conecta los directores con sus subordinados, ambos representados por la entidad *Empleado*. A este tipo de relación se le conoce tambien como relación recursiva o anillo

Una relación de **grado dos** o **binaria**, es una interrelación entre dos entidades. Por ejemplo, una relación *Vive_en* de la figura 3.3 entre *Persona* y *Ciudad* o mejor descrita

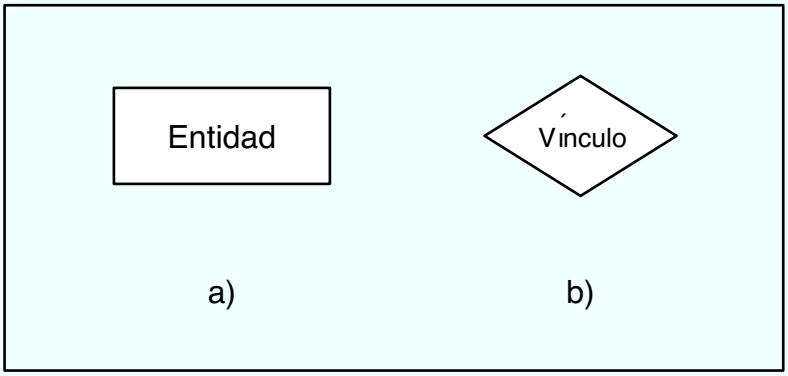


Figura 3.1: Representaciones gráficas de la versión básica del modelo Entidad-Vínculo.

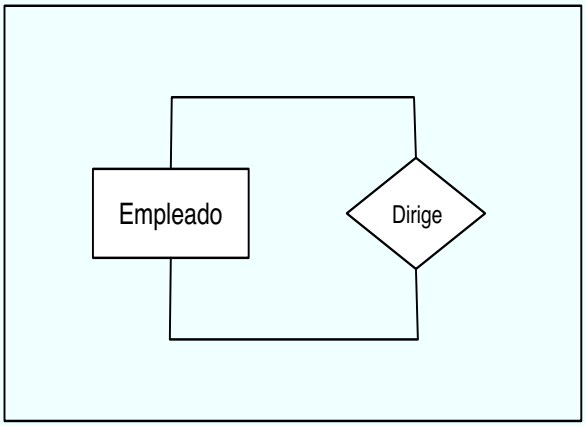


Figura 3.2: La relación grado uno *Dirige*.

con roles como: Una *Persona Vive_en* una *Ciudad*.

Conectividad de los vínculos

El número de las instancias en los *vínculos* en los que puede participar una *Entidad* es conocida como la **conectividad de los vínculos** o como la **razón de cardinalidad**.

Los valores posibles de conectividad de las instancias es (1) o muchos(M/N). Por tanto pueden existir las combinaciones de uno a uno (1:1), de uno a muchos (1:M) y de muchos a muchos (N:M)

Ejemplos de conectividades en relaciones de grado uno:

Una relación *Esposo_de* tiene conectividad (1:1) entre empleados, cuando se describe un vínculo de los esposos actuales entre empleados.

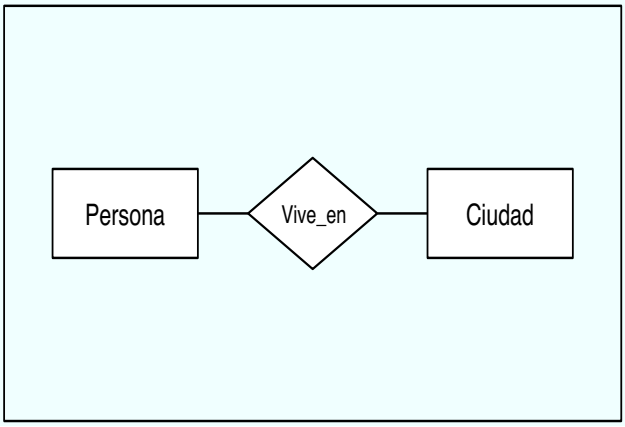


Figura 3.3: La relación grado dos *Vive_en*.

Una relación *Supervisor_de* entre instancias de la entidad *Empleados* tiene conectividad (1:M), cuando se describe que un empleado es subordinado de un sólo supervisor, pero este a su vez puede supervisar a muchos empleados.

Ejemplos de conectividades en relaciones de grado dos:

Una relación *Dirige* entre instancias de las entidades *Empleado* y *Departamento* tiene conectividad (1:1), cuando se describe que un empleado puede dirigir un sólo Departamento y que un Departamento puede ser dirigido por un Empleado.

Una relación *Trabaja_en* entre las instancias de las entidades *Empleado* y *Proyecto* tiene conectividad (1:M), cuando se describe que un empleado trabaja en un Proyecto, y que en ese proyecto trabajan muchos empleados.

Una relación *Cursa* entre instancias de las entidades *Alumno* y *Materia* tiene conectividad (N:M), cuando se describe que un alumno puede cursar varias materias y una materia puede ser cursada por varios alumnos.

3.3.2. Extenciones al modelo entidad-vínculo.

El modelo original entidad-vínculo ha evolucionando con el paso del tiempo y diferentes autores incluso el mismo Chen lo han enriquecido.

Han sido varias propuestas las que han surgido, sin embargo no se cuenta con una extensión estándar, en esta sección mostraremos los elementos de mayor importancia que han sido propuestos [14].

Interrelaciones de Grado Tres

Una relación de **grado tres** o **ternaria**, es una interrelación entre tres entidades. Por ejemplo una relación *Tiene* de la figura 3.4 entre *Empleado*, *Vivienda* y *ciudad* o mejor descrita con roles como: Un *empleado Tiene* una *Vivienda* en una*Ciudad*.

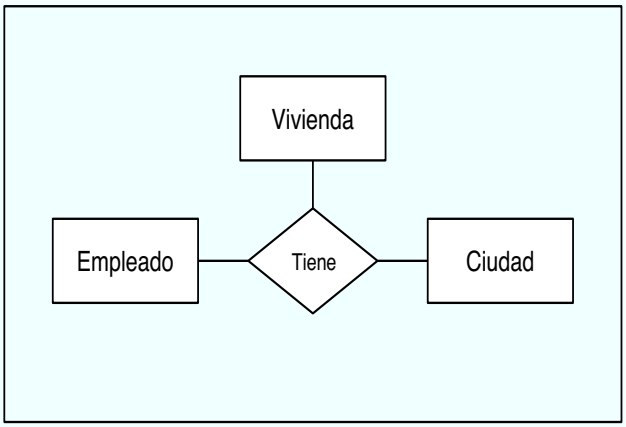


Figura 3.4: La relación grado dos *Vive_en*.

Al adicionar este tipo de interrelaciones tuvo que adicionarse en consecuencia conectividades tales como:

- De uno a uno a uno (1:1:1)
- De uno a muchos a uno (1:M:1)
- De uno a muchos a muchos(1:M:N)
- De muchos a muchos a muchos(M:N:M)

Una relación de grado tres, *Tiene* entre *Empleado*, *Vivienda* y *ciudad* es de conectividad (1:1:1) cuando se describe los roles de:

Un *empleado Tiene* una *Vivienda* en una *Ciudad*.

Un *empleado* en una *Ciudad Tiene* una *Vivienda*.

Una *Vivienda* en una *Ciudad Tiene* un *Empleado* como dueño.

Una relación de grado tres *Asignado_a* entre *Empleado*, *Proyecto* y *Actividad* es de conectividad (1:M:1) cuando se describen los roles:

Una *Actividad* es *Asignada_a* un *Empleado* participando en un *Proyecto*.

Un *Proyecto* es *Asignado_a* un *Empleado* efectuando una *Actividad*.

Una *Actividad Asignada_a* un *Empleado* puede desempeñarla en más de un *Proyecto*.

Una relación de grado tres *Ofrece* entre *Curso*, *Semestre* y *Profesor* es de conectividad (M:N:1) cuando se describe los roles:

Un *Curso* se ofrece en varios *semestres* por un sólo *Profesor*.

Un *Curso* es impartido por un *Profesor* ofrecido en un *Semestre*.

En un *Semestre* se ofrecen *Cursos* impartido por un *Profesor*.

Una relación de grado tres *Suministra* entre *Proveedor*, *Componente* y *Proyecto* es de conectividad (M:N:M) cuando se describe el rol:

Mas de un *Proveedor Suministra* más de un *Componente* en más de un *Proyecto*.

Interrelaciones de Jerarquías: Generalizaciones y Especializaciones.

Los primeros modelados de bases de datos, orientados hacia problemas específicos, fueron resueltos con el modelado EV propuesto por Chen. El advenimiento del modelado de la empresa del sistema integral, dejó al modelo EV sin elementos para plantear las jerarquias entre entidades.

La práctica de muchos diseñadores en modelados de empresas descubrieron interrelaciones que suponían tener jerarquias entre entidades, y propusieron plantear esas jeraquias extendiendo el modelo con generalizaciones y especializaciones.

Una entidad E es una **generalización** de un grupo de entidades $E1$, $E2$, ..., E_n, si cada objeto de las clases $E1$, $E2$, ..., E_n es también un objeto de la clase E. Es decir, E queda conformada con los atributos semejantes de las entidades $E1$, $E2$, ..., E_n. De estas definiciones podemos afirmar que para crear una generalización E debieron primeramente existir las entidades $E1$, $E2$, ..., E_n. Consideremos el esquema de la figura 3.5a, donde se expresa que las entidades *Varon* y Hembra exisistieron cada una con sus atributos, y crean una clase generalizada llamada *Persona*. Esta figura muestra la concepción inicial de esta generalización. La propiedad de la herencia establece que los atributos NOMBRE y DIRECCION son también atributos de *Varon* y *Hembra*; luego, pueden ser eliminados de las entidades originales, simplificando el esquema como se muestra en la figura 3.5b. Lo opuesto a la generalización es la **especialización**, por tanto se dice que una entidad $E1$. $E2$ o E_n es una especialización de E si cada objeto de $E1$. $E2$ y E_n es también un objeto de la entidad E. Por tanto, podemos afirmar que para crear $E1$. $E2$ y E_n debió de exisitir primeramente la entidad E. Y que todos los atributos de E son heredados a las entidades

$E1$. $E2$ y E_n.

Cada entidad puede participar en multiples generalizaciones, posiblemente en el papel de entidad genérica con respecto a una generalización y en el papel de entidad subconjunto con respecto a otra generalización.

La diferencia fundamental entre la especialización y la generalización es que la primera especifica conjuntos traslapables de entidades de subtipo, mientras que la segunda especifica conjuntos mutuamente excluyentes de entidades de subtipo.

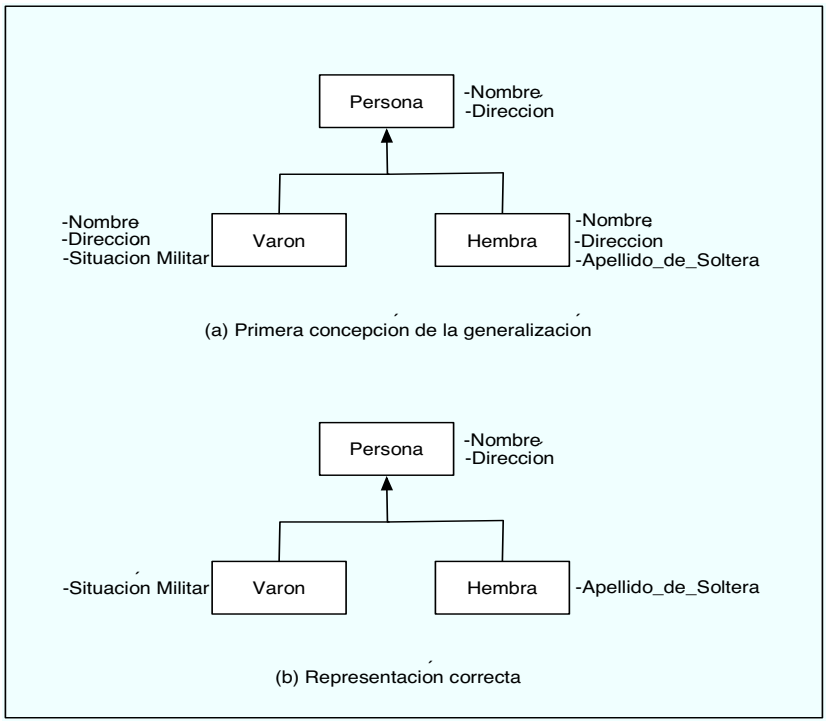


Figura 3.5: Jerarquias de generalización.

3.3.3. Mecanismos de abstracción utilizados en el modelo endidad-vínculo

Es de interés mostrar cómo algunas de las abstracciones mostradas en una sección anteriro son asimiladas por los conceptos del modelo entidad-vínculo [3].

Abstracción de clasificación.

Los conceptos básicos del modelo entidad-vínculo se desarrollan como aplicaciones de abstracción de clasificación:

1. Entidad. Es una clase de objetos del mundo real con propiedades comunes.

2. Interrelación. Es una clase de hechos atómicos (elementales) que relacionana dos o más entidades.

3. Atributo. Es una clase de valores que representan propiedades atómicas de las entidades o interrelaciones.

Abstracción de agregación.

Dos tipos de agregaciones caracterizan el modelo entidad-vínculo.

1. Una entidad es una agregación de atributos.

2. Una interrelación es una agregación de entidades y atributos.

Abstracción de generalización.

Las abstracción de generalización la encaran las gerarquias de generalización y especialización, usualmente sólo se aplica a entidades, auque algunas ampliaciones del mode entidad-vínculo aplican la abstracción de generalización también a interrelaciones o atributos.

3.3.4. Familias de modelos con Entidad Vínculo Extentido (EVE).

Como hemos visto en la sección anterior se han hecho extensiones al modelo original propuesto por Chen y no se ha aprobado un modelo entidad vínculo estandar, en vez de ello, los modelos se han venido agrupando en familias. De las familias de modelos EVE, los modelos mas utilizados son los de Chen, de Teorey, de Everest y los de IDEF1X(desarrollado por la Fuerza Aérea de los EU). Las principales diferencias en las metodologías consisten en que algunas extienden del modelo Entidad Vínculo, el grado de las relaciones a tres mientras otros sólo permiten dos. Pese a diferir en su notación, los objetos visuales un diagrama de un modelo pueden encontrar su equivalente en los objetos visuales de otro [4].

En los diagramas EVE de Chen se tienen las siguientes representaciones (fig 3.6):

Entidades: Mediante cajas o rectángulos.
Interrelaciones: Admite sólo interrelaciones unaria y binarias representadas con rombos.
Conectividad de Interrelaciones: Se representan con etiquetas uno (1) o muchos (M) en las líneas de conexión que van de las relaciones a las entidades.
Especialización y Generalización: Se representan con un triangulo del cual se conecta por arriba por la entidad subtipo y por debajo con las entidades subtipos. El grosor de la línea de conexión proporciona la diferencia entre especialización y generalización. Las líneas con mayo grosor, representan a la generalización y las de menor grosor a la especialización.

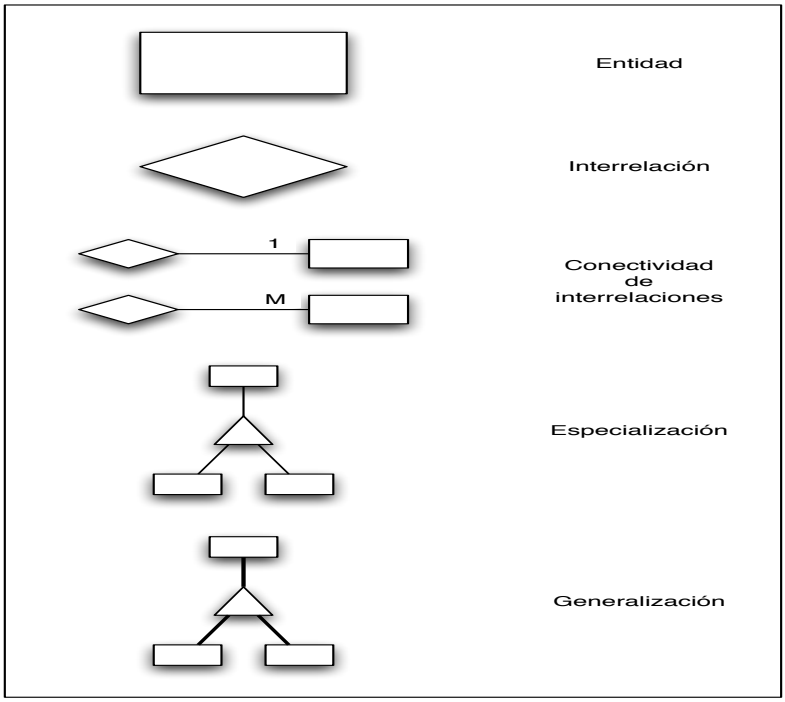


Figura 3.6: Representaciones graficas de Chen.

Teorey y Reiner utilizan las representaciónes descritas a continuación (fig 3.7):

Entidades: Mediante cajas o rectángulos.
Interrelaciones: Con rombos para las interrelaciones unaria y binaria, y triangulos para las interrelaciones ternarias.
Conectividad de Interrelaciones: Se expresa visualmente aristas del polígono de la relación. La conectividad muchos se representa con una arista sombreada y la conectividad de uno, con una arista sin sombrear.
Especialización: Una flecha con su punta hacia arriba.
Generalización: Una flecha señalando hacia arriba pero con base. Las conexiones de supertipo salen del apunta de la flecha, y las de subtipo llegan por abajo.

En Everes se representan como se describe a continuación (fig 3.8):

Entidades: Mediante cajas o rectángulos.
Interrelaciones y Conectividad: No se representan con un objeto visual o ícono como en las metodologías descritas anteriormente. Las entidades se conectan con líneas y en ambos extremos de la conexión se establece el grado mínimo y máximo de conectividad de la relación. Con una línea vertical se representan la conectividad de uno y con tres líneas que parten de un extremo de la relación llegan al extremo de la entidad, semejaqndo las líneas

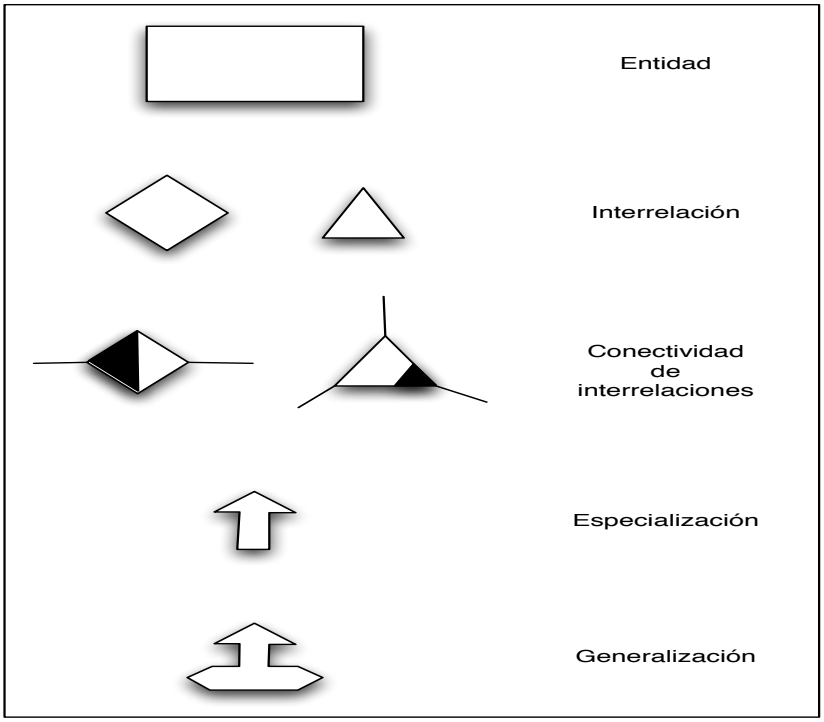


Figura 3.7: Representaciones graficas de Teorey y Reiner.

unas patas de gallo. Esta metodología excluye a las relaciones ternarias.
Especialización y Generalización: Se representan con un triángulo diminuto colocado debajo de la entidad supertipo. De la base del triángulo diminuto, parte una línea de conexión hacia los subtipos.

