TESIS
II 200
P7
v.2

UNIVERSIDAD CATÓLICA ANDRÉS BELLO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA INFORMÁTICA

DISEÑO E IMPLANTACIÓN DE UNA HERRAMIENTA PARA MANEJO DE MODELOS DE DATOS BASADA EN UN METAMODELO INDEPENDIENTE DE LA PLATAFORMA DESARROLLO DE APLICACIONES. (APÉNDICES)

TRABAJO ESPECIAL DE GRADO

presentado ante la

UNIVERSIDAD CATÓLICA ANDRÉS BELLO

como parte de los requisitos para optar al título de

INGENIERO EN INFORMÁTICA

REALIZADO POR Pinto Suárez, María Alejandra.
Martinez Marcano, Luis Santiago.

PROFESOR GUIA Cardoso Militao, Lúcia.

FECHA Caracas, Mayo de 2005.

APÉNDICE A. ENCUESTA REALIZADAS PARA LA CAPTURA DE REQUISITOS DEL SISTEMA

A.1 Modelos de Encuestas.

A.1.1 Modelo de Encuesta #1

Responde a las siguientes preguntas marcando solo una opción de las presentadas a continuación.

Una herramienta CASE se utiliza para:

____Diseñar bases de datos únicamente.

____Automatizar el desarrollo de software.

____Hacer diagramas en general.

____ No tengo conocimientos.

Para construir un Modelo de Datos utilizando una herramienta CASE (Ej. Power Designer) es necesario conocimientos acerca de:

____ Lenguaje PL/SQL.

____ Lenguaje Java.

____Diagramación UML.

____ Conocimientos en diseño de bases de datos.

____Lenguaje C++.

____ODBC o JDBC.

Marca con una X la(s) funcionalidad(es) que conoces que posee una herramienta CASE.

____Modelo de Procesos de Negocios.

____Modelos conceptuales de datos.

____Modelos Libres.

____Modelos orientados a objetos.

____Modelos Físicos de Datos.

____Modelos XML.

____Manipulación de Vistas y Datos en tiempo real.

____Definición de Repositorios de datos.

____Ejecutar Comandos SQL.

____Ejecutar Instrucciones de creación

____Diagramas de Clases.
____Diagramas de Actividades.
____Diagramas de Casos de USOS.
____Diagramas de Objetos.
____Diagramas de Colaboración.
____Diagramas de Secuencia.
____Diagramas de Componentes.
____Diagramas de Implantación.
____Ingeniería Reversa de los modelos mediante ODBC/JDBC.
____Conexión con Bases de Datos.
____Conexión con Bases de Datos.
____Manipulación de Vistas y Datos en tiempo real.
____Definición de Repositorios de datos.
____Ejecutar Comandos SQL.
____Ejecutar Instrucciones de creación de base de datos.
____Generar Reportes de Diccionario de Datos.
____Generar Modelos Físicos a partir de modelos conceptuales de datos.
____Chequear errores existentes en modelos de datos.
____Generación de código fuente a partir de modelos conceptuales.

Marca con una X las funcionalidades que utilizas regularmente en una herramienta CASE.

____Modelo de Procesos de Negocios.
____Modelos conceptuales de datos.
____Modelos Libres.
____Modelos orientados a objetos.
____Modelos Físicos de Datos.
____Modelos XML.
____Diagramas de Clases.
____Diagramas de Actividades.
____Diagramas de Casos de USOS.
____Diagramas de Objetos.
____Diagramas de Colaboración.
____Diagramas de Secuencia.
____Diagramas de Componentes.
____Diagramas de Implantación.
____Ingeniería Reversa de los modelos mediante ODBC/JDBC.
____Conexión con Bases de Datos.
____Manipulación de Vistas y Datos en tiempo real.
____Definición de Repositorios de datos.
____Ejecutar Comandos SQL.
____Ejecutar Instrucciones de creación de base de datos.
____Generar Reportes de Diccionario de Datos.
____Generar Modelos Físicos a partir de modelos conceptuales de datos.
____Chequear errores existentes en modelos de datos.
____Generación de código fuente a partir de modelos conceptuales.

A.1.2. Modelo de Encuesta #2

Basado en las experiencias de la utilización de herramientas CASE (Ej. Power Designer) Marca con una X las afirmaciones con las que estés de acuerdo.