En IDEF1X se diagraman (fig 3.9):

Entidades: Mediante cajas o rectángulos.
Interrelaciones: No se cuenta con un símbolo propio.
Conectividad de Interrelaciones: Las líneas de conexión entre entidades colocan un círculo en los extremos de la línea que une a las entidades. La conectividad uno, se representa con un círculo sin rellenar y la de muchos con un círculo relleno.
Especialización y Generalización: Se representan con un círculo sin rellenar con dos íneas horizontales como base, de la parte de arriba del círculo parte la conexión al supertipo y de la horizontal parte la línea hacia el subtipo.

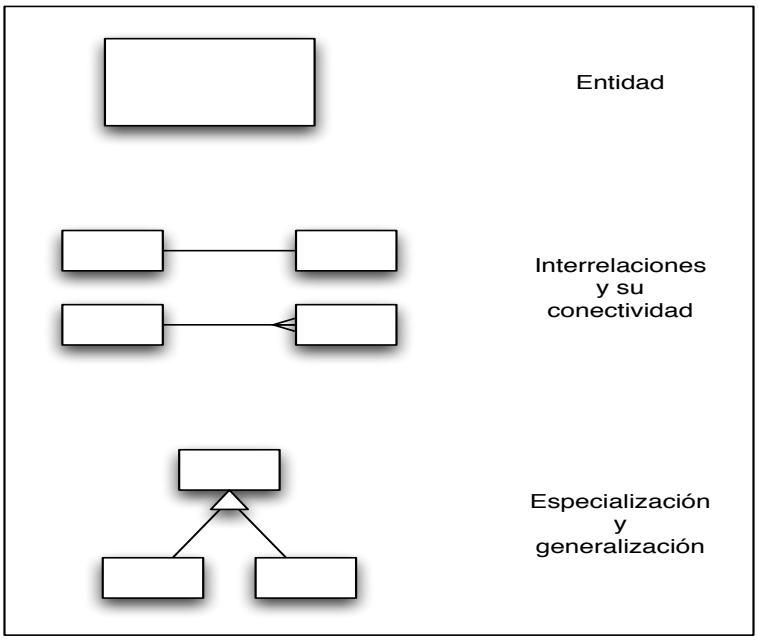


Figura 3.8: Representaciones graficas de Everest.

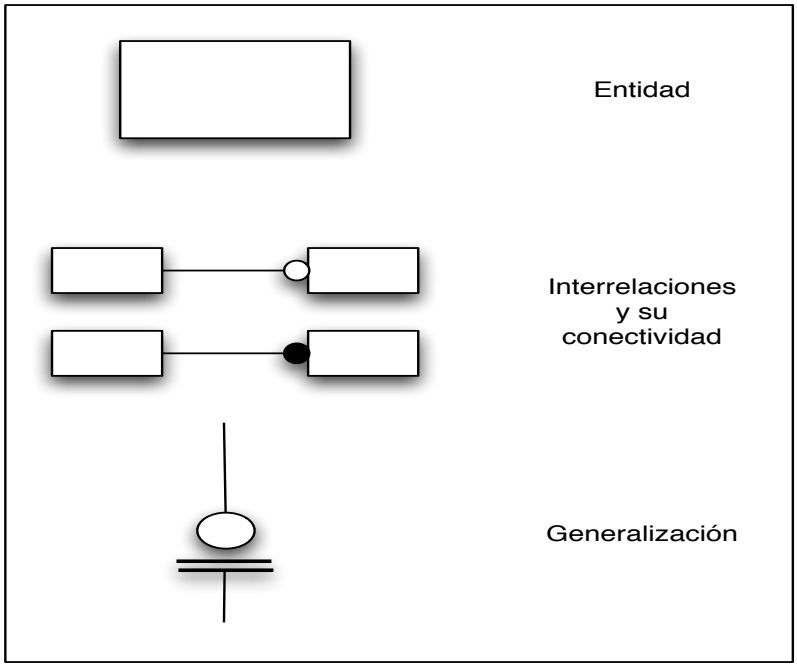


Figura 3.9: Representaciones graficas de IDEF1X.

3.3.5. Ventajas del modelo entidad vínculo extendido (EVE).

La meta de modelar base de datos es diseñar una base de datos eficiente y apropiada. Algunos de los criterios más importantes son rendimiento, integridad, comprensibilidad y expandibilidad. La extensión del modelo EV ha sido eficaz en reunir esas metas. Este enfoque demuestra que una teoría fuerte puede desarrollar y aplicarse a problemas prácticos. La historia del DBMS demuestra que una débil teoría básica conduce a una mala tecnología que es difícil de aplicar. El modelo extendido presentado tiene las siguientes ventajas:

1. Cuenta con una fuerte teoría básica.

   - El modelo es basado en una lógica multitipos que es equivalente a la lógica de predicados de primer orden. Consecuentemente, puede ser usada en la teoría relacional y en resultados obtenidos de matemáticas discretas.
   - El modelo incluye el completo espacio para modelar la información. La estructura semántica estática, operaciones genéricas específicadas por el usuario, interfaz de usuarios y el comportamiento de una aplicación pueden estar describiendo todo el modelo.
   - La teoría es simple y clara. Secuencias, subconjuntos y conjunto potencias de objetos pueden ser modelados directamente.
   - Podemos obtener esquemas normalizados como en la teoría clásica.

2. El modelado es más natural y puede ser aplicado en una manera simple.
   El modelo solamente necesita información escencial para ser expresada. Aqui no hay requerimientos de repetición y redundancia o la introducción de tipos dependientes de implementación artificial.

   - El modelo soporta traslado directo a esquemas relaciones, de red y jerárquicos, la decisión del diseño puede ser usada inmediatamente para obtener esquemas en forma normal. La teoría de traslación puede ser usada para soportar multimodelos y multisistemas, y presenta una solución práctica para la interoperabilidad del sistema.
   - El algebra EVE es usada para la definición de consultas. Las consultas pueden ser automáticamente generadas correspondiendo a los modelos relacional, de red o jerárquico.

3. La teoría es aplicable a la necesidad de la práctica:

   - Basado en la teoría, un multimodelo, diseña una metodología robusta al incorporar los enfoques usados en el modelo orientado a objetos, programación modular y programación avanzada.

- En teoría, el modelo y el desarrollo de diseño de sistemas puede ser usado para mantener una variedad de base de datos heterógeneas, posiblemente bajo la ilusión de un simple modelo de datos. Esto directamente soporta migración de un sistema a otro.

4. El resultado del diseño es mucho más simple que con otros enfoques.

   - Los esquemas obtenidos son entre tres y cinco veces más simples que los que se obtienen utilizando otros modelos. El ejemplo es simplificado en cuatro pliegues y puede ser puesto en una página o en una pantalla. En otros ejemplos, la simplificación hace posible encontrar un modelo. Utilizando este enfoque de modelación fue modelada una aplicación de conteo en un aeropuerto utilizando menos de 40 tipos de entidades y menos de 120 tipos de relaciones. La solución original incluía más de 150 tipos de entidades y más de 400 tipos de relaciones, por lo que no fue utilizada por los usuarios por su complejidad y su no transparencia.
   - La simplificación también lleva a un mejor entendimiento de la aplicación y y hace más fácil preservar la normalización.

5. El modelo es fácil de entender, simple y comprensible.

   - El modelo puede ser usado como una herramienta de disenño de base de datos.
   - El modelo soporta interfaces de usuario de alto nivel, expandibilidad en base de datos y tiene un conjunto de metodologías especiales para su utilización.
   - Basada la información en las etapas del diseño, las abstracciones en el esquema pueden ser usadas para visualizar la parte del esquema generalmente bajo consideración sin perder la información en el contexto de esta parte. La representación modular simplifica el diseño de tareas de esquemas a gran escala.

3.3.6. EVEMac

EVEMac es el nombre de la herramienta que desarrollamos para asistir el diseño de bases de datos empleando el modelo Entidad Vínculo Extendido.

Esta herramienta presenta un marco de trabajo que permite realizar diseños de bases de datos en dos niveles: nivel de Meta-Diseño y nivel de diseño. El primer nivel se emplea para diseñar una base de datos que almacenará información del proyecto de software tal como los recursos, roles, tareas, etc., esta permitirá llevar un control y administración del proyecto. En el segundo nivel se diseña la base de datos que corresponde a la abstracción del negocio que se esta modelando.

Ambos diseños son basados en el método entidad vínculo extendido, por lo que cada uno crea un diagrama EVE independiente bajo un mismo marco de trabajo. Los diagramas se realizan utilizando una símbología simplificada que se implementó después de analizar las simbologías de las familias presentadas en la sección anterior.

Esta simplificación se obtuvo al dar mayor peso a los siguientes aspectos:

1. La simbología para representar vínculos y conectividades de los modelos propuestos por Teorey y Reiner en el mejor de los casos resultan ser un total de 7 símbolos (fig 3.10). Esta cantidad de símbolos puede confundir a los diseñadores no experimentados e incluso retrasar a los experimentados en el proceso de selección de que tipo de vínculo desean diagramar. Por lo anterior omitimos este enfoque.

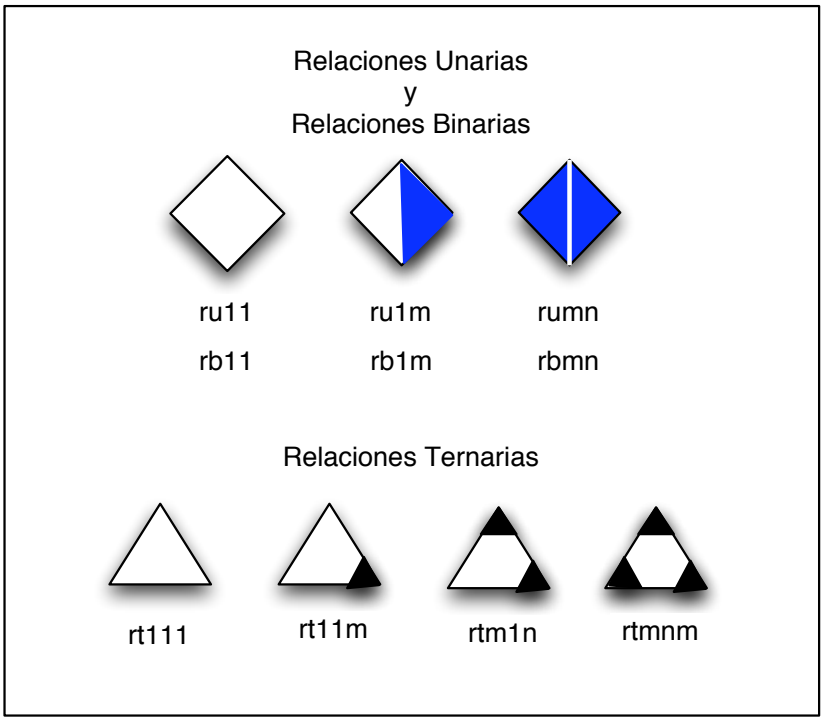


Figura 3.10: Símbolos que represen los vínculos y sus conectividades en la propuesta de Teorey y Reiner.

2. El enfoque de IDEF1X y Everest no representa visualmente los vínculos. En este trabajo descartamos este enfoque por dos razones principales. La primera es que un vínculo es un objeto conceptual de la realidad que se debe representar gráficamente, sobre todo porque los vínculos pueden tener sus propios atributos (es decir su propia identidad). La segunda razón es que deseamos abstraer la mayor cantidad de conceptos de la realidad en nuestros diagramas y representarlos gráficamente, con la finalidad que sean más expresivos y por ende facilmente interpretados.

3. El enfoque propuesto por Chen utiliza rombos para representar las relaciones unarias y binarias. En EVEMac no se aplicó este enfoque ya que en el modelo EVE es

necesario representar las interrelaciones ternarias y usar rombos para estas relaciones crearía confunción a los diseñadores,

Por lo anterior, proponemos utilizar círculos para representar las interrelaciones. El grado de cada relación se determina por el número de entidades con las que asociada . La figura 3.11 muestra la simbología que implementamos en EVEMac. Como podemos observar, es una combinación de la mayoria de las familias de los modelos analizados previamente. Pretendiendo que los diagramas aporten mayor claridad.

En [3] y [11] se mencionan las cualidades que deben de poseer los modelos conceptuales y que fueron consideradas para la elección de esta símbología propuesta.

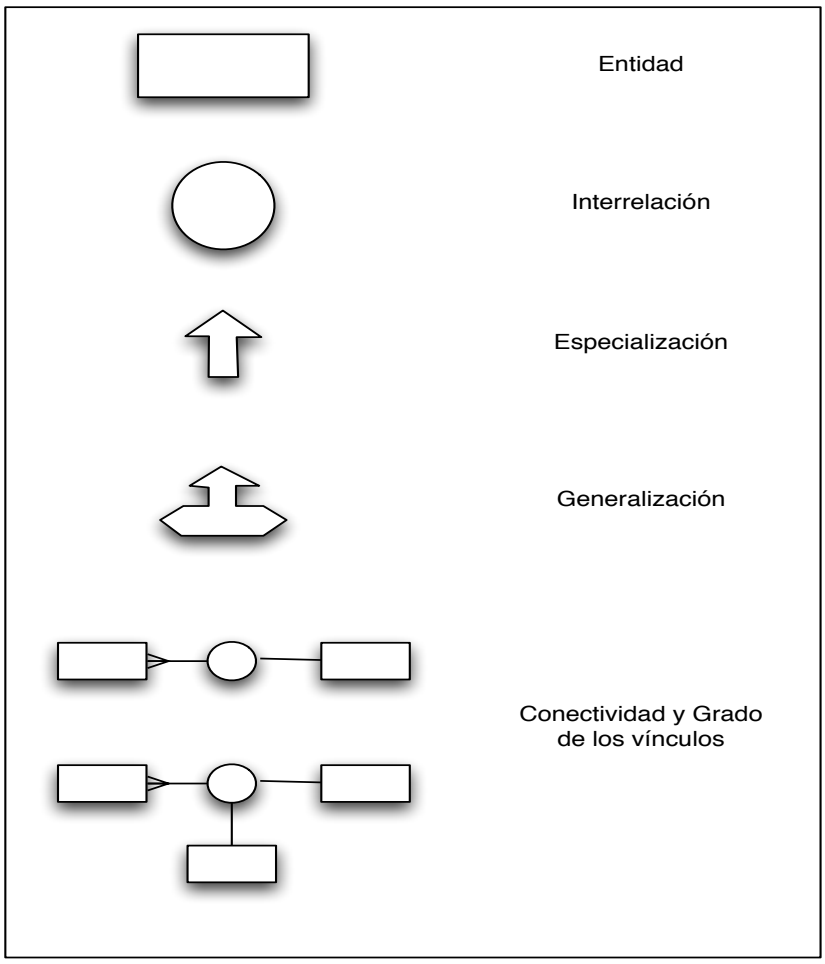


Figura 3.11: Símbolos que represen los vínculos y sus conectividades en la propuesta de Teorey y Reiner.

Entidades: Mediante cajas o rectángulos.
Interrelaciones: Círculos para las relaciones (unarias, binaria, y ternarias.
Grado de las Interrelaciones: El grado de las interrelaciones se obtiene del número de en-

tidades con las que se asocia, *Conectividad de Interrelaciones:* Las entidades se conectan con líneas y en ambos extremos de la conexión se establece el grado mínimo y máximo de conectividad de la relación. Con una línea vertical se representan la conectividad de uno y con tres líneas que parten de un extremo de la relación llegan al extremo de la entidad, semejaqndo las líneas unas patas de gallo. Esta metodología excluye a las relaciones ternarias.
Especialización: Una flecha con su punta hacia arriba.
Generalización: Una flecha señalando hacia arriba pero con base. Las conexiones de supertipo salen del apunta de la flecha, y las de subtipo llegan por abajo.

Los diagramas EVE se realizan bajo una edición dirijida por sintaxis, lo que facilita la obtención de diagramas correctos sintacticamente. En el mismo marco de trabajo se crea el diccionario de atributos en cual se capturan todos los atributos identificados en la fase de análisis de información, mismos que serán asignados a las entidades y relaciones (vínculos) que lo requieran. Una vez que se han asignado nombres a cada *entidad* y *relación* así como sus atributos llaves y descriptores se tiene un diagrama EVE completo o lo que comunmente se conoce por el esquema conceptual de una base de datos. Los esquemas conceptuales obtenidos con esta herramienta pueden ser objeto de traducción a un esquemá lógico y aplicar además un proceso de normalización para culminar con la creación de un esquema físico de las bases de datos.

Capítulo 4

EVEMac: Herramienta para diseño de bases de datos

Hoy en día las herramientas de diseño de bases de datos se enfocan en el diseño de estructuras que almacenan información requerida y necesaria de los negocios para garantizar el eficiente comportamiento de un sistema de información, Sin embargo, pragmáticamente se ha observado que todo proyecto de software debe de administrarse para asegurar que se cumpla con el alcance, tiempo, costo y calidad necesarios. Normalmente esta actividad de administración de proyectos se apoya de herramientas extras de software para llevarse a cabo. Por lo tanto planteamos una aplicación que trabaje en dos niveles de diseño: el diseño conceptual tradicional de bases de datos más un nivel adicional llamado Meta-Diseño, donde se modela una estructura general del proyecto definiendo actores, roles y recursos. Ambos niveles de diseño se enfrentan con la misma metodología y ambiente visual. A esta herramienta la hemos llamado EVEMac (**E**ntidad **V**inculo **E**xtendido para **Mac**intosh).

En este capítulo presentamos el diseño orientado a objetos de la herramienta EVEMac y su implementación, comenzando con la arquitectura general de sistema y detallando primeramente el diccionario de atributos y posteriormente el editor de diagramas EVE al cual nos referiremos de ahora en adelante como el "digramador".

La herramienta EVE Mac se desarrolló tomando en cuenta las técnicas de programación visual y programación orientada a objetos [6] para obtener una aplicación que facilite el diseño de bases de datos con un lenguaje visual basado en diagramas.

Esta aplicación contiene varios módulos, ellos trabajan de manera integral para convertir un esquema conceptual basado en un diagrama entidad-vículo a un esquema físico de una base de datos; este esquema físico se basa en un esquema de relación en tercera forma normal (3FN).

El primer módulo corresponde al editor del diccionario de atributos. Este permite la captura de todos los atributos de las diferentes entidades que se representan en un diagrama EVE. La captura de los atributos puede realizarse previamente a la edición del diagrama o posteriormente cuando se requiera enlazar cada entidad o relación diagramada con sus respectivos atributos que las conforman.

El segundo módulo es editor de diagramas EVE. Este permite realizar las siguientes

actividades:

- Crear y modificar diagramas empleando la simbología propuesta por nuestra herramienta.
- Dirigir la edición de los diagramas EVE por sintáxis
- Enlazar los atributos capturados en el diccionario de atributos con las entidades y relaciones correspondientes.

El traductor es el tercer módulo y se encarga de dos tareas. Primero debe concluir la verificación de sintaxis de los diagramas, es decir, la sintaxis que no es posible verificar en el momento de la construcción del diagrama. Una vez que valida que el diagrama es correcto sintácticamente, convierte los diagramas a sus equivalentes en esquemas relacionales siguiendo reglas de transformación para cada elemento diagramado.

El cuarto módulo es el normalizador. Dentro de este modulo se leen los esquemas traducidos para obtener un esquema normalizado al aplicar el algoritmo de Bernstein.

El último modulo es el correspondiente al constructor del scrip. En este modulo se toma el esquema normalizado y se traduce a su equivalente en el lenguaje de definición de datos de PostgreSQL.

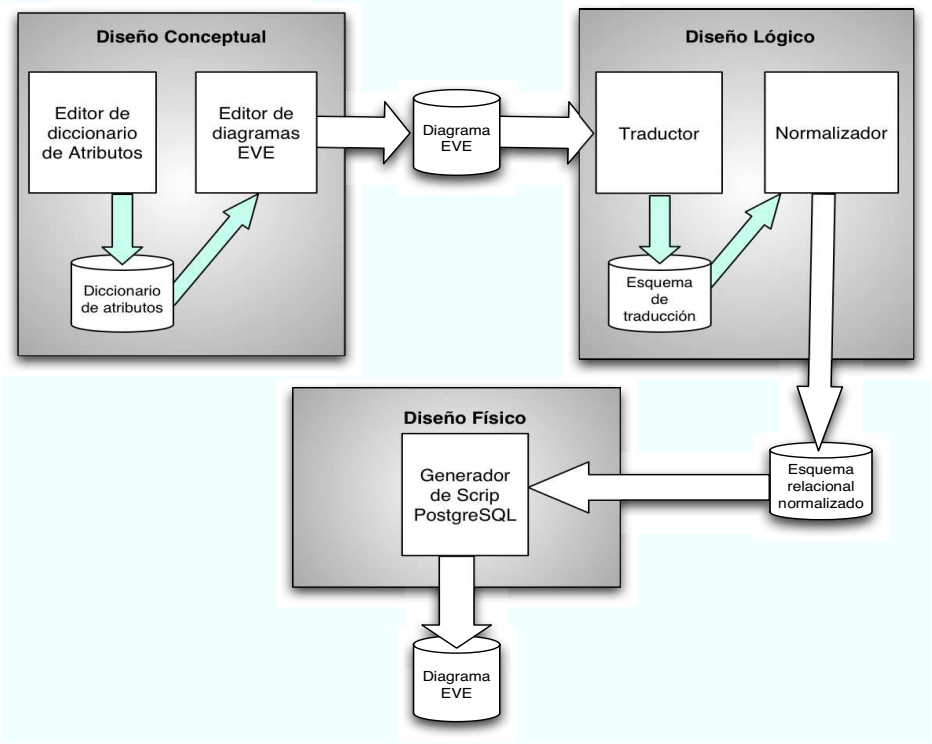


Figura 4.1: Arquitectura general de la aplicación EVE Mac.

4.0.7. EVEMac como aplicación basada en documentos.

El diseño de EVEMac fue realizado con la arquitectura de aplicaciones basadas en documentos proporcionada por cocoa. Esta arquitectura permite tener abiertos más de un documento[1] (en nuestro caso, diagramas EVE), lo que facilita poder visualizar varios diagramas al mismo tiempo. Además, esta arquitectura sigue los principios del patrón Modelo-Vista-Controlador (MVC) en el cual nos basamos para el diseño de nuestra herramienta. Este patrón propone dividir la funcionalidad de la aplicación en tres clases: una clase que modela los datos (el modelo), otra clase más que despliega los datos (la vista) y por último una clase que sirve como mediadora entre el modelo y la vista (el controlador). La arquitectura de multiples documentos en cocoa adopta el patrón MVC con esas tres clases, sin embargo, sugiere dividir el control en dos clases controladoras: La clase modelo-controlador y la clase vista-controlador. La primera controla y administra el modelo de datos y la otra clase cumple con el rol tradiciona; del controlar la interfaz de usuario [5] [15].

Los proyectos generados con la arquitectura basada en documentos, proporcionan un esquema para desarrollo como el que puede observarse en la figura 4.2 . En donde los archivos .nib, son objetos que nos permiten diseñar la interfaz gráfica de usuario. Se crean dos archivos por defecto al iniciarse un proyecto de este tipo. Uno que contiene el menú al cual puede se le pueden incluir mas elementos según las necesidades de los desarrolladores y otro que aparece vació y permite construir la interfaz gráfica de usuario deseada.

NSDocument.
Es una clase model-control, su principal trabajo es poseer y administrar objetos que conforman un documento y proveer la forma de almacenar esos objetos en archivos para recuperarlos posteriormente. Por tanto toda aplicación basada en documentos debe tener una subclase de NSDocument y debera redefinir aquellos métodos primitivos necesesarios para la funcionalidad de su aplicación. Esos métodos realizan actividades tales como las siguientes:

- Leer datos desde un archivo.
- Escribir datos en archivos.
- Guardar documentos.
- Revertir cambios en documentos.
- Imprimir documentos.

NSWindowController.
Es una clase vista-control, su principal trabajo es poseer y administrar objetos que son

[1] Contenedor repetible para un cuerpo único de información identificado por el mismo nombre bajo el cual es almacenado

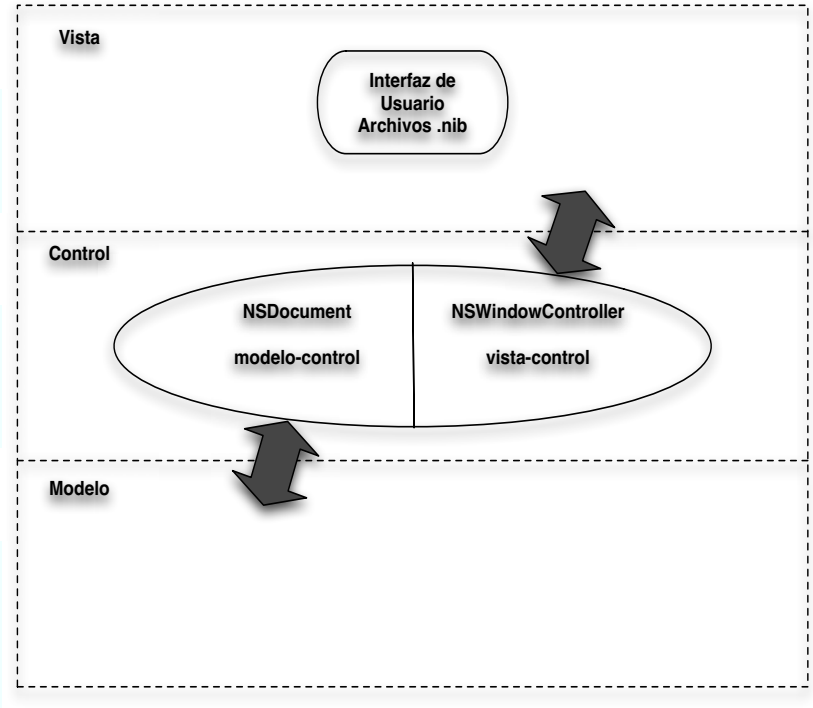


Figura 4.2: Elementos del Patrón MVC en la arquitectura de aplicaciones basada en documentos.

usados para desplegar y editar un documento, es decir, los encargados de realizar la representación visual del modelo de datos. Las aplicaciones basadas en documentos tienen subclases de *NSWindowControler* para controlar la intefaz gráfica y por tanto son los propietarios de los archivos .nib.
En esta arquitectura que proporciona cocoa tenemos el esqueleto inicial de una aplicación para complementar en patrón MVC hace falta definir las clases necesarias del modelo de datos, asi como diseñar la interfaz de usuario sobre los archivos .nib.

Las nuevas clases creadas para conformar la funcionalidad de nuestra aplicación son presentadas en el diagrama de clases de la figura 4.3, donde presentamos en la parte superior las clases que pertenecen al *framework* Application Kit de cocoa c-objetivo, de las cuales en necesario heredar hacia las clases nuevas que se han integrado. Las clases creadas para EVE Mac se presentan en la parte inferior del diagrama. Aqui es posible distinguir las clases que pertenecen a cada elemento del patrón de diseño: Modelo-Vista-Controlador.

EvexDocument.
Esta clase hereda de NSDocument por tanto es la clase que juega el rol del objeto Modelo-Controlador, y se encargará de administrar las actividades relacionadas con los documentos creados.

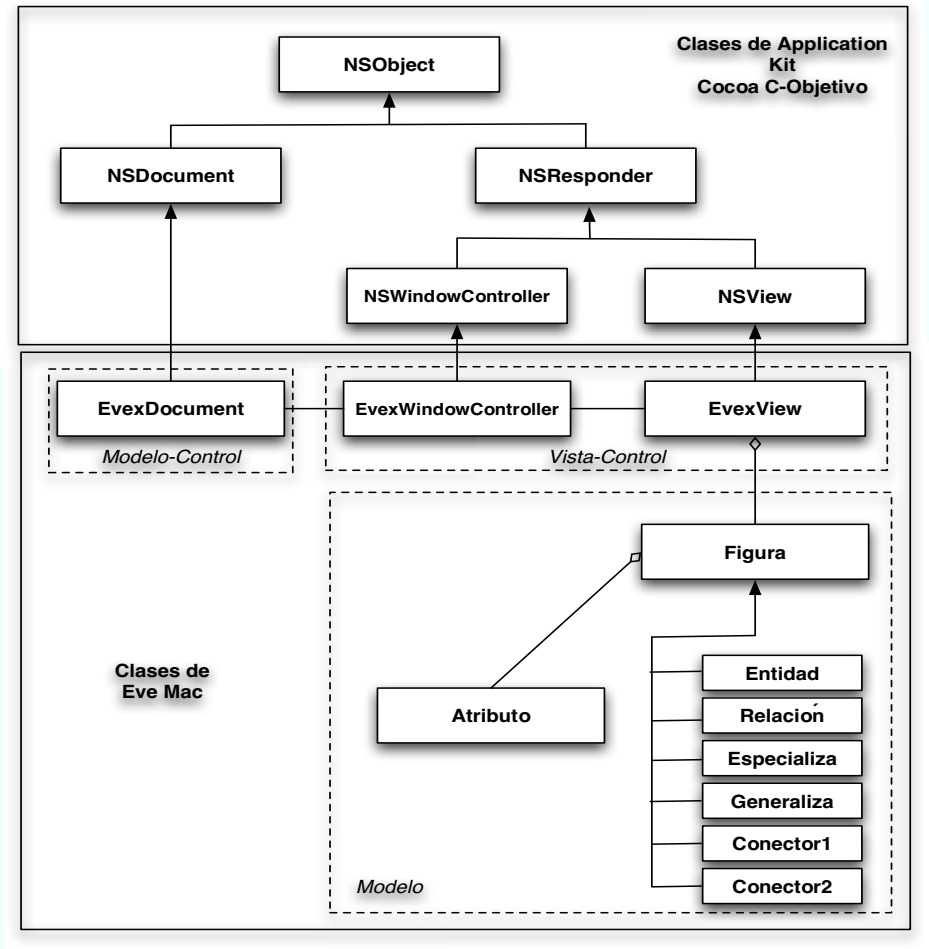


Figura 4.3: Diagrama de clases de EVE Mac.

EvexWindowController.
Esta clase hereda de NSWindowController por tanto es la clase que cumple el rol de Vista-Controlador, y se encarga de recibir y controlar los eventos de usuario que ocurren en la interfaz gráfica, especialmente los eventos que se generan en la paleta.

EvexView.
Es una clase que hereda de NSView y es la clase que nos permite tener y controlar la interfaz gráfica donde se pueden elaborar los diagramas EVE. Dentro de esta clase que trabaja conjuntamente con la clase *EvexWindowController* se crean los objetos del modelo que son representados gráficamente, dichos objetos son creados con clases que heredan de una clase base llamada *Figura*.

4.1. Diseño e implementación de los módulos: Editor y Diagramador

En la sección anterior presentamos la arquitectura general de nuestra aplicación así como el diagrama general de clases, ahora nos concentraremos en el diseño e implementación de los modulos: Editor del diccionario de atributos y el diagramador.

4.1.1. Editor del diccionario de atributos

El editor de diccionario de atributos es el módulo que permite crear el diccionario de atributos que se relaciona con un diagrama EVE específico. Todos los atributos que se logren identificar en el proceso de recolección y análisis de información se capturan sin saber necesariamente en esta etapa a que entidades o relaciones pertenecerán.

Diseño

El diseño de este módulo se realizó considerando también el patrón MVC, ya que se cuenta con una interfaz gráfica (vista) donde se captura y se muestra la información (modelo) acerca de los atributos que conforman un diccionario de atributos.

En la figura 4.4 se puede observar la representacion del diseño de alto nivel de este módulo.

Objetos Modelo

El modelo de datos del diccionario de atributos está conformado por la información que nos interesa administrar y representar en cierto momento dentro de nuestra interfaz de usuario. Esta información esta relacionada con las propiedades de los atributos, por tanto modelamos una clase llamada “atributo” que contiene todos los datos de interes en variables de instancia y las operaciones que pueden realizadar.

Objetos Vista

La vista de un diccionario de atributos es un objeto gráfico en cocoa que corresponde a un archivo .nib, en el cual se utilizaron diferentes objetos de la paleta para diseñar la interfaz del usuario. Esta intefaz es una pantalla que funcionalmente se divide en tres partes principales. La primera corresponde al área en donde se puede capturar la información. Otra más corresponde a los botones que permiten realizar las acciones de adiccionar y borrar atributos, as como de limpiar la pantalla de captura. Y por último se encuentra el área donde se muestra una lista de todos los atributos que han sido capturados.

Objetos Control

El objeto Control del diccionario de datos es DicciControl y es el responsable de mediar el flujo de datos entre la interfaz de usuario y el modelo que encapsula los datos del objeto Atributo.

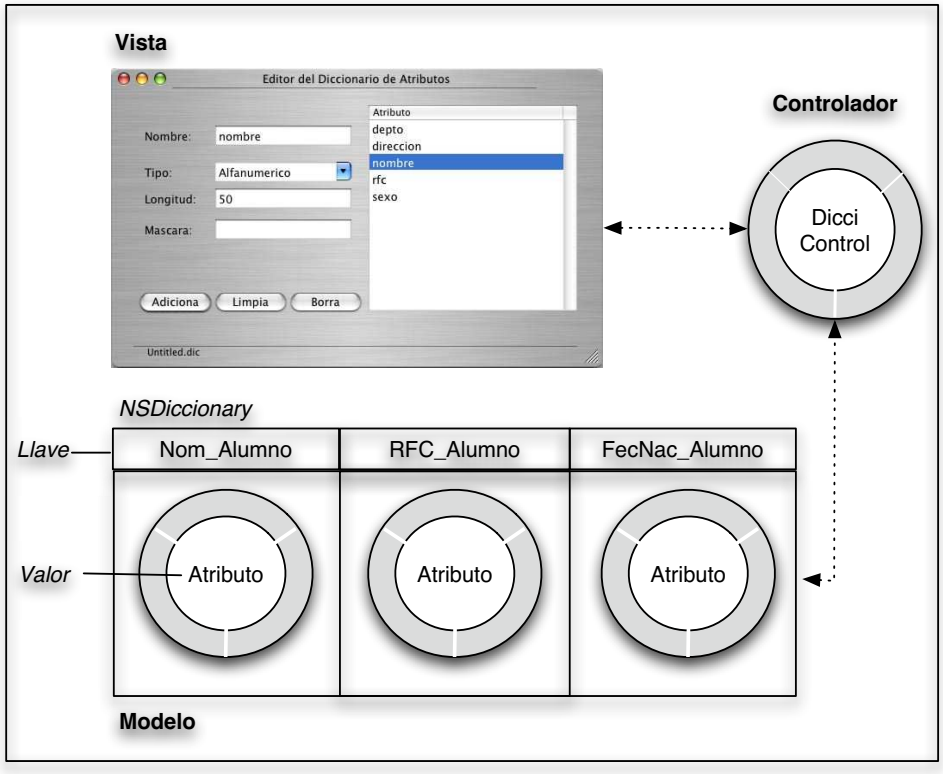


Figura 4.4: Diseño general del editor del diccionario de atributos.

Una de las tareas a cargo de DicciControl es proveer la manera de almacenar en memoria todos los objetos *atributo* creados y permitir recuperarlos posteriormente en caso de requerirlo. Esta tarea se apoya de un objeto heredado de *NSDictionary* que funje como un contenedor que permite almacenar y recuperar objetos a través de una asociación de un valor llave. Esta clase también desempeña el rol de transferir datos de los objetos modelo a los campos de texto de la interfaz, y cuando se insertan nuevos datos o se modifican los existentes realiza la operación contraria, es decir, transferir el contenido de los campos de texto de la interfaz hacia los objetos modelo. Esta clase además coordina los datos desplegados en la lista de atributos disponibles (capturados) para realizar las acciones debidas al realizar una selección o al presionar el botón de adicionar o eliminar. Los mecanismos de esta última actividad requieren de un arreglo (Objeto NSArray) para ordenar alfabéticamente los atributos y por tanto puedan ser visibles en la pantalla considerando este ordenamiento.

Implementación

Ahora que hemos hablado acerca del diseño del editor del diccionario de datos comenzaremos a describir con mayor detalle la implementación explicando las clases que lo

componen.

La clase *Atributo*

Esta clase es la encargada de generar instancias de los atributos que perteneceran a un determinado diccionario. (Figura 4.5)

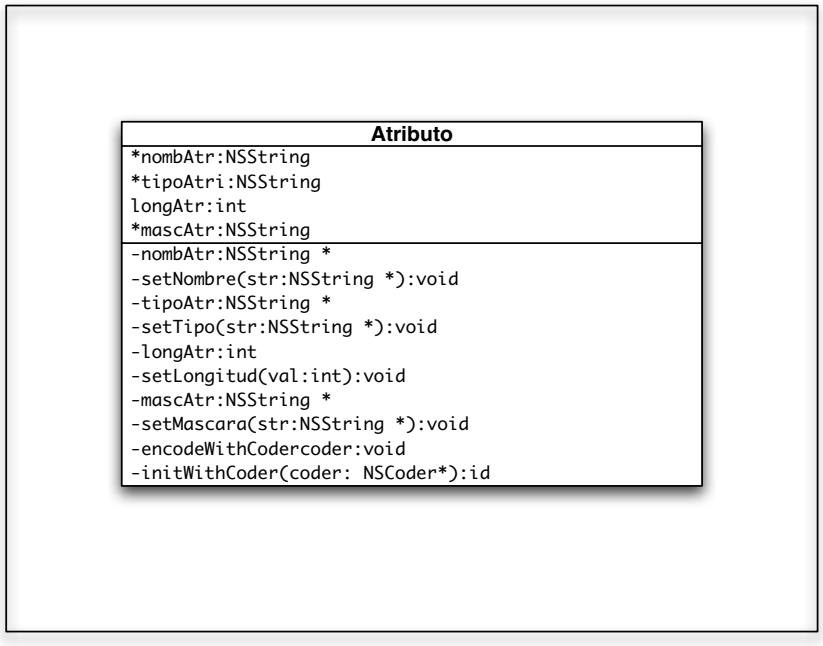


Figura 4.5: Clase Atributo.

Las variables de instancia de esta clase almacenan las propiedades que tiene un atributo como son: nombre del atributo, tipo de dato, longitud y mascara (formato con que se desea presentar la información).

Los métodos de instancia más importantes de esta clase son los siguientes: *setNombre, setTipo, setLongitud y setMascara* se encargan de asignar nuevos valores a las variables de instancia a los cuales hacen referencia en su nombre. Son invocados cuando es necesario actualizar el valor de una variable de instancia de un objeto *Atributo*. Los métodos *nombAtr, tipoAtr, longAtr y mascAtr* retornan el valor almacenado en las variables de instancia a la que hacen referencia en su nombre. Son invocados cuando es necesario conocer el valor que almacena una variable de instancia de un objeto *atributo*.

encodeWhitCode Este método es implementado para poder codificar objetos que serán almacenados en memoria secundaria. Cuando este método es invocado codifica todos los datos de un atributo preparándolo para su almacenamiento.

initWithCoder Este método es implementado para poder decodificar objetos que han sido almacenados en memoria secundaria. Cuando este método es invocado decodifica los datos de un atributo preparándolo para que pueda ser utilizado en la aplicación.