____ Me desagrada no poder mantener activas múltiples versiones de los modelos que realizo.

____ Sus funciones facilitan el trabajo considerablemente.

____ Debo realizar un número considerable de operaciones para poder cumplir con los objetivos.

____ Su interfaz es sencilla y agradable.

____ Su interfaz es complicada.

____ Debo poseer conocimientos en leguaje SQL para poder utilizarlas.

____ Poseen una útil ayuda para la consulta de dudas y procedimientos.

____ La ayuda es pobre y difícil de consultar.

____ Solo las utilizo como herramientas de graficación.

____ Permiten disminuir los tiempos de codificación y prueba.

____ Usualmente las utilizo para generar código y su documentación.

Marca con una X .Si pudieses modificar Power Designer, te gustaría que:

____ El Menú fuese más sencillo en su navegación.

____ Independiente de la plataforma de desarrollo de aplicaciones.

____ Mantener múltiples versiones activas de los modelos y la compatibilidad entre ellas.

____ Poseer un API de Funciones que me permita la manipulación de los datos directamente desde la herramienta.

____ Permitir la flexibilidad de crear y alterar el modelo de datos que se desea implantar, sin necesidad de poseer conocimientos previos de lenguajes de manipulación de datos (SQL).

____ Proveer el manejo de seguridad de datos, en cuanto a la creación de perfiles de usuario administración de estos en cada perfil.

____ Definir un lenguaje sencillo de recuperación de datos.

____ Que fuese útil y sencillo para el desarrollo de modelos conceptuales de datos.

En tu opinión, ¿Que aspectos te gustaría que tuviese una Herramienta CASE?

_____

_____

A.2. Resultados Encuesta

A.2.1. Resultados Encuesta #1

A.2.1.1. Grupo de Muestra #1 (Sistemas de Bases de Datos II)

Tamaño de la muestra: 30 Personas.

Pregunta 1.

Una herramienta CASE se utiliza para:

Opción	Descripción	% de Respuesta
1	Diseñar bases de datos únicamente.	13.33
2	**Automatizar el desarrollo de software.(Correcta)**	30
3	Hacer diagramas en general.	56.66
4	No tengo conocimientos.	0

Tabla 9. Resultados pregunta #1, encuesta 1 (Muestra 1).
Fuente: Elaborado por los autores.

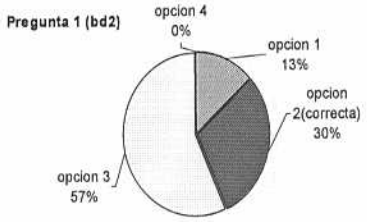


Figura 32. Diagrama pregunta #1, encuesta 1 (Muestra 1).
Fuente: Elaborado por los autores

Pregunta 2.

Para construir un Modelo de Datos utilizando una herramienta CASE (Ej. Power Designer) es necesario conocimientos acerca de:

Opción	Descripción	% de Respuesta
1	Lenguaje PL/SQL.	0
2	Lenguaje Java.	0
3	Diagramación UML.	10
4	**Conocimientos en diseño de bases de datos.(Correcta)**	90
5	Lenguaje C++.	0
6	ODBC o JDBC.	0

Tabla 10. Resultados pregunta #2, encuesta 1 (Muestra 1).
Fuente: Elaborado por los autores.

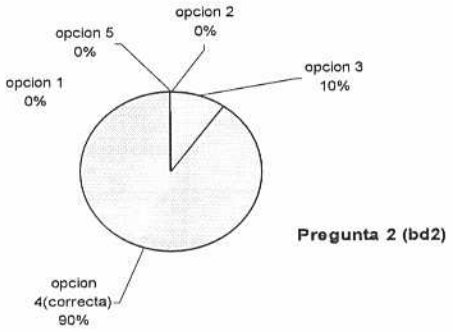


Figura 33. Diagrama pregunta #2,encuesta 1 (Muestra 1).
Fuente: Elaborado por los autores

Pregunta 3.

Marca con una X la(s) funcionalidad(es) que conoces que posee una herramienta CASE.

Opción	Descripción	% de Respuesta
1	Modelo de Procesos de Negocios.	16.67
2	Modelos conceptuales de datos.	63.3
3	Modelos Libres.	10
4	Modelos orientados a objetos.	63.3
5	Modelos Físicos de Datos.	33.3
6	Modelos XML.	23.3
7	Diagramas de Clases.	46.6
8	Diagramas de Actividades.	23.3
9	Diagramas de Casos de USOS.	36.6
10	Diagramas de Objetos.	33.3
11	Diagramas de Colaboración.	3.3
12	Diagramas de Secuencia.	3.3
13	Diagramas de Componentes.	16.6
14	Diagramas de Implantación.	13.33
15	Ingeniería Reversa de los modelos mediante ODBC/JDBC.	26.6
16	Conexión con Bases de Datos.	10
17	Manipulación de Vistas y Datos en tiempo real.	3.3
18	Definición de Repositorios de datos.	0
19	Ejecutar Comandos SQL.	20
20	Ejecutar Instrucciones de creación de base de datos	23.3
21	Generar Reportes de Diccionario de Datos	13.3
22	Generar Modelos Físicos a partir de modelos conceptuales de datos.	43.3
23	Chequear errores existentes en modelos de datos.	26.6
24	Generación de código fuente a partir de modelos conceptuales.	46.6

Tabla 11. Resultados pregunta #3, encuesta 1 (Muestra 1).
Fuente: Elaborado por los autores.

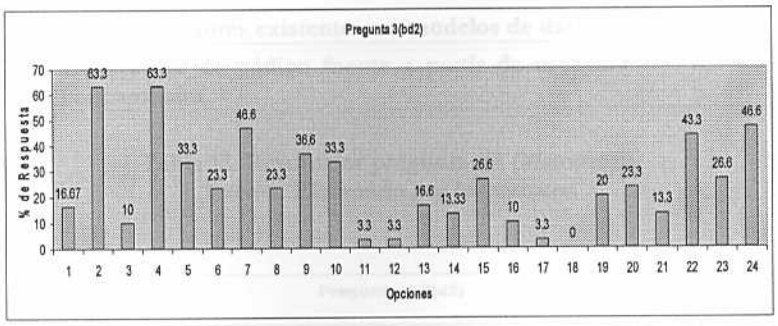


Figura 34. Resultados pregunta #3,encuesta 1 (Muestra 1).
Fuente: Elaborado por los autores.

Pregunta 4.

Marca con una X las funcionalidades que utilizas regularmente en una herramienta CASE.

Opción	Descripción	% de Respuesta
1	Modelo de Procesos de Negocios.	3.33
2	Modelos conceptuales de datos.	66.6
3	Modelos Libres.	6.66
4	Modelos orientados a objetos.	20
5	Modelos Físicos de Datos.	13.33
6	Modelos XML.	0
7	Diagramas de Clases.	30
8	Diagramas de Actividades.	3.33
9	Diagramas de Casos de USOS.	20
10	Diagramas de Objetos.	26.66
11	Diagramas de Colaboración.	0
12	Diagramas de Secuencia.	0
13	Diagramas de Componentes.	6.66
14	Diagramas de Implantación.	3.33
15	Ingeniería Reversa de los modelos mediante ODBC/JDBC.	3.33
16	Conexión con Bases de Datos.	3.33
17	Manipulación de Vistas y Datos en tiempo real.	0
18	Definición de Repositorios de datos.	0
19	Ejecutar Comandos SQL.	13.33
20	Ejecutar Instrucciones de creación de base de datos	13.33
21	Generar Reportes de Diccionario de Datos	3.33
22	Generar Modelos Físicos a partir de modelos conceptuales de datos.	26.66

23	Chequear errores existentes en modelos de datos.	6.66
24	Generación de código fuente a partir de modelos conceptuales.	13.33

Tabla 12. Resultados pregunta #4 (Muestra 1).
Fuente: Elaborado por los autores.

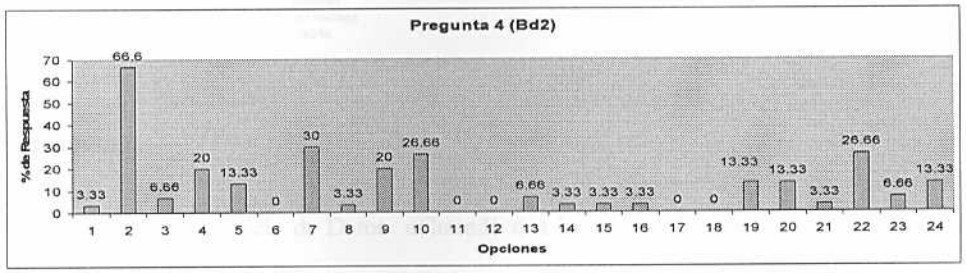


Figura 35. Resultados pregunta #4, encuesta 1 (Muestra 1).
Fuente: Elaborado por los autores

A.2.1.2. Grupo de Muestra #2 (Desarrollo del Software)

Tamaño de la muestra: 9 Personas.