La clase *DicciControl*

Esta clase es la encargada de crear nuevas instancias de atributos, de almacenarlos en un contenedor (que después permitirá guardarlos en disco), de eliminarlos y de modificarlos. (Figura 4.6)

Esta clase recibe eventos desde la interfaz de usuario y puede obtener los valores que se capturan desde ella. Asi mismo actualiza la interfaz de usuario con los cambios en la lista de atributos capturados.

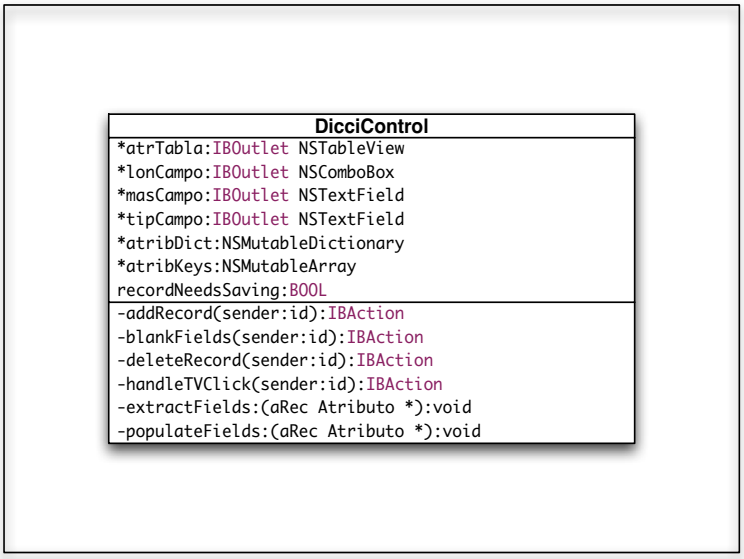


Figura 4.6: Clase DicciControl.

Las variables en las que su definición comienza con la palabra *IBOutlet* son variables de instancia que permiten conectarnos directamente con objetos de la interfaz de usuario y por tanto manipular y desplegar información en ella. La variable **atrTabla* nos permite desplegar la lista con los nombres de todos los objetos *Atributo* adicionados al diccionario. Las varibles restantes son los campos de texto que permiten editar las propiedaes de un objeto *Atributo*: nombre, tipo, longitud y máscara respectivamente.

La variable de instancia **atribDict* es una colección que nos permite almacenar todos los objetos *Atributo* que son adicionados al diccionario, esta variable nos permite almacenar en memoria secundaria la información con sólo adoptar un par de métodos para tal efecto. La variable **atribKeys* es un arreglo en donde se almacenan también objetos *Atributo*, sin embargo su uso es muy diferente ya que se toma únicamente como almacenamiento temporal auxiliar que nos permitirá ordenar alfabeticamente todos los atributos

adicionados a un diccionario y de esta manera poderse desplegar en la interfaz de usuario. La variable recordNeedsSaving es empleada para detectar si ha ocurrido una modificación en algún dato del atributo para lanzar automáticamente la actualización del modelo con los nuevos datos.

El diccionario de atributos responde a eventos de usuario con los siguientes métodos:

addRecord Este método se ejecuta cuando el usuario presiona el botón "adicionar" en la interfaz de usuario y cuando se detecta que se ha modificado la información del atributo actual. Cuando el objeto DicciContol recibe este mensaje obtiene de la interfaz de usuario el nombre del atributo que se desea adicionar y verifica si ese nombre de atributo existe. En el caso que el atributo no exista se crea una instancia de la clase *Atributo* y se adiciona al diccionario (atribDict) al igual que a atribKey para poder actualizar y reordenar la lista de atributos existentes. Posteriormente son actualizados los valores de los datos del atributo con los capturados en la interfaz de usuario.

deleteRecord Este método se ejecuta cuando el usario presiona el botón "borrar" en la interfaz de usuario. Cuando el objeto DicciControl recibe este mensaje obtiene de la interfaz de usuario el nombre del atributo que fué seleccionado previamente. Este atributo es buscado y borrado del diccionario, al mismo tiempo que busca el indice de ese atributo en el arreglo para eliminarlo y por tanto actualizar la lista de atributos capturados.

blankFields Este método es ejecutado cuando el usuario presiona el botón "Limpiar" en la interfaz de usuario y después de borrar un atributo del diccionario. Cuando el objeto DiccioControl recibe este mensaje limpia el contenido de cada uno de los campos de texto de la interfaz de usuario.

handleTVClick Este método es ejecutado cuando el usario presiona el botón izquierdo del ratón sobre un atributo existente en la lista de atributos del diccionario. Cuando el objeto DicciControl recibe este mensaje identifica cual atributo fué seleccionado y despliega su información en los campos de texto de la interfaz de usuario.

extractFields Este método es ejecutado cuando se inserta o se actualiza un objeto *Atributo* y por tanto es llamado por el método addRecord. Cuando el objeto DicciControl recibe este mensaje obtiene los valores de los campos de texto de la interfaz de usuario y los almacena en las variables de instancia del objeto *atributo* actual.

populateFields Este método es ejecutado cuando se selecciona objeto *atributo* y por tanto es llamado por el método handleTVClick. Este método realiza la función inversa de extractFields. Cuando el objeto DicciControl recibe este mensaje toma los valores de las variables de instancia del objeto *atributo* actual y los despliega en los campos de texto de la interfaz de usuario.

4.1.2. El diagramador

Este módulo de nuestra aplicación permite crear diagramas EVE y por tanto se tratará todo lo relacionado con la construcción de estos diagramas.

Diseño

El diseño del diagramador esta completamente influenciado por la tecnología que fue empleada en su implementación. Por tanto se consiero la técnología de aplicaciones basadas en documentos de cocoa para el diseño. De esta manera tenemos que fué necesario considerar la herencia de las clases que proporciona esta tecnología.

La figura 4.7 nos muestra la representación del diseño de alto nivel de este módulo

En este diseño de alto nivel podemos observar los componentes del patrón de diseño MVC.

Objetos Modelo
El modelo de datos del diagramador esta conformado por todas aquellas figuras que se desplegaran como íconos dentro de un diagrama EVE, estos incluyen por tanto las características con que debe ser desplegado un objeto gráfico: posición, área y color. Así mismo contiene en las figuras de tipo Entidad y Relacion contenedores capaces de almacenar los atributos llave y los descriptores.

Objetos Vista
La vista del diagramador son los objetos graficos en cocoa los cuales corresponde a archivos .nib. Principalmente esta conformada por una panel de dibujo donde se representan los diagramas EVE. Además incluye una paleta que contiene todas las opciones de íconos que pueden ser diagramados, así como una pantalla que nos permite ver y asignar el nombre de cada *Entidad* y *Relación* incluyendo sus atributos llave y descriptores.

Objetos Control
Como objetos control tenemos a EvexWindowController, EvexView y AtribInspextor que juegan el rol de objetos vista-control, y son encargados de recibir los eventos de usuario ya sea sobre la paleta, el panel de dibujo o la pantalla de actualizar atributos. Otro objeto control es EvexDocument sin embargo este juega el rol de modelo-contol y se encarga de administrar los documetos creados.

4.1.3. Implementación

A continuación describiremos las diferentes clases que componen el diagramador , su relación entre ellas y los aspectos técnicos más importantes de su implementación.

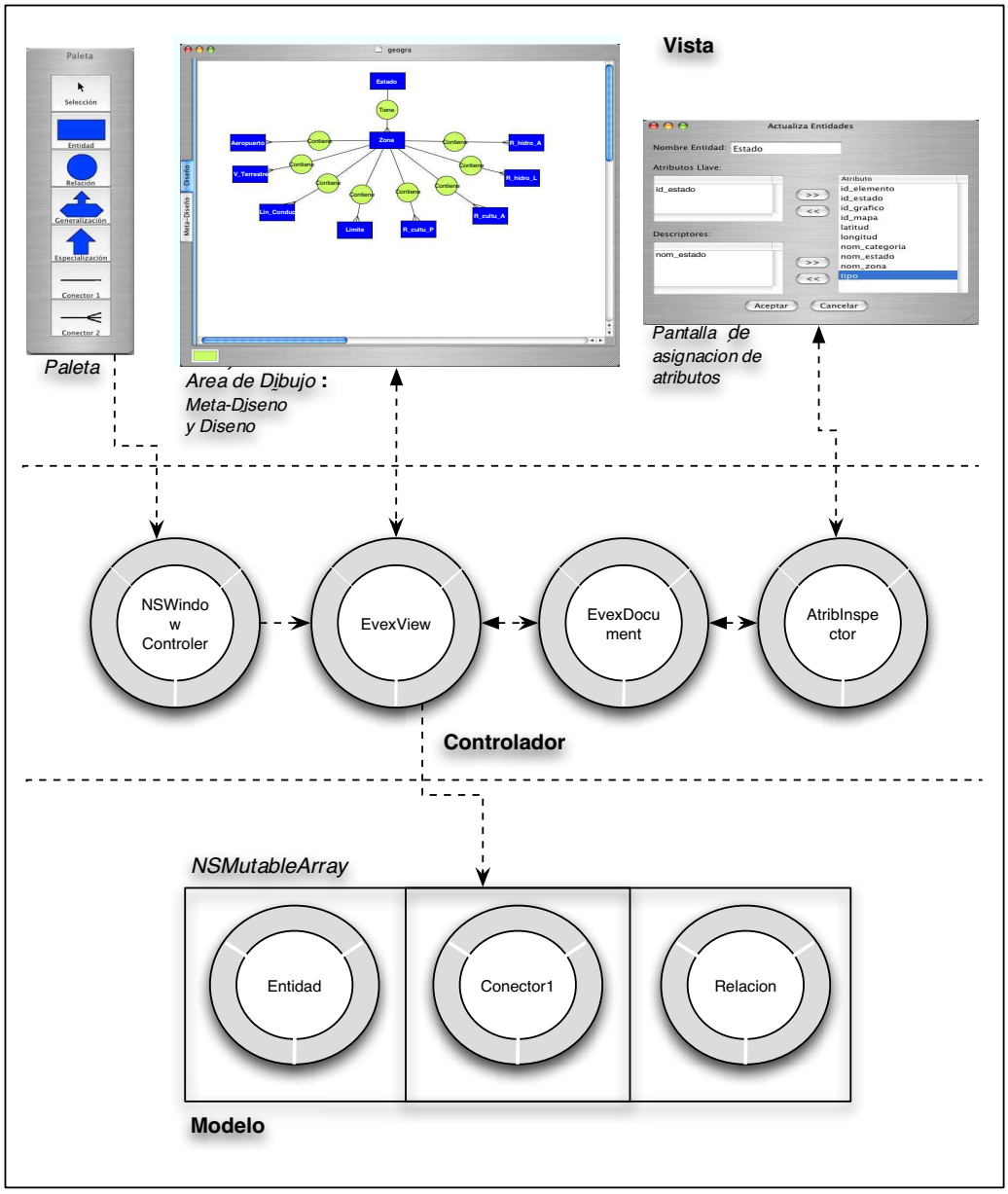


Figura 4.7: Diseño general del Editor de Diagramas EVE.

La clase *Figura*

Es una super clase que sirve de base para crear las clases de los objetos que se representan gráficamente en un diagrama EVE: Entidades, Relaciones, Especializaciones, Generalizaciones y Conectores. En consecuencia, tiene definidos tanto los atributos como los métodos comunes de todos esos objetos gráficos.

Las variables y métodos de instancia de esta clase más importantes son los que se

muestran en la figura 4.8.

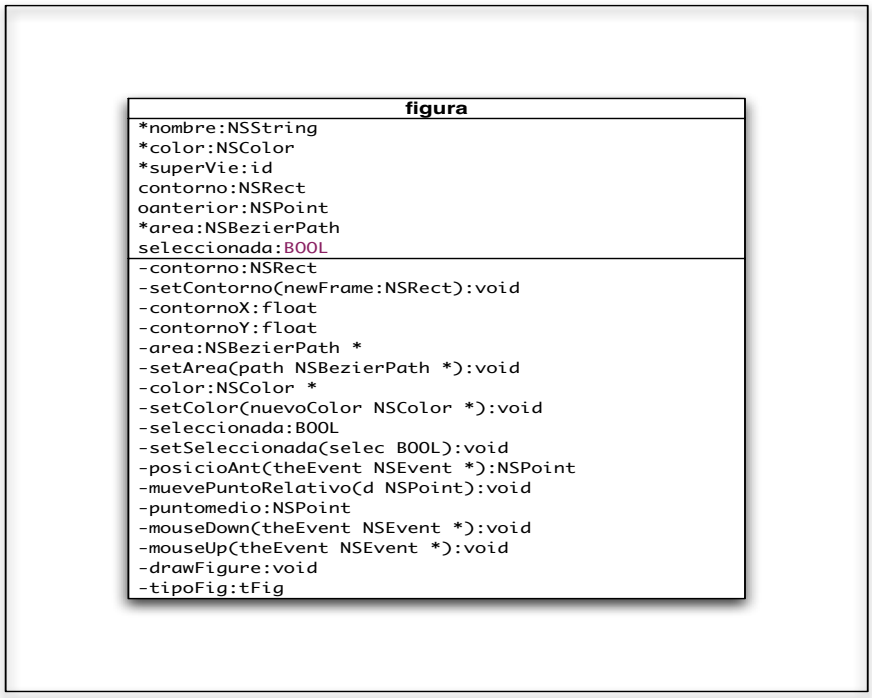


Figura 4.8: Clase Figura.

Las variables **nombre* y **color* son utilizadas para almacenar el nombre y color asignado por el usuario a cada uno de los íconos representados en un diagrama EVE. La variable superView hace referencia al objeto *EvexView* donde se desplegará el símbolo. La variable contorno es de tipo *NSRect*, el cual es una estructura que almancena tanto la posición como el tamaño del rectángulo dentro del cual se desplegaran cada uno de los íconos que conforman un diagrama. La variable *oanterior* es utilizada como una variable auxiliar para almacenar temporalmente coordenas que nos permiten simular el desplazamiento de las figuras del diagrama. La variable **area* es utilizada para almacenar el área de los símbolos: rectángulos, círculos, líneas y polígonos. Por último la variable *seleccionada* es utilizada para saber si una figura ha sido seleccionada presionando el botón izquierdo del ratón sobre de ella; este estado servirá para aplicar acciones válidas sobre los objetos que se seleccionaron tales como: cambiar de color, mover y borrar.

Los métodos de esta clase los podemos dividir por su función en cuatro tipos:

- Métodos que se dedican sólo a devolver el valor que tienen almacenado los atributos de esa clase y los métodos que llevan a cabo la tarea de asignar nuevos valores a esos mismos atributos. Estos últimos métodos inician su nombre con el prefijo “set”.

  - contorno

- setContorno
- contornoX
- contornoY
- area
- setArea
- color
- setColor
- seleccionada
- setSeleccionada

- Métodos que son utilizados para hacer posible el desplazamiento de las figuras dentro del área de dibujo.

  posicioAnt. Método que calcula el desplazamiento de la figura respecto a las coordenadas donde se encontraba la figura antes de indicar el moviento y las coordenadas de la nueva posición.

  muevePuntoRelativo. Este método cambia de posición las figuras dentro del área de dibujo considerando el desplazamiento calculado en el método *posicioAnt*.

  puntoMedio. Este método es el encargado de obtener el punto medio de las figuras, utilizado para conectar y desplazar los conectores entre figuras.

- Métodos que reciben eventos desde la interfaz de usuario.

  mouseDown. Este método es el encargado de almacenar en la variable *oanterior* el punto donde se realizó el evento de usuario en el sistema de coordenas del objeto EvexView.

  mouseUp. Este método limpia el valor de la variable *oanterior* para poder utlizarse en otro mensaje *mouseDown*.

- Métodos que sólamente se definen pero no son implementados, forman parte de un protocolo que tienen los objetos de esta familia de clases y que utilizan los objetos EvexView para desplegar a los obtetos gráficos de los diagramas EVE.

  drawFigure. Este método no es implementado en esta clase, sin embargo cada subclase debe implementarlo para poder representar un objeto gráficamente.

  tipoFig. Este método también debe ser implementado por todas las subclases que represente objetos gráficamente y debe retornar el tipo de figura que se trate: Entidad, Relación, Generalización, Especializacion, Conector.

Una vez que hemos analizado la clase base de los símbolos que se representan gráficamente en un diagrama EVE, revisaremos las clases que heredan los atributos y los métodos de esta super clase.

Las clases que heredan de la superclase *Figura* son clases que se encargan de crear los objetos que contienen las características de como deben representarse los diferentes íconos gráficamente dentro de un diagrama EVE. Esas clases son:Entidad, Relación, Generaliza y Especializa mismas que heredan tanto los atributos como los métodos de la superclase.

La clase *Entidad*

Esta clase es la encargada de crear los objetos que respresentan gráficamente a las Entidades en forma de un rectángulo dentro de un diagrama EVE. La figura 4.9 nos muestra los principales atributos y métodos de esta clase.

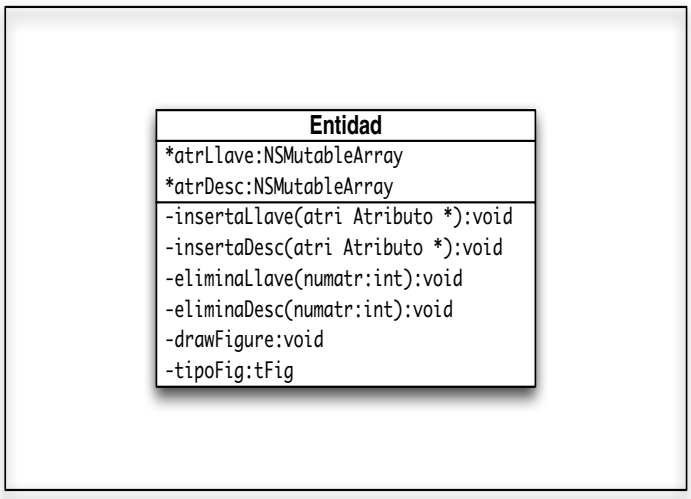


Figura 4.9: Clase Entidad.

La variable de instancia **atrLlave* es utilizada para almacenar aquellos atributos considerados como llave. La variable **atrDesc* almacena los atributos considerados como descriptores.

Los métodos de instancia de esta clase los describimos a continuación:

drawFigure. Este método es el encargado de crear un objeto *NSBezierPath* con la forma de un rectángulo y almacenarlo en atributo correspondiente al área de este objeto, también se asigna el color que se haya seleccionado. Estas propiedades que se asignan en este método permiten que el objeto se presente gráficamente en un objeto EvexView.

tipoFig. Este método es el encargado de devolver el tipo de figura ENTIDAD.

insertaLlave. Este método es ejecutado cada que se asigna un atributo llave. Cada que una instancia *Entidad* recibe este mensaje, adiciona un atributo al arreglo que almacena los atributos llave.

insertaDesc. Este método es ejecutado cada que se asigna un atributo descriptor. Cada que una instancia *Entidad* recibe este mensaje, adiciona un atributo al arreglo que almacena los atributos descriptores.

eliminaLlave. Este método es ejecutado cada que se elimina un atributo llave. Cuando se recibe este mensaje se elimina un atributo del arreglo que almacena los atributos llave.

eliminaDesc. Este método es ejecutado cada que se elimina un atributo descriptor. Cuando se recibe este mensaje se elimina un atributo del arreglo que almacena los atributos descriptores.

La clase *Relación*

Esta clase se encarga de crear los objetos que representan gráficamente a las relaciones en forma de un círuclo dentro de un diagramas EVE. La figura 4.10 nos muestra los principales atributos y métodos de esta clase.