Pregunta 1.

Una herramienta CASE se utiliza para:

Opción	Descripción	% de Respuesta
1	Diseñar bases de datos únicamente.	22.2
2	**Automatizar el desarrollo de software.(Correcta)**	33.3
3	Hacer diagramas en general.	44.4
4	No tengo conocimientos.	0

Tabla 13. Resultados pregunta #1, encuesta 1 (Muestra 2).
Fuente: Elaborado por los autores.

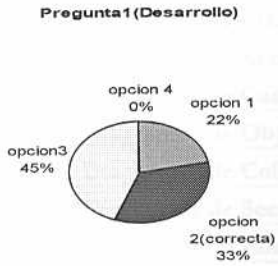


Figura 36. Resultados pregunta #4, encuesta 1 (Muestra 2).
Fuente: Elaborado por los autores

Pregunta 2.

Para construir un Modelo de Datos utilizando una herramienta CASE (Ej. Power Designer) es necesario conocimientos acerca de:

Opción	Descripción	% de Respuesta
1	Lenguaje PL/SQL.	0
2	Lenguaje Java.	0
3	Diagramación UML.	11.11
4	Conocimientos en diseño de bases de datos.	88.88
5	Lenguaje C++.	0
6	ODBC o JDBC.	0

Tabla 14. Resultados pregunta #2 ,encuesta 1 (Muestra 2).
Fuente: Elaborado por los autores.

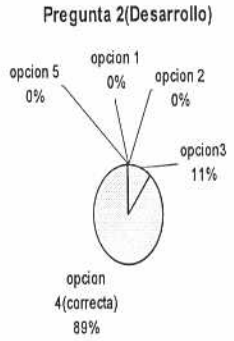


Figura 37. Diagrama pregunta #2, encuesta 1 (Muestra 2).
Fuente: Elaborado por los autores

Pregunta 3.

Marca con una X la(s) funcionalidad(es) que conoces que posee una herramienta CASE.

Opción	Descripción	% de Respuesta
1	**Modelo de Procesos de Negocios.**	**33.3**
2	**Modelos conceptuales de datos.**	**77.77**
3	**Modelos Libres.**	**55.5**
4	**Modelos orientados a objetos.**	**100**
5	**Modelos Físicos de Datos.**	**55.5**

6	Modelos XML.	11.11
7	Diagramas de Clases.	88.88
8	Diagramas de Actividades.	55.55
9	Diagramas de Casos de USOS.	77.77
10	Diagramas de Objetos.	66.66
11	Diagramas de Colaboración.	22.22
12	Diagramas de Secuencia.	33.33
13	Diagramas de Componentes.	22.22
14	Diagramas de Implantación.	11.11
15	Ingeniería Reversa de los modelos mediante ODBC/JDBC.	44.44
16	Conexión con Bases de Datos.	33.33
17	Manipulación de Vistas y Datos en tiempo real.	0
18	Definición de Repositorios de datos.	22.22
19	Ejecutar Comandos SQL.	44.44
20	Ejecutar Instrucciones de creación de base de datos	55.55
21	Generar Reportes de Diccionario de Datos	77.77
22	Generar Modelos Físicos a partir de modelos conceptuales de datos.	88.88
23	Chequear errores existentes en modelos de datos.	55.55
24	Generación de código fuente a partir de modelos conceptuales.	100

Tabla 15. Resultados pregunta #3, encuesta 1 (Muestra 2).
Fuente: Elaborado por los autores.

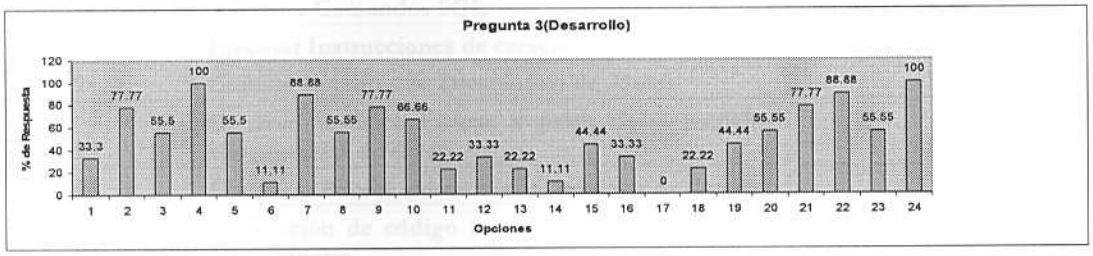


Figura 38. Resultados pregunta #3,encuesta 1 (Muestra 2).
Fuente: Elaborado por los autores

Pregunta 4.

Marca con una X las funcionalidades que utilizas regularmente en una herramienta CASE.

Opción	Descripción	% de Respuesta
1	Modelo de Procesos de Negocios.	33.3
2	Modelos conceptuales de datos.	77.77
3	Modelos Libres.	55.5
4	Modelos orientados a objetos.	100
5	Modelos Físicos de Datos.	55.5
6	Modelos XML.	11.11
7	Diagramas de Clases.	88.88
8	Diagramas de Actividades.	55.55
9	Diagramas de Casos de USOS.	77.77
10	Diagramas de Objetos.	66.66
11	Diagramas de Colaboración.	22.22
12	Diagramas de Secuencia.	33.33
13	Diagramas de Componentes.	22.22
14	Diagramas de Implantación.	11.11
15	Ingeniería Reversa de los modelos mediante ODBC/JDBC.	44.44
16	Conexión con Bases de Datos.	33.33
17	Manipulación de Vistas y Datos en tiempo real.	0
18	Definición de Repositorios de datos.	22.22
19	Ejecutar Comandos SQL.	44.44
20	Ejecutar Instrucciones de creación de base de datos	55.55
21	Generar Reportes de Diccionario de Datos	77.77
22	Generar Modelos Físicos a partir de modelos conceptuales de datos.	88.88
23	Chequear errores existentes en modelos de datos.	55.55
24	Generación de código fuente a partir de modelos conceptuales.	100

Tabla 16. Resultados pregunta #4,encuesta 1 (Muestra 2).
Fuente: Elaborado por los autores.

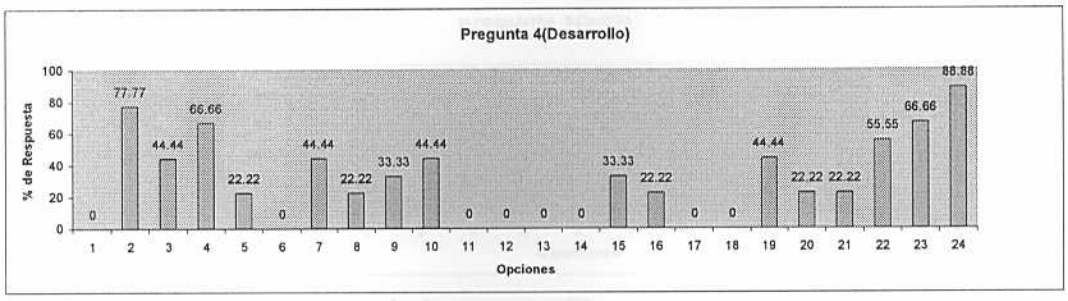


Figura 39. Resultados pregunta #4, encuesta 1 (Muestra 2).
Fuente: Elaborado por los autores

A.2.2. Resultados Encuesta #2

A.2.2.1. Grupo de Muestra #1 (Sistemas de Bases de Datos II)

Tamaño de la muestra: 30 Personas.

Pregunta 1,

Basado en las experiencias de la utilización de herramientas CASE (Ej. Power Designer).Marca con una X las afirmaciones con las que estés de acuerdo.