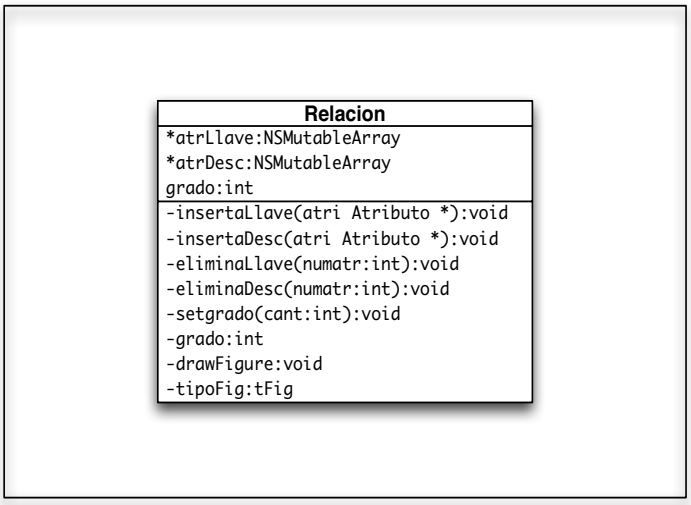


Figura 4.10: Clase Relación.

Las varibles de instancia de esta clase son las mismas que en la clase *Entidad* ya que en los objetos creados con ambas, es necesario asignar atributos llave y descriptores. Sólo la variable *grado* se adiciona a la clase *Relacion*, esta variable es la encargada de almacenar el grado de la relación, es decir, el número de entidades con las que se asocia. El contenido

de esta variable es considerada cada que se realiza una conección para verificar que no rebase el grado 3 permitido por cada relación.

Los métodos de esta clase tienen la misma funcionalidad que en la clase *Entidad*, por tanto sólo explicaremos aquellos que son especifícos de esta clase:

setGrado. Este método se ejecuta cada que se crea o elimina una conección, y se encarga solamente de modificar el valor almacenado.

grado. Este método se ejecuta cuando se desea saber el grado de una relación, y sólo devuelve el valor almacenado.

drawFigure. Este método crea un objeto NSBezierPath con la forma de un círculo y lo almacena en el atributo correspondiente al *area* de este objeto, también se asigna el color que se haya seleccionado. Las propiedades que se asignan en este método permiten que el objeto se visualice gráficamente en un objeto *EvexView*.

tipoFig. Este método es el encargado de devolver el tipo de figura RELACION.

La clase *Generaliza*

Esta clase se encarga de crear los objetos que representan gráficamente las generalizaciones en forma de una flecha apuntando hacia arriba con base dentro de un diagramas EVE. En la figura 4.11 visualizamos que no se anexaron atributos y sólo se redefinen los siguientes métodos.

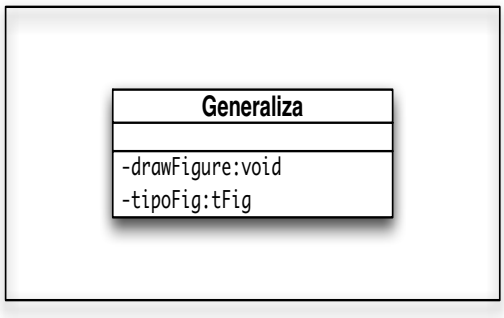


Figura 4.11: Clase Generaliza.

drawFigure. Este método crea un objeto *NSBezierPath* con la forma de un polígono en forma de una flecha apuntando hacia arriba con base y a diferencia de las clases *Entidad* y *Relacion*, no existe un BezierPath en c-objetivo que devuelva una figura con las características requeridas, por tanto este método es diferente y más extenso. Primero se crean todos los puntos (par de coordenadas X,Y) que serán los vertices de la figura, estos puntos son relativos al punto que se obtiene por el usuario desde

la interfaz gráfica. Posteriormente estos puntos se van agregando uno a uno a un BezierPath vacío (sin forma), hasta conformar la figura. Este BezierPath es almacenado en el atributo correspondiente al *area* de este objeto. El color de la generalización también es asignado en este método. Las propiedades que se asignan en este método permiten que el objeto se visualice gráficamente en un objeto EvexView.

tipoFig. Este método es el encargado de devolver el tipo de figura GENERALIZA.

La clase *Especializa*

Los objetos de esta clase representan a las especializaciones en los diagramas EVE. Su representación gráfica dentro del objeto EvexView es una flecha apuntando hacia arriba. La figura 4.12 nos muestra los principales métodos que se redefinieron en esta clase.

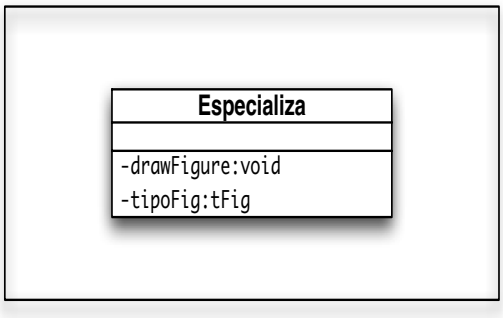


Figura 4.12: Clase Especializa.

drawFigue. Este método crea un objeto NSBezierPath con la forma de un polígono en forma de flecha apuntando hacia arriba y al igual que en la clase *Generaliza* no se cuenta con un BezierPath en c-objetivo que cree una figura con la forma requerida, por tal razón también son generados primeramente los puntos que conforman los vertices de la figura y posteriormente se anexan a un BezierPath sin forma para obtener la figura deseada. El BezierPath es almacenado en la variable de instancia *area*, así mismo se almacena el color elegido.

tipoFig. Este método es el encargado de devolver el tipo de figura ESPECIALIZA.

Existen dos subclases más de la superclase *Figura* y son los conectores (Conector1 y Conector2), los tratamos por separado de los demás símbolos gráficos porque contiene un número mayor de carácteristicas especiales. Al igual que los demas símbolos redefinieron el método tipoFig para devolver el tipo de símbolo de que se trata.

Los conectores son como su nombre lo dice, los encargados de conectar los símbolos que previamente describimos. Existen dos tipos de conectores porque con ellos representamos

la conectividad de las relaciones. Los valores de la conectividad de las instancias es uno (1) o muchos (M).

La clase *Conector1*

La clase Conector1 es la encargada de crear los conectores que representan la conectividad uno (1) y gráficamente se representan con una línea recta que conecta dos instancias gráficas. La figura 4.13 nos muestra los principales atributos y métodos de esta clase.

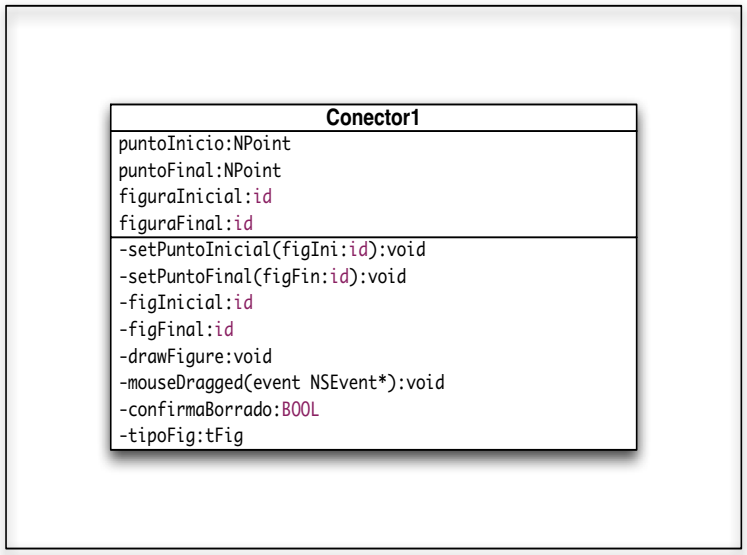


Figura 4.13: Clase Conector1.

La variable de instancia *puntoinicio* contiene la coordenada X,Y donde comienza dicho conector. Este punto de inicio debe ser el punto medio de una figura. Con esto no se quiere decir que exactamente se debe presionar el botón del ratón sobre el punto medió del símbolo gráfico sino que solamente es necesario elegir el símbolo y la aplicación es la encargarda de asignar la coordenada del punto medio de la figura. La variable *puntofinal* almacena la coordenada X,Y donde finaliza la línea recta del conector, es decir, el punto medio de otra símbolo gráfico. La variable *figuraInicial* almacena la figura donde comienza la línea recta del conector y la varible *figuraFinal* almacena la figura donde termina dicha línea. Es necesario saber que tipo de figura son las que se desan conectar para validar si esta conección es o no válida.

Los primeros dos métodos de instancia son los encargados de actualizar los valores de las variables de instancia a los que hace referencia su nombre. Los siguientes dos métodos devuelven el valor almacenado al que hace referencia su nombre.

Los métodos restantes se ejecutan como a continuación describimos:

drawFigure. Este método crea un objeto *NSBezierPath* en forma de una línea recta con

origen en el punto medio de una figura y como punto final cada coordenada por donde pasa el ratón sobre el objeto EvexView (panel de dibujo). La última actualización del punto final se realiza en cuanto se llega a una segunda figura con la cual se conecta.

mouseDragged. Este método es el encargado de mandar actualizar la variable *puntoFinal* para que de esta manera cuando es creado un conector dé el efecto de movimiento y crecimiento de la línea recta que lo representa gráficamente hasta alcanzar su instancia final con que debera de conectarse.

confirmaBorrado. Este método es el encargado de eliminar aquellas conecciones que no son válidas dentro del diagrama, es decir, este método es el que dirige la edición por sintaxis de los diagramas EVE. Cabe destacar que en el momento que se inicia el trazo de un conector no es posible definir si es válido o no, ya que esto depende del los símbolos que se deseen conectar.

En la figura 4.14 podemos observar cuales son las conecciones entre símbolos que se consideran por sintaxis como válidas.

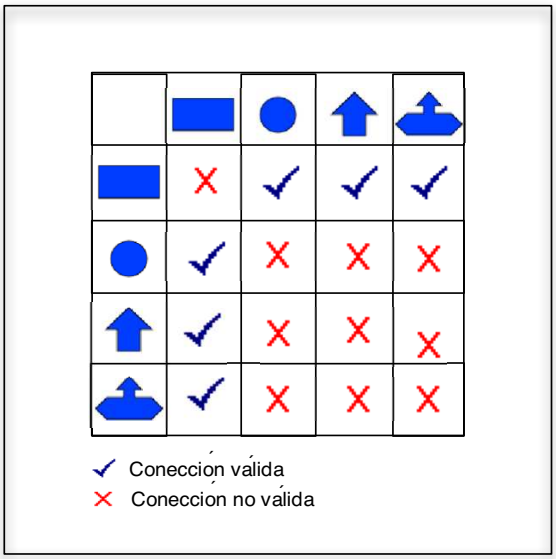


Figura 4.14: Conexiones válidas entre íconos con el Conector1

La clase *Conector2*

La clase Conector2 es la encargada de crear los conectores que representan la conectividad muchos (M) y gráficamente se representan como una línea recta que parte del punto medio de una relación y finaliza con tres líneas semejando una pata de gallo en el extremo de una entidad. En la figura 4.15 se muestran los principales atributos y métodos de esta clase.

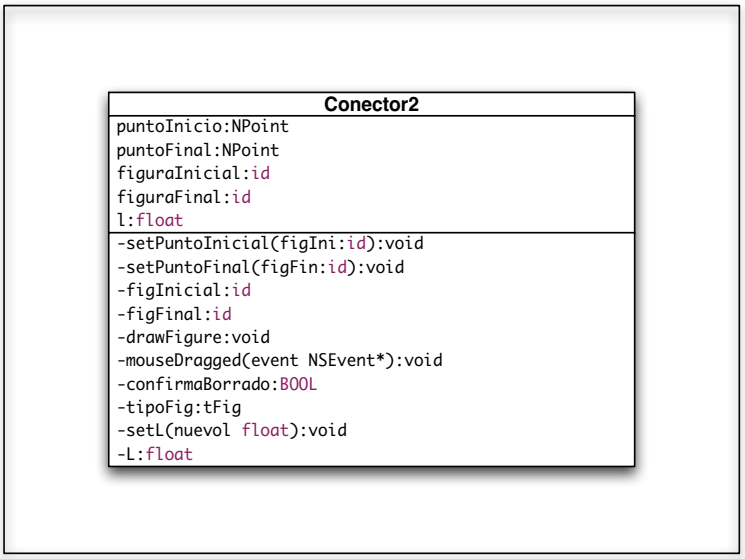


Figura 4.15: Clase Conector2.

Los atributos o variables de instancia de esta clase son los mismos que en la clase conector1: *puntoInicio, puntoFinal. figuraInicial* y *figuraFinal*. Sólo se adiciona un atributo mas que es la variable *l*. Esta variable es utilizada para representar un detalle gráfico, ya que auxilia en el proceso de variar el tamaño de la pata de gallo que se encuentra al final de este tipo de conector. Esta variación de tamaño es necesaria para mantener un buen nivel de estética en el diagrama a pesar de la posición de los símbolos que son posibles conectar. Los únicos símbolos que pueden conectarse con este tipo de conector son las relaciones y entidades, llevando el extremo que finaliza en forma de pata de gallo del conector hacia el lado de la entidad y no asi de la relación.

Los métodos de esta clase son en su mayoria los mismos que los de la clase Conector1, y realizan las mismas tareas, sin embargo contamos con tres métodos más que hacen la diferencia y se ejecutan como a continuación se describe:

drawFigure. Este método crea un objeto *NSBezierPath* con forma de una línea recta que en uno de sus extremos finaliza con tres líneas semejando una pata de gallo. En C Objetivo, no existe un BezierPath con las características necesarias para representar este conector, por tanto primeramente creamos el objeto BezierPath sin forma y creamos los puntos que serán los vertices de la figura, posteriormente adicionamos uno a uno al BezierPath hasta conformar la forma del conector en su totalidad. El BezierPath obtenido se almacena en la variable de instancia *area* de este objeto.

setL Este método es el encargado de actualizar la variable de instancia *l*, es invocado cada que el indicador del ratón del usuario avanza dentro del objeto EvexView.

L Este método es el encargado de devolver el valor de la variable *l*, es invocado cada vez que se despliega graficamente el conector.

La clase *EvexDocument*

EvexDocument es la clase que juega el rol de modelo-controlador de la arquitectura de aplicaciones basadas en documentos, Los objetos de esta clase mantienen una relación estrecha con los objetos de las clases EvexWindowController y EvexView. Y es la encargada de contolar las operaciones que pueden realizarse con los documento (diagramas EVE). Estas operaciones son: apertura de documentos existentes, creación de nuevos documentos, almacenamiento de documentos existentes, almacenamientos de nuevos dococumentos.

Los métodos más importantes de esta clase pueden observarse en la figura 4.16 y se trabajan de la siguiente manera:

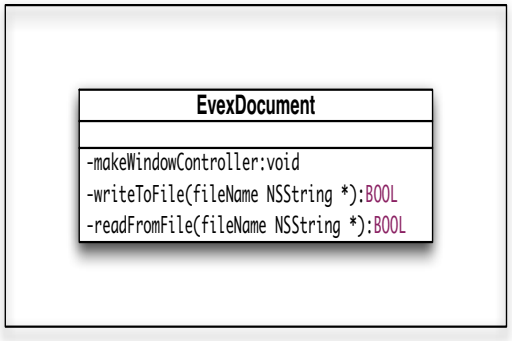


Figura 4.16: Clase EvexDocument.

makeWindowControllers. Este método es el encargado de crear un controlador de la clase EvexWindowCotroller para poder conectar la paleta de herramientas con los documentos de reciente creación o documentos ya existentes.

writeToFile. Este método se ejecuta cuando el usuario selecciona la opción *Save* y *SaveAs* del menú *File*. Cuando el documento no tiene asociado ningún nombre, se despliega un objeto NSSavePanel, que es una pantalla en la cual se solicíta al usuario insertar el nuevo nombre del docuemento. En el caso que el documento ya contenga un nombre, es decir, que previamente se haya creado y almacenado y sólo se requiera almacenar cambios, solamente se ejecuta la acción de almacenar con el mismo nombre existente.

readFromFile. Este método se ejecuta cuando el usuario selecciona la opción *Open* del menú *File*. En este momento se crea un nuevo documento y se carga la información del documento que se seleccione en un Panel que muestra todos los archivos .eve que se encuentran en un determinado directorio.

La clase *EvexWindowController*

Esta clase hereda de la superclase NSWindowController por tanto sus instancias juegan el rol de un objeto vista-controlador. Es necesario crear instancias de esta clase para poder asociar la paleta de herramientas con un objeto EvexView. Este controlador se encarga de recibir el evento de usario que fué ejecutado en la paleta de herramientas, es decir, es el encargado de recibir la selección de un elemento en la paleta de herramientas y activarlo como el ícono actual que puede desplegar un objeto graficos relacionado a él dentro de un objeto EvexView.

Los métodos más importantes se muestran en la figura 4.17 y se ejecutan como a continuación se describe:

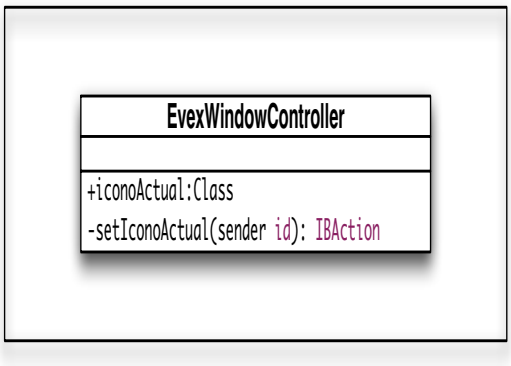


Figura 4.17: Clase EvexWindowController.

iconoActual Este método es el encargado de regresar la clase a que corresponde el objeto seleccionado en la paleta de herramientas, y utiliza como parámetro la variable **claseNombre* de tipo *NSString* que contiene el nombre de la clase obtenido en el método *setIconoActual*.

setIconoActual Este método es un método de clase que es el encargado de obtener el nombre de la clase del objeto gráfico que el usuario ha seleccionado en la paleta de herramientas. Esto es, si se selecciona en la paleta de herramientas el ícono correspondiente a un objeto *Entidad*, el ícono tiene el nombre *Entidad*, que también corresponde con el nombre de la clase del objeto se se agregará al diagrama EVE actual.

La clase *AtribInspector*

Esta clase es la encargada de asignar nombre a las *Entidades* y *Relaciones* así como sus atributos llave y descriptores. Principalmete se encarga de detectar cual es el atributo seleccionado de toda la lista capturada en la pantalla del editor del diccionario de atrubutos. Para después poder agregarla a la lista de atributos llave o descriptores. Todos los

movientos realizados se almacenan a la hora de seleccionar la opción "Aceptar", o pueden no almacenarse al seleccionar la opción "Cancelar".

AtribInspector contiene atributos y métodos de instancia y los más significativos se muestran en la figura 4.18 y se describen a continuación:

Las variables de esta clase hacen posible una comunicación con objetos de la interfaz de usuario y por tanto manipular y desplegar información en ella. La variable *nombre* es un campo de texto que nos permite editar el nombre de la *Entidad* o *Relacion* que fue seleccionada en el área de dibujo. La variable **atrDicT* nos permite desplegar la lista de todos los objetos *Atributo* adicionados en el diccionario de atributos. La variable **atrLlaveT* nos permite desplegar los atributos que asignemos como atributos llave. La variable **atrDescT* nos permite desplegar los atributos que asignemos como atributos descriptores.

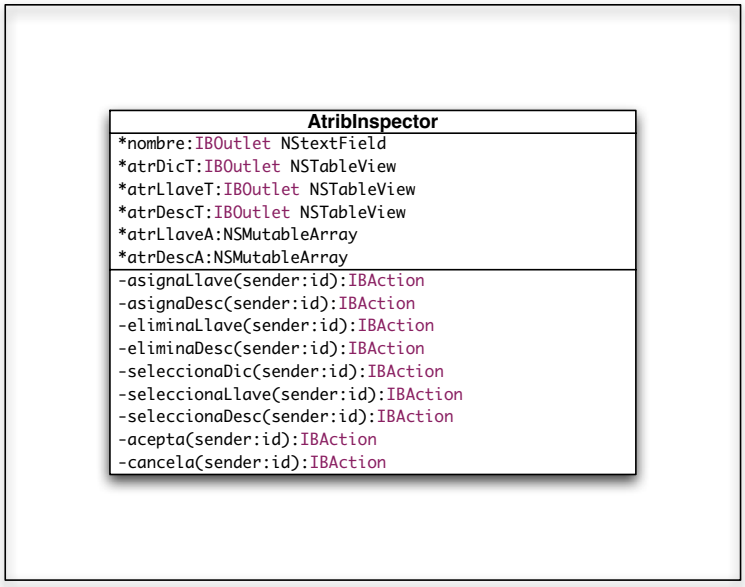


Figura 4.18: Clase AtribInspector.

Los métodos más importantes se ejecutan como a continuación se describe:

asignaLlave. Este método se ejecuta cada vez que se ha seleccionado un atributo de la lista de atributos del lado derecho de la pantalla y es presionado el botón con la leyenda "<<" que se encuentra ubicado al lado de la lista de atributos *Llave*. Lo que produce un cambio en esa lista, adicionando este atributo seleccionado.

asignaDesc. Este método se ejecuta cada vez que se ha seleccionado un atributo de la lista de atributos del lado derecho de la pantalla y es presionado el botón con la leyenda "<<" que se encuentra ubicado al lado de la lista de atributos *Descriptores*. Lo que produce un cambio en esa lista, adicionando este atributo seleccionado.

eliminaLlave. Este método se ejecuta cada vez que se selecciona un atributo de la lista de atributos *Llave* ubicada en la parte superior izquierda de la pantalla y presionando el botón con la leyenda ">>" que se encuentra al lado de esa lista. Esto produce que se elimine el atributo de la lista.

eliminaDesc. Este método se ejecuta cada vez que se selecciona un atributo de la lista de atributos *Descriptores* ubicada en la parte inferior izquierda de la pantalla y presionando el botón con la leyenda ">>" que se encuentra al lado de esa lista. Esto produce que se elimine el atributo de la lista.

seleccionaDic. Este método es ejecutado cuando el usario selecciona con el ratón un atributo existente en la lista de atributos del diccionario. Cuando el objeto AtribInsperctor recibe este mensaje identifica cual atributo fué seleccionado y lo sombrea en color azul.

seleccionaLlave. Este método es ejecutado cuando el usario selecciona con el ratón un atributo existente en la lista de atributos *Llave*. Cuando el objeto AtribInsperctor recibe este mensaje identifica que atributo fué seleccionado y lo sombrea en color azul.

seleccionaDesc. Este método es ejecutado cuando el usario selecciona con el ratón un atributo existente en la lista de atributos *descriptores*. Cuando el objeto AtribInsperctor recibe este mensaje identifica cual atributo fué seleccionado y lo sombrea en color azul.

acepta. Este método es ejecutado cuando se presiona el botón *Aceptar*. Cuando se recibe este mensaje, se almacenan las asignaciones realizadas en los contenedores correspondientes de cada *Entidad* y *relación*.

cancela Este método es ejecutado cuando se presiona el botón *Cancelar*. Cuando se recibe este mensaje, se ignoran las asignaciones realizadas.

La clase *EvexView*

Esta clase es la que cobra mayor importancia dentro del proyecto, ya que es la clase que permite desplegar gráficamente todos los objetos de la paleta de herramientas y asi construir diagramas EVE. *EvexView* es una subclase de *NSView* de cocoa C obejtivo, por tanto permite recibir eventos de usuario ya sea desde el teclado o del ratón, además de desplegar información en su área de dibujo. Dentro del la estructura de nuestra aplicación basada en el patrón MVC, esta clase juega el rol del elemento Vista, sin embargo las características heredadas de la super clase *NSView* amplian su función abarcando un rol de Controlador. Siendo de esta manera clasificada en nuestro diseño como un elemento vista-controlador. Los objetos gráficos que pueden desplegarse en esta área de dibujo son los objetos creados por las clases que heredan de la clase Figura (Entidad, Relación, Generaliza, Especializa, Conector1 y Conector2). De esta manera es clara la existencia de

una estrecha relación entre estas clases. Otra relación importante es la que se tiene con el objeto controlador de la clase EvexWindowController, que es el que indica cual es el tipo de ícono seleccionado y por tanto el ícono que puede ser desplegado en el área de dibujo.

EvexView contiene atributos y métodos de instancia y los más significativos se muestran en la figura 4.19 y se describen a continuación:

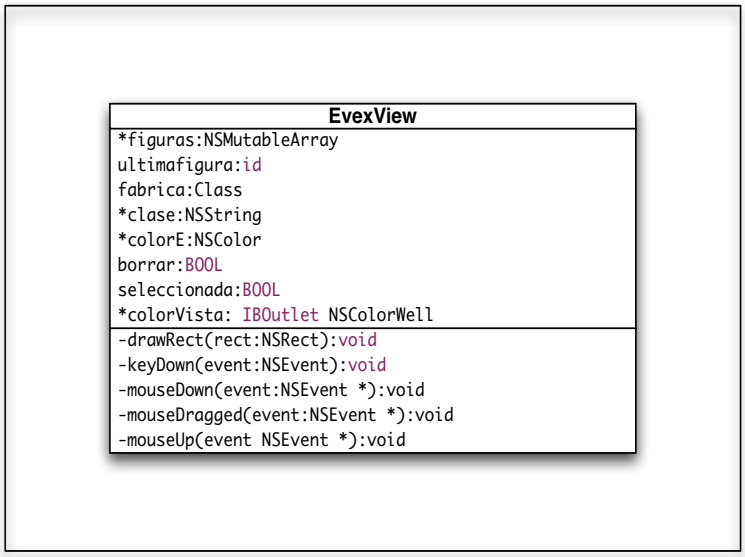


Figura 4.19: Clase EvexView.

La variable **figuras* es un contenedor que se utiliza para almacenar todas los íconos que se despliegan gráficamente, es decir, todos los íconos que conforman un diagrama EVE. La variable *ultimafigura* almacena el último objeto que ha sido diagramado o bien el último objeto que se ha involucrado con los eventos:*mouseDown, mouseDragged* y *mouseUp*. Esto permite realizar acciones definidas sobre ese objeto tales como: seleccionarlo, eliminarlo, conectarlo con otros objetos gráficos, etc. La variable *fabrica* almacena la clase del objeto que se daigramará en el documento actuaL. La clase es obtenida del objeto EvexWindow-Controller, cuado éste detecta el evento de usuario al seleccionar un ícono de la paleta de herramientas. La variable *colorE* almacena el color que se ha seleccionado para los íconos, esta varable es actualizada cada que se selecciona un nuevo color desde el objeto visual de colores. La variable *borrar*, es una bandera que auxilia a controlar el borrado de los íconos de un diagrama EVE. Por último, la variable **colorVista* es una variable IBOutlet, este tipo de variable permite conectarnos directamente con la interfaz de usuario; es utilizada para recibir el evento de usuario de selección de color.

Los métodos de instancia de los objetos EvexView se utilizan principalemente para la edición gráfica y asi desplegar graficamente los diagramas EVE de un documento. Estos métodos trabajan de la siguente forma:

drawRect. Este método se ejecuta cada vez que se requiere actualizar el área de dibujo,

es decir, después de recibir eventos de usuario como: dibujar un nuevo ícono, seleccionarlo, borrarlo, moverlo, etc. Su función es limpiar el área de dibujo y mandar a dibujar cada uno de los objetos que se tienen almacenados en el contenedor *figuras*.

keyDown. Este método es ejecutado cuando se tiene seleccionado un ícono y se presiona la tecla *delete*, su función es la de borrar los objetos que se tienen seleccionados.

mouseDown. Este método es ejecutado cada vez que se presiona el botón del ratón sobre el área de dibujo. Su implementación es más compleja que cualquier otro método de las clases que hemos explicado, ya que puede responder de diversas maneras dependiento de la situación actual en un documento. Estas respuestas pueden ser: crear y agregar un nuevo ícono al diagrama EVE (el que se haya seleccionado en la paleta de herramientas), seleccionar un objeto para una acción posterior(eliminarlo, moverlo, cambiarlo de color), o para agregarlo a un grupo de íconos seleccionados que tambien podrán realizar alguna accíon posteriormente.

Lo primero que se realiza al recibir un mensaje a este método es obtener las coordenadas dentro del área de dibujo donde ocurrio el evento de usuario. Después se verifica que ícono se tiene seleccionado en la paleta de herramientas -el ícono que se tiene por defecto cuando se inicia un documento es el de selección-; para de esta manera saber de que forma responder al evento. En caso de que el objeto creado sea *Entidad, Relacion, Generaliza o Especializa*, éste se adiciona al contenedor *figuras* donde se almacena los íconos del diagrama EVE actual y por último se envía un mensaje *mouseDown* a ese objeto para que almacene el punto en donde se registro el evento de usuario, este dato es importante porque auxiliar las actividades que realiza el método *mouseDragged*. Si el objeto que se ha creado es un *Conector1* o *Conector2* se verifica que el evento de usuario se haya realizado sobre algún objeto *Entidad, Relacion, Generaliza* o *Especializa*, de ser así, ese objeto permanece activo, pero no se agrega aún al diagrama EVE hasta que se verifique que el objeto destino sea sintácticamente válido. De no haberse identificado el evento de usuario, simplemente no se anexa al diagrama. Si no ha sido seleccionado desde la paleta algún elemento gráfico, es decir, el ícono activo es el de *seleccionar* no se crea ningún objeto, sin embargo, es necesario verificar si el evento de usuario ocurrio sobre algún ícono diagramado. Esto se lleva a cabo revisando si las coordenadas del evento de usuario se encuentran dentro del área de alguno de los elementos del diagrama. Si algún objeto lo contiene, entonces se procede a seleccionarlo -se cambia de color su contorno para que gráficamente se distinga el objeto seleccionado- o deseleccionarlo según sea el caso.

- Si el objeto sobre el cual se ha detectado el evento de usuario se encuentra seleccionado, éste se deselecciona.
- Si el objeto sobre el cual se ha detectado el evento de usuario no se encuenttra seleccionado, éste de selecciona.

- Si se tiene presionada la tecla shift al momento de verificar el evento de usario, se puede hacer una selección multiple o deselección en caso de que el objeto se encuentre previamente seleccionado.

Después de ejecutarse cualquiera de los casos anteriores , se manda a redibujar el objeto EvexView para actualizar los íconos del diagrama EVE.

mouseDragged. Este método es ejecutado cuando el usario arrastra el indicador del ratón sobre el área de dibujo. Cuando el objeto EvexView recibe este mensaje lo que debe de hacer es cambiar la posición de los objetos *Entidad, Relacion, Generaliza* o *Especializa* o dirigir un nuevo objeto *Conector1* o *Conector2* hacia el objeto final que conectará. Para identificar que acción ha de seguirse es necesario considerar la acción que se realizó en el evento *mouseDown*. Si la variable *utlimaFigura* hace referencia a un valor nulo, es decir, tener el ícono *seleccionar* activo, entonces se deben de mover los objetos que se encuntran seleccionados: *Entidad, Relacion, Generaliza* o *Especializa* y en caso que estos íconos tengan asociados uno o más conectores, éstos deben también ser desplazados en relación a los objetos que conectan. Si la variable *utlimaFigura* hace referencia a un *Conector1* o *Conector2* en el evento *mouseDown* ese objeto debe cambiar su dimención para dar el efecto de moverse hacia otro objeto dentro del área de dibujo. Este efecto se obtiene al estar actualizando el punto final del conector con las coordenadas del evento de usuario.

mouseUp. Este método es ejecutado cuando se libera el botón del ratón sobre el área de dibujo. Cuando el objeto EvexVier recibe este mensaje verifica si la variable *ultimaFigura* es un *Conector1* o *Conector2*, en caso de que efectivamente se trate de un objeto de este tipo, se comprueba primeramente que las coordenadas del evento de usuario se encuentren dentro de área de un ícono del diagrama. En el caso de que un objeto si contenga esas coordenadas, se verifica por una llamada a un método del conector del que se trate si ese objeto destino del conector es un objeto gráfico válido para concluir la conección en caso contrario ese conector es eliminado.

4.2. Diagrama Funcional

En esta sección mostramos el diagrama funcional de la aplicación donde se refleja la implementación vista en la sección anterior (fig 4.20).

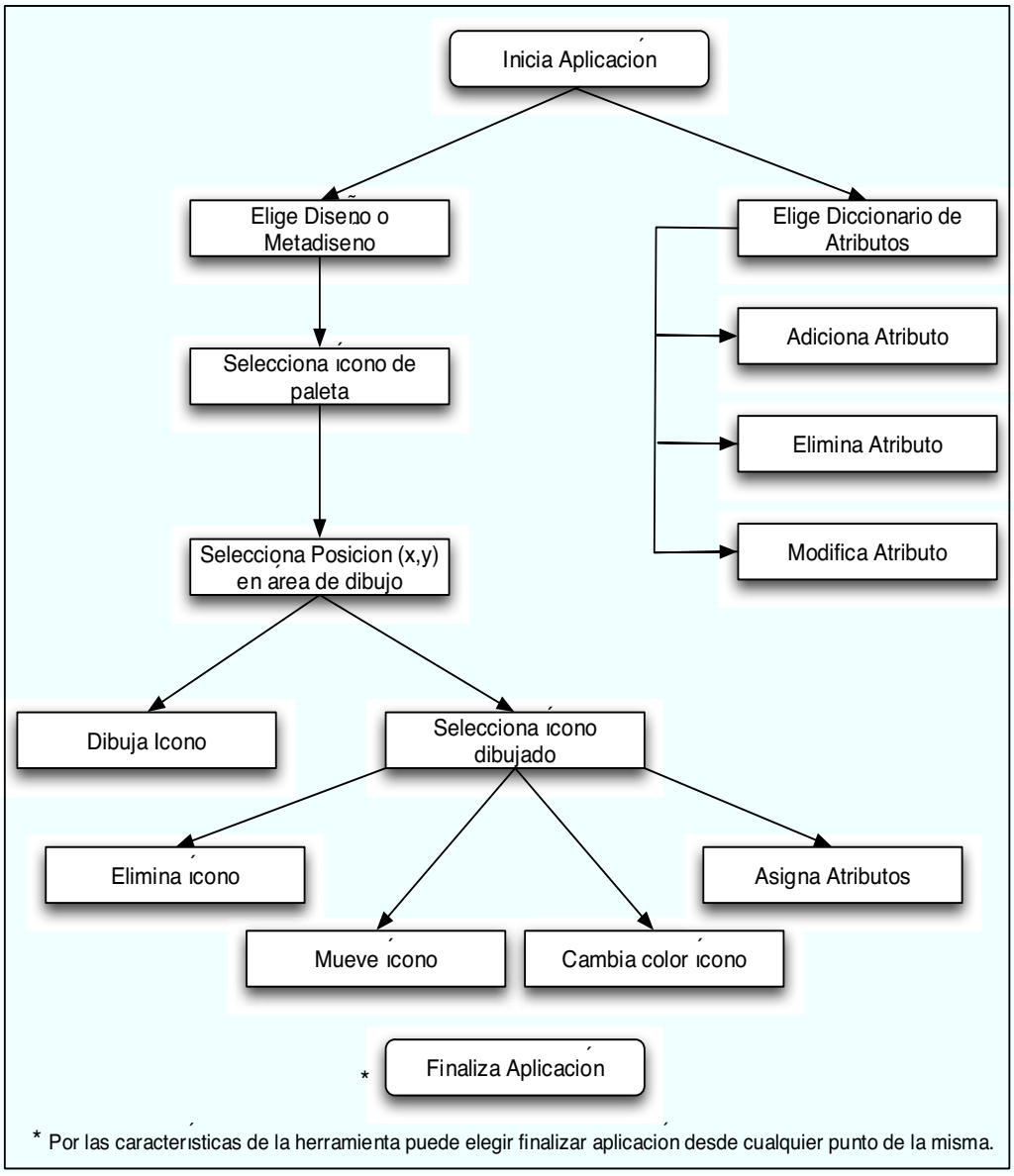


Figura 4.20: Diagrama funcional de EVEMac.

Capítulo 5

Caso de Estudio

En este capítulo se presenta la aplicación de la metodología de diseño de bases de datos apoyada con la herramienta EVEMac. Se desarrolla el diseño de dos tipos de proyectos diferentes. El primero es un diseño para una base de datos de un sistema de información geográfica y el segundo para una base de datos de un sistema para automatizar el control de las operaciones realizadas en un almacen.

En ambos casos de estudio iniciamos con la descripción del problema; el cual tomaremos como punto de partida para el diseño conceptual. Primeramente se identificaran los atributos, entidades y vínculos. Con el editor del diccionario de datos se capturan todos los atributos reconocidos. Con el diagramador de EVEMac se construye el diagrama que es considerado como el modelo conceptual. Por último, se asignan atributos a cada entidad o relación, diferenciando entre los atributos llave y los atributos descriptores.

5.1. Diseño de una base de datos geográfica

La herramienta EVEMac nos permite realizar el diseño de bases de datos en dos niveles: Metadiseño y diseño conceptual tradicional. En este primer caso de estudio no emplearemos el nivel de Metadiseño, con la finalidad de ejemplificar que puede no utilizarse, es decir, se presenta como una opción de diseño de nuestra metodología la cual apoya las tareas del control y administración de proyectos sin embargo, no toma el carácter de fase obligatoria dentro del proceso del diseño de bases de datos.

5.1.1. Nivel de diseño

Descripción preliminar

En la organización X se pretende realizar un Sistema de Información Geográfica (SIG). El cual servirá como una herramienta de análisis de información. La información debe tener una referencia espacial para poder hacer representaciones geográficas.

Este sistema deberá tener la capacidad para poder dar respuesta a las siguientes preguntas [16]:

1. Dónde está el objeto A?

2. Dónde está A con relación a B?

3. Cuantas ocurrencias del tipo A hay en unda distancia D de B?

4. Cuál es la dimensión de B?

5. Cuál es el resultado de la intersección de diferentes tipos de información?

6. Que hay en el punto (Latitud, Longitud)?

7. Qué objetos estas próximos a aquellos objetos que tienen una combinación de características?

Este sistema por tanto deberá representar objetos con una ubicación definida sobre la superficie terreste bajo un sistema convencional de coordenadas. Un objeto es considerado como cualquier elemento relativo a la superficie terrestre que tiene un tamaño es decir, una dimensión (alto-ancho-largo) y una localización espacial o un espacio relativo a la superficie terrestre. A todo objeto se asocian atributos que pueden ser: graficos y no graficos. Los atributos gráficos de un objeto son las representaciones de los objetos geográficos asociados con ubicaciones específicas en el mundo real. La representación de los objetos se hace por medio de puntos, líneas o áreas. Los atributos no gráficos o tambien llamados alfanuméricos, corresponden a las descripciones, cualificaciones o características que nombran y determinan los objetos o elemnetos geográficos [12].

El sistema que automatizará este proceso de análisis de información geográfica contendrá por lo anteriormente descrito, datos gráficos y no gráficos. Los datos gráficos son la digitalización de imagenes que representan a las diferentes capas de las cartas topográficas de interés [1]. Son almacenadas en diferentes formatos y estructuras de almacenamiento que dependen de la tecnología empleada en el proceso. Por otro lado se encuentran los datos no gráficos, para lo cual es necesario diseñar una estructura de base de datos que permita el óptimo almacenamiento y por ende el mejor desempeño de la aplicación.

Identificación de entidades, atributos e interrelaciones.

Cómo primer paso del diseño conceptual de nuestra base de datos, se analiza la descripción del problema y se obtienen los elementos a modelar: entidades, atributos e interrelaciones. (Tabla 5.1)

[1] Las capas(categorias) que se han consierado para este caso de estudio son las que se encuentran en las cartas topográficas del INEGI

Entidades	Atributos	Relaciones
Estados	id_estado	contiene
Zonas	nom_estado	esta
Categorias	id_zona	es
Vias Terrestres	nom_zona	
Aeropuertos	id_categoria	
Lineas de conducción	nom_categoria	
Rasgos Culturales	id_elemento	
Rasgos Hidrográficos	id_grafico	
	id_mapa	
	longitud	
	latitud	
	tipo_vectorial	

Tabla 5.1: Identificación de entidades, atributos e interrelaciones.