1. Me desagrada no poder mantener activas múltiples versiones de los modelos que realizo.
2. Sus funciones facilitan el trabajo considerablemente.
3. Debo realizar un número considerable de operaciones para poder cumplir con los objetivos.
4. Su interfaz es sencilla y agradable.
5. Su interfaz es complicada.
6. Debo poseer conocimientos en leguaje SQL para poder utilizarlas.
7. Poseen una útil ayuda para la consulta de dudas y procedimientos.
8. La ayuda es pobre y difícil de consultar.
9. Solo las utilizo como herramientas de graficación.
10. Permiten disminuir los tiempos de codificación y prueba.
11. Usualmente las utilizo para generar código y su documentación.

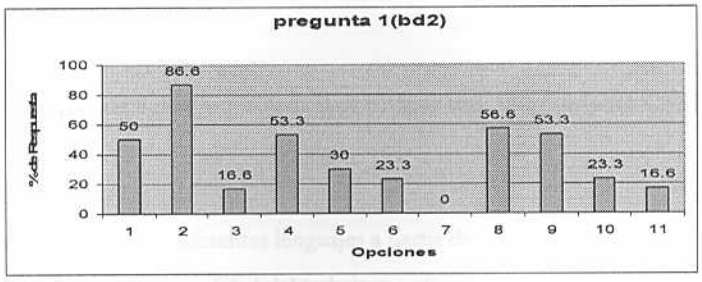


Figura 40. Resultados pregunta #1, encuesta 2 (Muestra 1).
Fuente: Elaborado por los autores

<u>Pregunta 2,</u>

Marca con una X .Si pudieses modificar Power Designer, te gustaría que:

1. El Menú fuese más sencillo en su navegación.
2. Independiente de la plataforma de desarrollo de aplicaciones.
3. Mantener múltiples versiones activas de los modelos y la compatibilidad entre ellas.
4. Poseer un API de Funciones que me permita la manipulación de los datos directamente desde la Herramienta.
5. Permitir la flexibilidad de crear y alterar el modelo de datos que se desea Implantar, sin necesidad de poseer conocimientos previos de lenguajes de manipulación de datos (SQL).
6. Proveer el manejo de seguridad de datos, en cuanto a la creación de perfiles de usuario y Administración de estos en cada perfil.
7. Definir un lenguaje sencillo de recuperación de datos.
8. Que fuese útil y sencillo para el desarrollo de modelos conceptuales de datos.

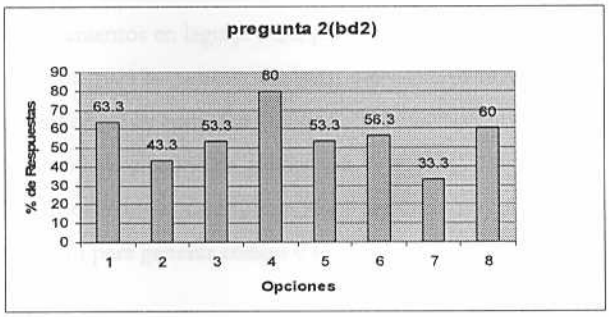


Figura 41. Resultados pregunta #2, encuesta 2 (Muestra 1).
Fuente: Elaborado por los autores

Pregunta 3,

En tu opinión, que aspectos te gustaría que tuviese una Herramienta CASE? (Opinión)

- Generación de código en diferentes lenguajes a partir de modelos.
- Sencillez, facilidad de uso y calidad del trabajo a realizar.
- Una herramienta case debe proporcionar ayuda para el modelado, prototipado entre otras.
- Que no tuviese problemas de versiones y que fuera en un lenguaje más flexible.
- Interfaz Agradable y fácil.

A.2.2.2. Grupo de Muestra #2 (Desarrollo del Software)

Tamaño de la muestra: 9 Personas.

Pregunta 1,

Basado en las experiencias de la utilización de herramientas CASE (Ej. Power Designer) Marca con una X las afirmaciones con las que estés de acuerdo.

1. Me desagrada no poder mantener activas múltiples versiones de los modelos que realizo.
2. Sus funciones facilitan el trabajo considerablemente.
3. Debo realizar un número considerable de operaciones para poder cumplir con los objetivos.
4. Su interfaz es sencilla y agradable.
5. Su interfaz es complicada.
6. Debo poseer conocimientos en leguaje SQL para poder utilizarlas.
7. Poseen una útil ayuda para la consulta de dudas y procedimientos.
8. La ayuda es pobre y difícil de consultar.
9. Solo las utilizo como herramientas de graficación.
10. Permiten disminuir los tiempos de codificación y prueba.
11. Usualmente las utilizo para generar código y su documentación.