Diccionario de atributos

Como segundo paso, especificamos el tipo de dato, longitud y mascara que tendran cada uno de los atributos. La tabla 5.2 muestra todos los datos que deben capturarse. Esa captura se realiza en la pantalla que aparece al seleccionar la opción "Diccionario de Atributos" del menú *tools* (fig 5.1)

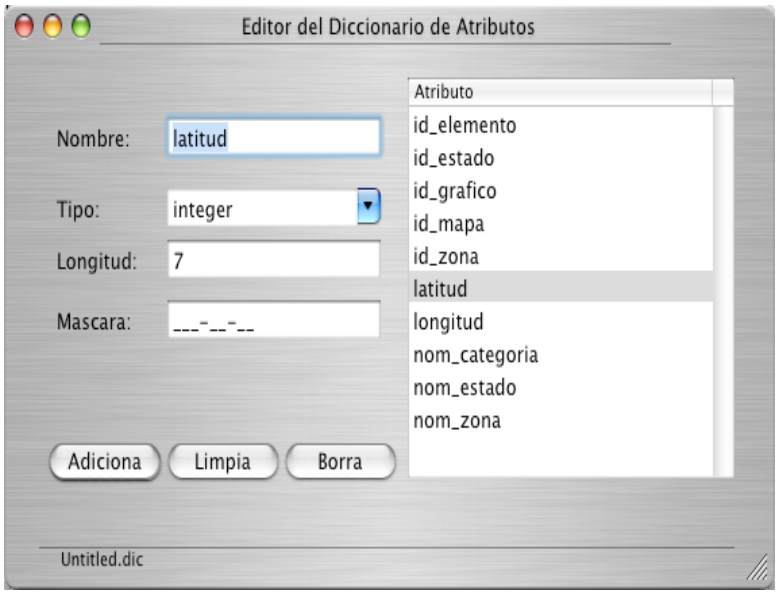


Figura 5.1: Pantalla del diccionario de atributos.

NOTA: Este paso puede invertise en orden, ya que se puede comenzar realizando el

atributo	tipo	longitud	mascara
id_estado	int	4	
nom_estado	varchar	25	
id_zona	int	a	
nom_zona	varchar	25	
nom_categoria	varchar	20	
id_elemento	int	8	
id_grafico	int	4	
id_mapa	varchar	15	
longitud	int	7	___-__-__
latitud	int	7	___-__-__
tipo	char	1	

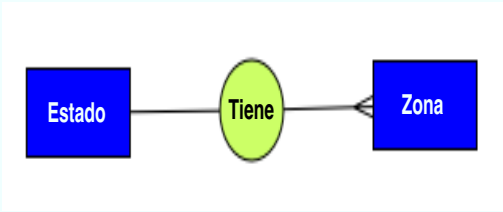
Tabla 5.2: Atributos y sus características.

diagrama y posteriormente capturarse los atributos que han de asociarse a las entidades e interrelaciones.

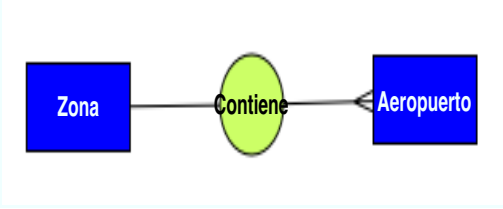
5.1.2. Diagrama EVE

El tercer paso es identificar los vículos entre las entidades e interrelaciones expresados en roles:

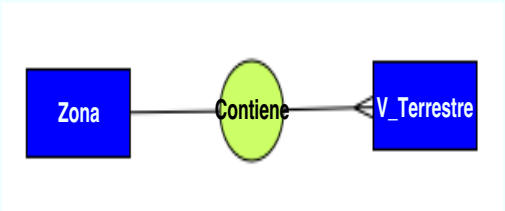
Un ESTADO tiene una o más de una ZONA geográfica.



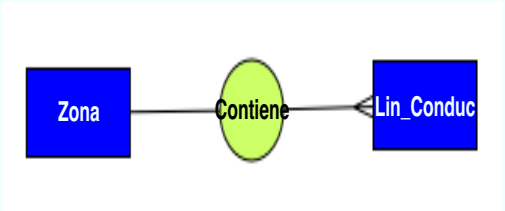
Una ZONA contiene mas de un AEROPUERTO.



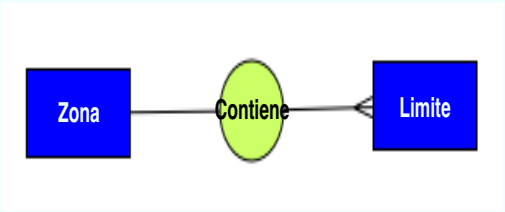
Una ZONA contiene mas de una VIA TERRESTRE.



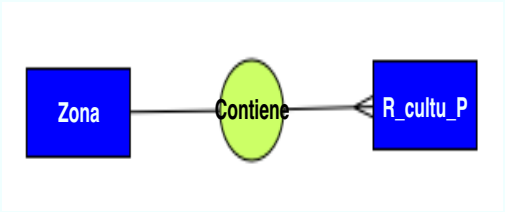
Una ZONA contiene mas de una LINEA DE CONDUCCION



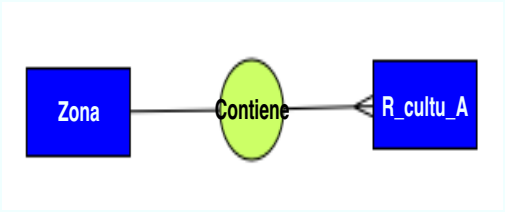
Una ZONA contiene mas de un LIMITE



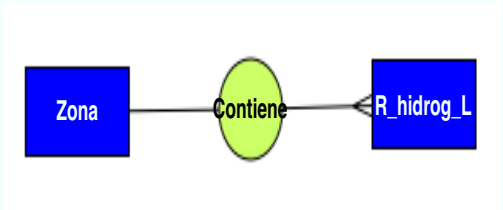
Una ZONA contiene mas de un RASGO CULTURAL_PUNTO



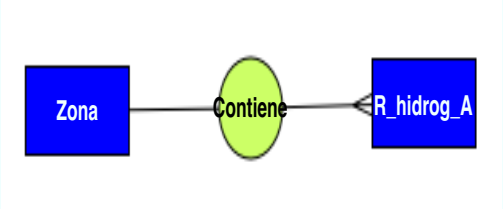
Una ZONA contiene mas de un RASGO CULTURAL_AREA



Una ZONA contiene mas de un RASGO HIDROGRAFICO_LINEA



Una ZONA contiene mas de un RASGO HIDROGRAFICO_AREA



Fusionamos los roles y elaboramos el diagrama de la figura 5.2.

Por último, a cada entidad le asociamos un nombre, sus atributos llave y sus atributos descriptores obteniendo como resultado las siguientes estructuras:

Estado(id_estado, nom_estado)
Zona(id_zona,nom_zona)
Aeropuerto(id_elemento, id_grafico, id_mapa, nom_categoria,longitud, latitud, tipo)
V_Terres(id_elemento, id_grafico, id_mapa, nom_categoria,longitud, latitud, tipo)
Limite(id_elemento, id_grafico, id_mapa, nom_categoria,longitud, latitud, tipo)
R_ cult_P(id_elemento, id_grafico, id_mapa, nom_categoria,longitud, latitud, tipo)
R_ cult_A(id_elemento, id_grafico, id_mapa, nom_categoria,longitud, latitud, tipo)
R_hidro_L(id_elemento, id_grafico, id_mapa, nom_categoria,longitud, latitud, tipo)
R_hidro_P(id_elemento, id_grafico, id_mapa, nom_categoria,longitud, latitud, tipo)

Estas asignaciones se realizan en la pantalla que aparce al seleccionar la opción "Actualiza entidades" del menú *tools*. Como ejemplo se muestran los identificadores y atributos de las relaciones Estado, Zona y Aeropuerto (figs. 5.3, 5.4 y 5.5).

Para bases de datos de sistemas de información geográfica diseñados por capas, todas las entidades relacionadas con alguna de las capas o categorías, contienen los mismos atributos ya que almacenan objetos similares, es decir objetos con propiedades con las cuales se les puede ubicar dentro de una superficie terrestre simulada por una imagen digitalizada de cartas topográficas. Sin embargo, no se recomienda crear una sola tabla y adicionar un atributo que indique la capa con la que se esta trabajando, bebido a que el número de objetos posibles de ubicar dentro de las superficies espaciales crece a gran

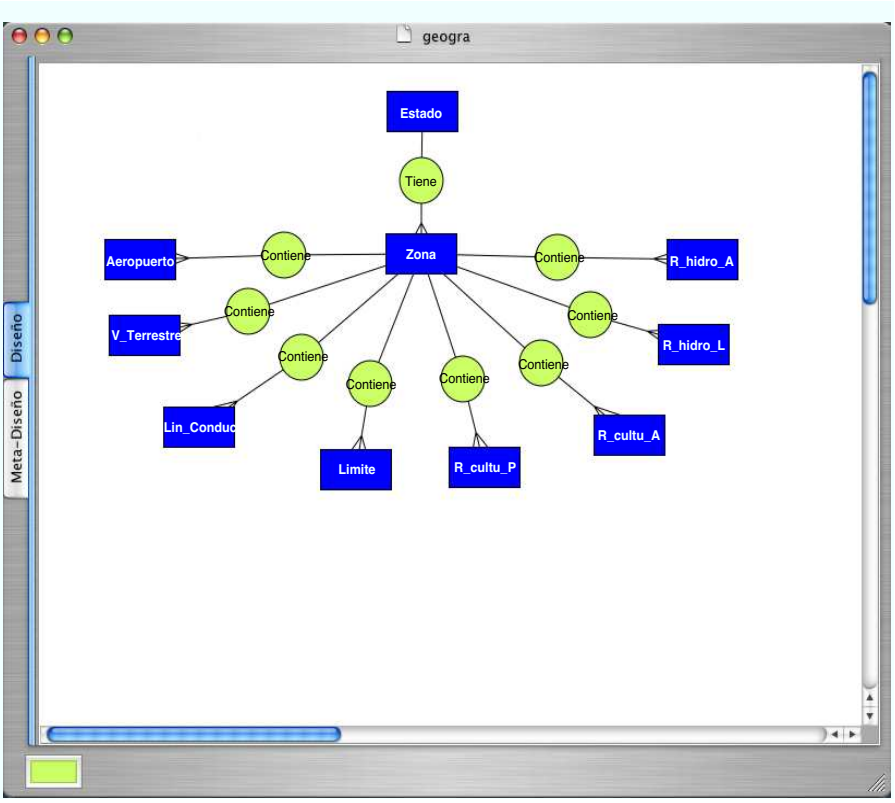


Figura 5.2: Diagrama EVE para el SIG.

escala por cada capa digitalizada. Por tanto las consultas consumirían mayor cantidad de tiempo en ejecutase.

5.2. Diseño de una base de datos para el control de un almacen

En este segundo caso de estudio se emplearan los dos niveles de diseño que la herramienta EVEMac ofrece.

5.2.1. Nivel de Metadiseño

En este nivel de diseño se requiere elaborar el diseño de una base de datos que apoyará a las tareas de la administración y control del proyecto de un sistema de información que automatice la administración de un almacen.

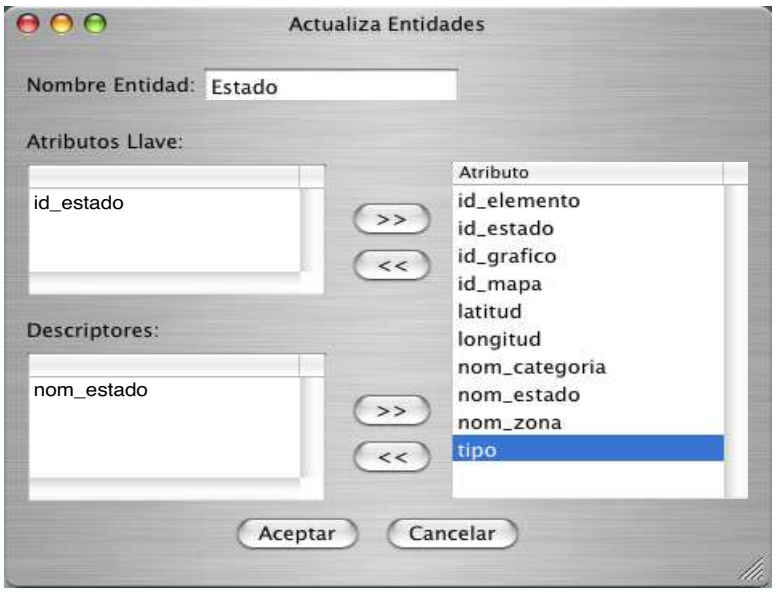


Figura 5.3: Asignación de atributos a la entidad "Estado".

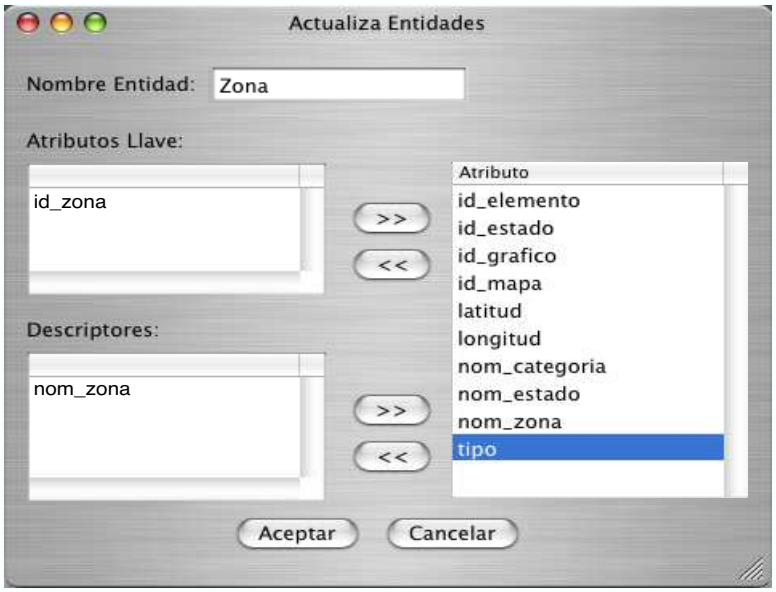


Figura 5.4: Asignación de atributos a la entidad "Zona".

Descripción Preliminar

En la empresa X se desarrollará un sistema de información para el control de un almacén, para lograr que éste proyecto se finalice con éxito en los término de tiempo. coste y calidad es nesario una planificación inicial del proyecto asi como un control adecuado

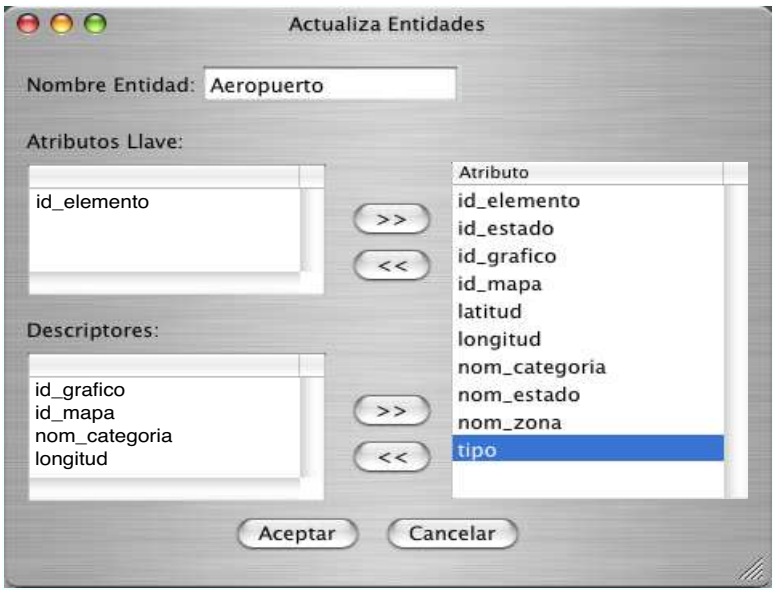


Figura 5.5: Asignación de atributos a la entidad "Aeropuerto".

para lograr ese objetivo.

El encargado del proyecto de software requiere tener un control acerca de las personas involucradas, de las tareas por realizar, del tiempo asignado por tarea y avances del proyecto.

La base de datos diseñada en este metadiseño además deberá almacenar información que ayuden a responder a los siguientes cuestionamientos.

1. Quien elaboró la tarea A?
2. Cuantas horas efectivas de trabajo comprobadas tiene el empleado B?
3. Cual es el porcentaje de avance del proyecto?
4. Qué personal ha cumplido satisfactoriamente las tareas asignadas?
5. Cuales son las tareas que se han realizado a una fecha?
6. A que persona se le asignó la tarea B?
7. Cuántos defectos se encontraron en una tarea realizada y de que tipo son?

Identificación de entidades, atributos e interrelaciones.

Lo primero que realzaremos es analizar la descripción del problema para obtener los elementos a modelar: entidades, atributos e interrelaciones (Tab 5.3)

Entidades	Atributos	Atributos	Relaciones
Proyecto	id_proyecto	id_recurso	asignado a
Tareas	nom_proyecto	nom_recurso	tiene
Recursos	fec_ini	direcc_recurso	encontrados
Defectos	fec_fin	telefono_recurso	
	duracion_hrs	hrs_jornadasem	
	rol_recurso	id_tarea	
	desc_tarea	duracion_hrs	
	complejidad	prioridad	
	id_defecto	id_tipodefecto	
	hrs_correcion	fec_deteccion	
	fec_correccion		

Tabla 5.3: Identificación de entidades, atributos e interrelacione

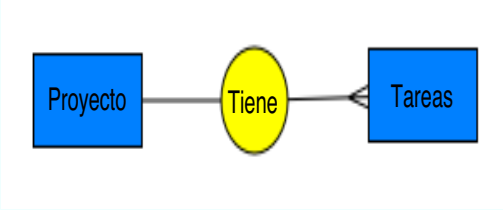
Diccionario de atributos

Ahora se especificarán las características de los atributos. La tabla 5.4 muestra todos los datos que deben capturarse. Esa captura se realiza en la pantalla que aparece al seleccionar la opción "Diccionario de Atributos" del menú *tools*.

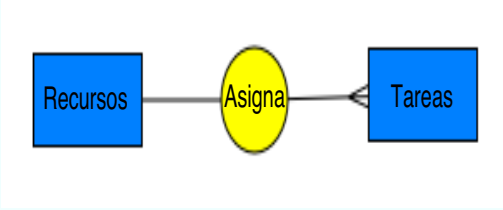
Diagrama EVE

El tercer paso es identificar los víclulos entre las entidades e interrelaciones expresados en roles:

Un PROYECTO tiene una o más de una TAREA.



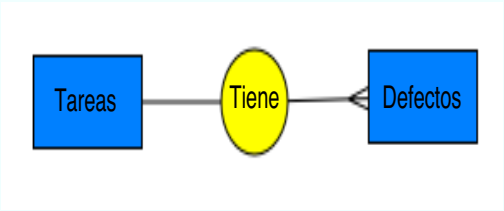
Un RECUROS es asignado a una o mas de una TAREA



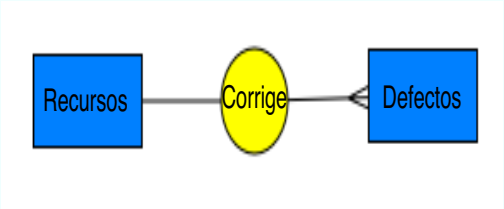
atributo	**tipo**	**longitud**	**mascara**
id_proyecto	char	7	
nom_proyecto	varchar	20	
fec_ini	date	8	__/__/____
fec_fin	date	8	__/__/____
duracion_hrs	int	3	
id_recurso	int	4	
nom_recurso	varchar	40	
direcc_recurso	varchar	60	
telefono_recurso	char	20	
hrs_jornadasem	int	2	
rol_recurso	varchar	25	
id_tarea	int	3	
desc_tarea	varchar	30	
duracion_hrs	int	3	
complejidad	int	1	
prioridad	int	1	
id_defecto	int	3	
id_tipodefecto	int	1	
hrs_correcion	int	3	
fec_deteccion	date	8	__/__/____
fec_correccion	date	8	__/__/____

Tabla 5.4: Atributos y sus características.

Una TAREA puede tener más de un DEFECTO



Un RECURSO es asignado para corregir a uno o mas DEFECTOS



Fusionamos los roles y elaboramos el diagrama de la figura 5.6.

Para finalizar el modelo conceptual asociamos a cada entidad y relación sus atributos llave y atritutos descriptores.

Proyecto(id_proyecto, nom_proyecto, fec_ini, fec_fin, duracion_hrs)
Tarea(id_tarea, desc_tarea, duracion_hrs, complejidad, prioridad)
Recursos(id_recurso, nom_recurso, direcc_recurso, telefono_recurso, hrs_jornadasem, rol_recurso)
Defecto(id_defecto, id_tipodefecto, hrs_correcion, fec_deteccion, fec_correccion)

5.2.2. Nivel de diseño

Este es el segundo nivel de diseño en nuestra metodología donde realizaremos la base de datos para el sistema de automatización del control de un almacen,

Descripción preliminar

En la fabrica X se producen tres productos: A, B, Y C. Cada producto se elabora en una división, y cada una de esta tiene un jefe de división La fabrica cuenta con otros departamentos: comercialización, compras y almacen de refacciones. El jefe del almacen general se encarga de despachar las solicitudes de refacciones de las divisiones y cuando no se tenga en existencia las refacciones, de solicitar los pedidos al departamente de compras. Las refacciones tienen un identificador, una descripción, una marca, una ubicación por anaquel y un número de unidades en existencia, un precio de compra y un punto de reorden.. Algunas refacciones, las de mayor uso, tienen refacciones sustitutas. Los jefes de

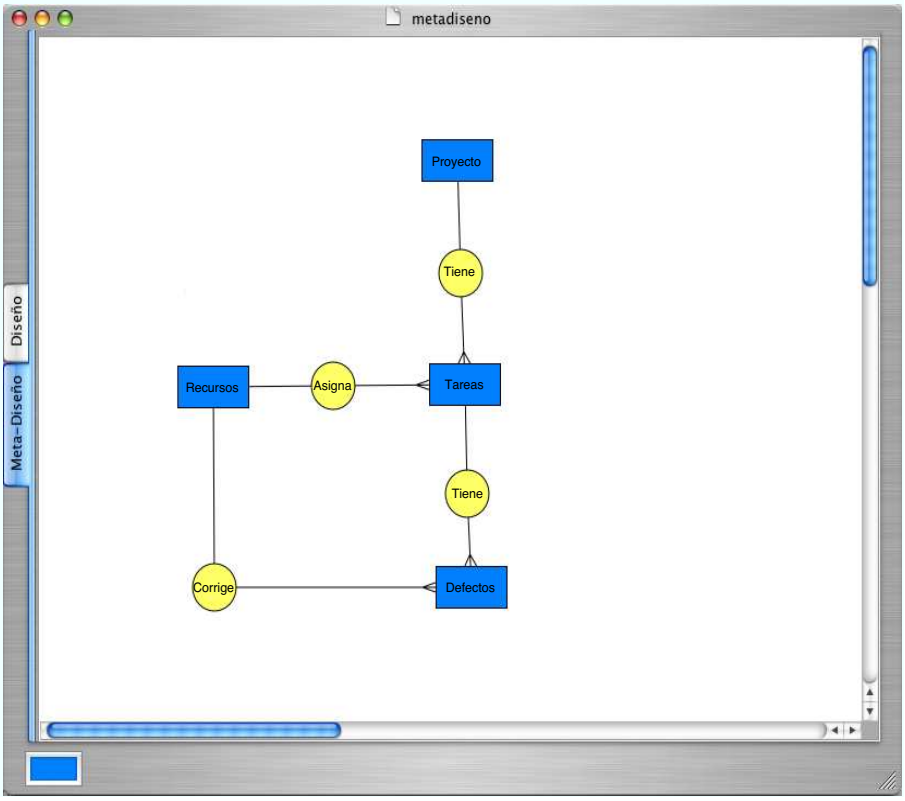


Figura 5.6: Diagrama EVE del metadiseño

división al día levantan una solicitud de refacciones. En la solicitud se indica el número de solicitud, la fecha, el nombre de la división, el nombre del jefe, las refacciones y sus cantidades solicitadas. Los empleados de almacén en el mismo día surte las solicitudes de refacciones. Con las refacciones despachadas entregan un reporte de despacho de la solicitud en dónde se señala si se ha despachado parcialmente o totalmente la solicitud, las piezas q se estan despachando y la cantidad, el nombre del despachador, el nombre del que recoge las refacciones. Como trabajo adicional deben descontar las existencias de refacciones que despacharon. Cuando el almacén no tenga una pieza original despacha una pieza sustituta. Cuando no exista la refacción sustituta, los almacenistas deben realizar una solicitud de pedido dirigido al departamento de compras. A este tipo de solicitud se le clasifica como urgente existe otro tipo de solicitud de pedido, clasificado como normal. en el que se solicitan refacciones que estén en o sobre el punto de reorden. En las solicitudes de pedidos se indica el número de pedido, la fecha, el tipo de solicitud, las refacciones y cantidades exactas. Cuando la solicitud es urgente debe acompañarse con los nombres de las divisiones que solicitan la refacción.

Se plantea la automatización de un sistema para automatizar la administración de este almacén general, para lo cual requeriremos una base de datos que sirva de soporte.

Entidades	Atributos	Atributos	Relaciones
división	id_division	teléfono	administra
solicitud	fec_solicitud	num_solicitude	especifica
refaccion	num_refacción	num_pedido	despacha
almacen	descripción	marca	surte
pedido	ubicación	unidades	requiere
urgente	max	min	hace
jefe_div	nom_division	cant_pedido	gestiona
normal	promedio	fec_pedido	
salida	almacenista	tipo_pedido	
	num_salida	fec_salida	
	observaciones	cant_salida	
	cant_solicitud		

Tabla 5.5: Identificación de entidades, atributos e interrelaciones

Identificación de entidades, atributos e interrelaciones

Analizando la descripción del problema, se identifican las entidades, los atributos y las interrelaciones mostrados en la tabla 5.5.

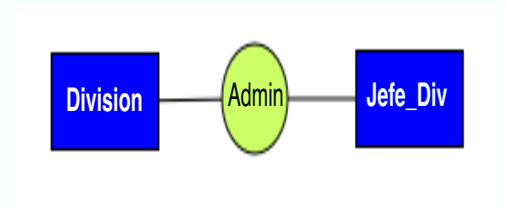
Diccionario de atributos

Ahora especificamos las caractéristicas de los atributos y las capturamos en el editor del diccionario de atributos, En la tabla 5.6 se muestra el contenido que debe de almacenarse.

Diagrama EVE

Ahora procedemos a identificar los vínculos entre las entidades e interrelaciones expresados en roles.

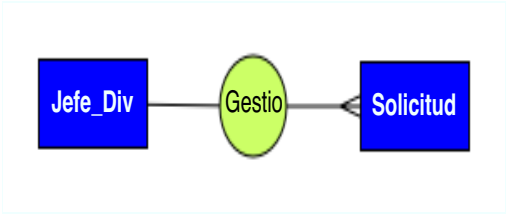
Un JEFE DE DIVISION administra una DIVISION.



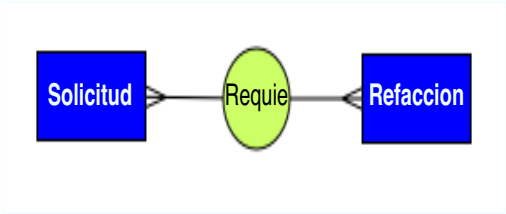
atributo	tipo	longitud	mascara
id_division	int	2	
nom_division	varchar	15	
telefono	char	12	
fax	char	12	
id_jefe	int	2	
nom_jefe	varchar	30	
fec_solicitud	date	8	__/__/____
num_solicitud	int	6	
num_refaccion	int	5	
num_pedido	int	4	
descripcion	varchar	60	
marca	varchar	15	
ubicacion	char	15	
unidades	int	3	
maximo	int	3	
minimo	int	3	
promedio	int	3	
fec_pedido	date	8	__/__/____
almacenista	varchar	30	
tipo_pedido	char	1	
num_salida	int	4	
fec_salida	date	8	__/__/____
observaciones	varchar	256	
cant_salida	int	3	
cant_solicitud	int	3	
cant_pedido	int	3	

Tabla 5.6: Atributos y sus características.

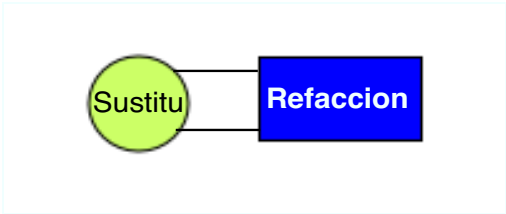
Un JEFE DE DIVISION gestiona más de una SOLICITUD.



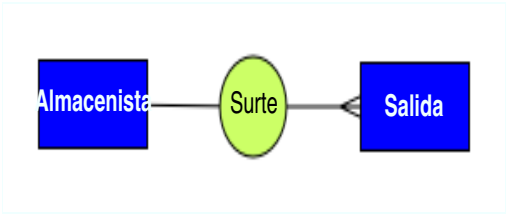
Una SOLICITUD requiere más de una REFACCION y una REFACCION es requerida en más de una SOLICITUD.



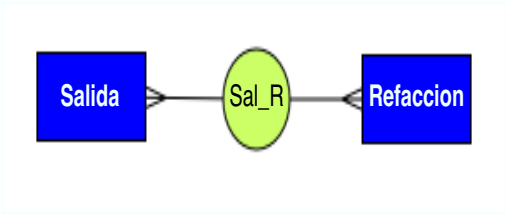
Una REFACCION sustituye a otra refacción.



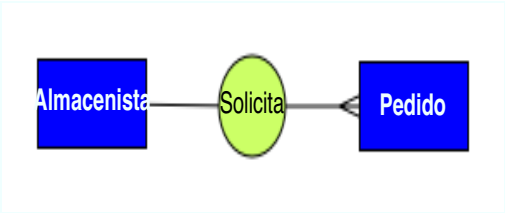
Un ALMACENISTA despacha más de una SALIDA.



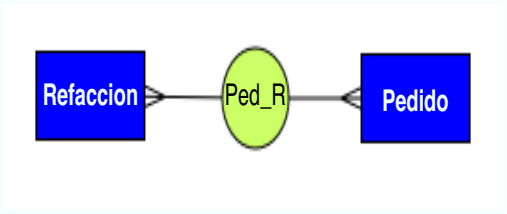
Una SALIDA requiere más de una REFACCION y una REFACCION es requerida en más de una SALIDA.



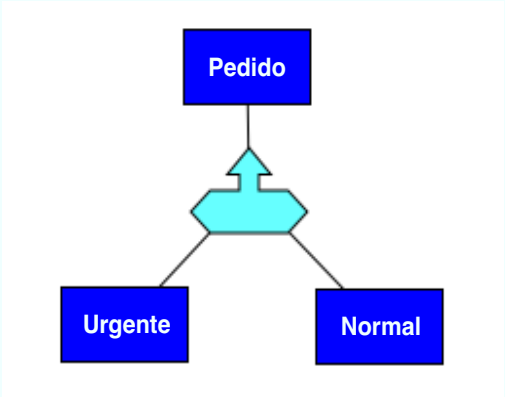
Un ALMACENISTA solicita más de un PEDIDO.



Un PEDIDO requiere más de una REFACCION, una REFACCION es requerida en más de un PEDIDO.



Un PEDIDO es la genealozación de un pedido URGENTE y un pedido NORMAL.



Fusionamos los roles y elaboramos el diagrama que se muestra en la figura 5.7.

De cada entidad asociamos su nombre, atributos llaves y atributos descriptores.

Division(id_division, nom_division, telefono, fax)
Jefe_Div(id_jefe, nom_jefe)
Solicitud(num_solicitud,fec_solicitud)
Refaccion(num_refaccion, descripcion, marca, ubicacion, unidades, maximo, minimo)
Pedido(num_pedido, fec_pedido, tipo_pedido)
Almacenista(id_almacenista, nom_almacenista)
Salida(num_salida, fec_salida)

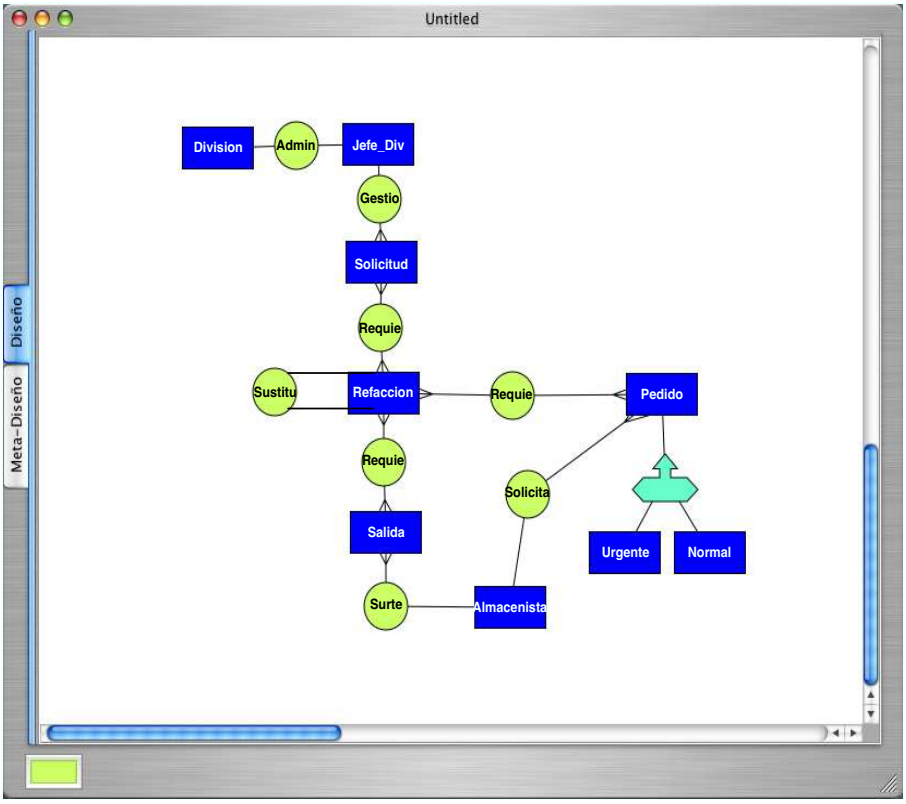


Figura 5.7: Diagrama EVE de la base de datos del sistema para la administración de un almacen.

Urgente(id_division)
Normal(observaciones)

Las interrelaciones:

Requie(cant_solicitud)
Ped_R(cant_pedido)
Sal_R(cant_salida)

Capítulo 6

Conclusiones y trabajos a futuro

La principal contribución de nuestro trabajo de tesis es haber diseñado y desarrollado una metodología y una herramienta que incorpora un marco de trabajo que permite realizar el diseño de bases de datos en dos niveles de diseño.
El meta-diseño. Nivel en el cual podemos definir los actores y los recursos involucrados en un proyecto creando de esta manera un esquema conceptual de una base de metadatos. El nivel de diseño. En este nivel definimos todos los conceptos relativos a la implantación de un sistema. Por tanto se obtienen el esquema conceptual de una base de datos del negocio.

Las características que contiene nuestra herramienta son las siguientes:

- Los niveles de meta-diseño y diseño se trabajan bajo el mismo lenguaje visual lo que da como ventaja que el diseñador no necesita conocimientos extras más que los mencionados por nuestra metodología para lograr obtener una base de datos de meta-diseño.
- Los diagramas EVE se realizan bajo una edición dirigida por sintaxis, lo que permite obtener diagramas correctos sintácticamente.
- En los diagramas EVE pueden emplearse colores, lo que permitirá a los diseñadores categorizar los íconos diagramados para darle un mayor significado semántico, además de agilizar la revisión y lectura de los diagramas.
- La estructura con que fué diseñada nuestra herramienta permiten facilmente hacer extensiones al modelo entidad vínculo extendido que se propone, Permitiendo de esta manera adoptar cambios surgidos con el paso del tiempo.

Nuestra contribución cobra importancia al considerar que es un subsistema de una herramienta CASE que pretende automatizar la mayor parte de las etapas del ciclo de vida de un sistema. Esta herramienta es llamada CADBD Diseño de Bases de Datos Asistido por Computadora, proyecto del área de investigación de Lenguajes Visuales y Bases de Datos dirigida por el Dr. Sergio V. Chapa Vergara.

Las extensiones próximas de nuestro trabajo son la implementación del modelo lógico y el modelo físico de la base de datos, lo que implica las siguientes tareas:

1. Traduccir de los diagramas EVE al modelo relacional.
2. Aplicar el algoritomo de normalización de sistésis de Berntein considerando las optimizaciones propuestas por Dierich J. y Milton J [1].
3. Crear la base de datos física en un SMBD.

Bibliografía

[1] Piattini Mario. Adoración de Miguel Castaño. *Concepción y Diseño de Bases de Datos del modelo ER al modelo relacional.* Addison Wesley Iberoamericana, 1993.

[2] Piattini Mario. Adoración de Miguel Castaño. *Fundamentos y modelos de bases de datos.* Alfaomega, 2000.

[3] Stefano Ceri Carlo Batini and Shamkant B. Navathe. *Conceptual Database Design. And entity relationship approach.* ADDISON-WESLEY, 1994.

[4] Alday Echevarría Pedro Enrique. *Diseño de bases de datos con EVEX entidad vínculo estendido para Xwindows.* Tesis(M.C.) Centro de Investigación y de Estudios Avanzados del IPN. Departamento de Ingeniería Eléctrica. Sección Computación, 1997.

[5] Apple Computer Inc. *Learning Cocoa.* O'Reilly, 2001.

[6] Ted G. Lewis Margaret M. Burnett, Adele Goldberg. *Visual Object-Oriented Programing Concepts and Environments.* Prentice Hall, 1995.

[7] Chen Peter. *The Entity-Relationship Model-Toward Unified View of Data.* ACM Transactions on Database System, 1976.

[8] Shamkant B.Ñavathe. Ramez Elmasri. *Sistemas de bases de datos: Conceptos Fundamentales.* Addison-Wesley Iberoamericana, 1997.

[9] Chapa Vergara Sergio. *Herramienta para consulta y captura basadas en el descriptor de archivos.* Reporte técnico No. 15, serie amarilla investigación, departamento de ingenieria Electrica, CINVESTAV-IPN, febrero, 1985.

[10] Chapa Vergara Sergio. *Programación automática a partir de descriptores de flujo de información.* Tesis de doctorado del Centro de Investigación y de Estudios Avanzados del IPN. Departamento de Ingeniería Eléctrica. Sección Computación, 1991.

[11] Bernhard Thalheim. *Entity-Relationship Modeling. Foundation of Database Technology.* Springer, Germany, 1998.

[12] Frank Swiaczny. Tomas Ott. *Time-Integrativ Geographic Information Systems. Managment and Analysis of Spation Temporal Data.* Springer, 2001.

[13] URL. *http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/index.html*.

[14] URL. *http://www.cs.cinvestav.mx/BDChapa/seck/basico3.html*.

[15] URL. *http://www.macdevcenter.com/pub/a/mac/2002/04/19/cocoa.html*.

[16] URL. *http://www.monografias.com/trabajos/gis/gis.shtml*.

[17] Brian Wilson. *Systems: Concepts, methodologies, and applications.* John Wiley and Sons, 1984.