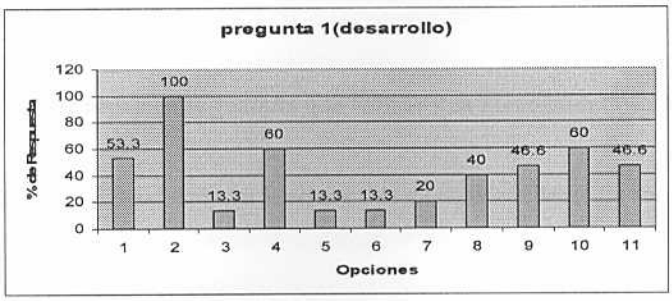


Figura 42. Resultados pregunta #1, encuesta 2 (Muestra 2).
Fuente: Elaborado por los autores

<u>Pregunta 2,</u>

Marca con una X .Si pudieses modificar Power Designer, te gustaría que:

1. El Menú fuese más sencillo en su navegación.
2. Independiente de la plataforma de desarrollo de aplicaciones.
3. Mantener múltiples versiones activas de los modelos y la compatibilidad entre ellas.
4. Poseer un API de Funciones que me permita la manipulación de los datos directamente desde la Herramienta.
5. Permitir la flexibilidad de crear y alterar el modelo de datos que se desea Implantar, sin necesidad de poseer conocimientos previos de lenguajes de manipulación de datos (SQL).
6. Proveer el manejo de seguridad de datos, en cuanto a la creación de perfiles de usuario y Administración de estos en cada perfil.
7. Definir un lenguaje sencillo de recuperación de datos.
8. Que fuese útil y sencillo para el desarrollo de modelos conceptuales de datos.

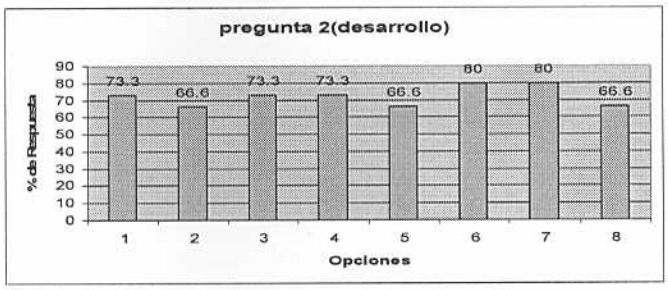


Figura 43. Resultados pregunta #2, encuesta 2 (Muestra 2).
Fuente: Elaborado por los autores

Pregunta 3,

En tu opinión, que aspectos te gustaría que tuviese una Herramienta CASE? (Opinión)

- Permitir realizar modificaciones a su interfaz y a su modo de modelaje sin aumentar la dificultad de uso.
- Generación de código.
- Una base de datos amplia sobre componentes Gamma.
- Una forma agradable de buscar en ellas.
- Pruebas de los scripts antes de ser usados.
- Que sea multiplataforma.
- Interfaz amigable.
- Fácil manejo.
- Codificación a través de modelos sencillos.
- Facilidad en su manejo y ahorro en tiempo de programación.

APÉNDICE B. TRADUCTORES E INTÉRPRETES.

B.1 Concepto de Traductor.

Un traductor se define como un programa que traduce o convierte desde un texto o programa escrito en un lenguaje fuente hasta un texto o programa equivalente escrito en un lenguaje destino produciendo, en algunos casos, mensajes de error. Los traductores engloban tanto a los compiladores como a los intérpretes.

La figura 44 muestra el esquema básico que compone a un compilador/intérprete. (Gálvez et al. 2005).

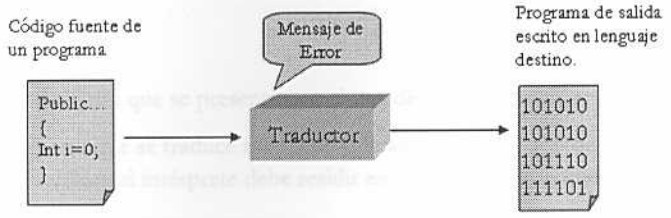


Figura 44. Esquema básico de un compilador/interprete.
Fuente: Modificada por los autores. Del trabajo de (Gálvez et al., 2005).

B.1.2.Tipos de traductores

Entre los tipos de traductores más importantes existentes en la actualidad se pueden diferenciar:

a) Traductores del idioma.

Traducen de un idioma dado a otro, como por ejemplo del inglés al español. Este tipo de traductores posee multitud de problemas, a saber:

- Necesidad de inteligencia artificial y problema de las frases hechas.
- Difícil formalización en la especificación del significado de las palabras.
- Cambio del sentido de las palabras según el contexto.

En general, los resultados más satisfactorios en la traducción del lenguaje natural se han producido sobre subconjuntos restringidos del lenguaje. Y aún más, sobre subconjuntos en los que hay muy poco margen de ambigüedad en la interpretación de los textos: discursos jurídicos, documentación técnica, etc.

b) Compiladores.

Es aquel traductor que tiene como entrada una sentencia en lenguaje formal y como salida tiene un archivo ejecutable, es decir, realiza una traducción de un código de alto nivel a código máquina.

c) Intérpretes.

Es análogo a un compilador, solo que la salida es una ejecución. El programa de entrada se reconoce y ejecuta a la vez. No se produce un resultado físico (código máquina) sino lógico (una ejecución).

Entre los inconvenientes que se presentan en el uso de los intérpretes se pueden citar:

- Al ejecutar a la vez que se traduce no puede aplicarse un alto grado de optimización.
- Durante la ejecución, el intérprete debe residir en memoria, por lo que consumen más recursos.

Además de que la traducción optimiza el programa acercándolo a la máquina, los lenguajes interpretados tienen la característica de que permiten construir programas que se pueden modificar a sí mismos.

Algunos lenguajes intentan aunar las ventajas de los compiladores e intérpretes y evitar sus desventajas; dichos lenguaje se conocen como pseudointerpretados. En estos, el código fuente pasa por un pseudocompilador que genera un pseudoejecutable. Para ejecutar este pseudoejecutable se le hace pasar por un motor de ejecución que lo interpreta de manera relativamente eficiente.

Esto tiene la ventaja de la portabilidad, ya que el pseudoejecutable es independiente de la máquina en que vaya a ejecutarse, y basta con que en dicha máquina se disponga del motor de ejecución apropiado para poder interpretar cualquier pseudoejecutable. El ejemplo actual más conocido lo constituye el lenguaje Java. (Ver figura 45).

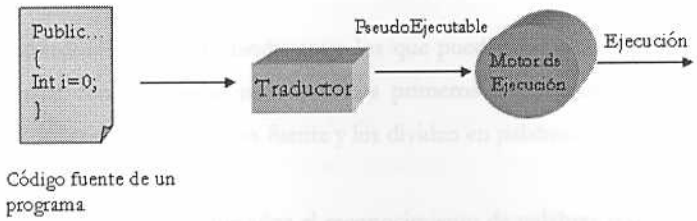


Figura 45.Esquema de traducción/ejecución de un programa interpretado. Fuente: Modificada por los autores. Del trabajo de (Gálvez et al., 2005).

d) Preprocesadores.

Permiten modificar el programa fuente antes de la verdadera compilación. Hacen uso de macroinstrucciones y directivas de compilación. Los preprocesadores suelen actuar de manera transparente para el programador, pudiendo incluso considerarse que son una fase preliminar del compilador.

e) Intérpretes de comandos.

Un intérprete de comandos traduce sentencias simples a invocaciones a programas de una librería. Se utilizan especialmente en los sistemas operativos.

f) Ensambladores y macroensambladores.

Son los pioneros de los compiladores, ya que en los principios de la informática, los programas se escribían directamente en código máquina, y el primer paso hacia los lenguajes de alto nivel lo constituyen los ensambladores. En lenguaje ensamblador se establece una relación biunívoca entre cada instrucción y una palabra mnemotécnica, de manera que el usuario escribe los programas haciendo uso de los mnemotécnicos, y el ensamblador se encarga de traducirlo a código máquina puro. De esta manera, los ensambladores suelen producir directamente código ejecutable en lugar de producir archivos objeto.

g) Metacompilador.

Un metacompilador es un compilador de compiladores. Se trata de un programa que acepta como entrada la descripción de un lenguaje y produce el compilador de dicho lenguaje (Gálvez et al., 2005). Hoy por hoy no existen metacompiladores completos, pero sí parciales en los que se acepta como entrada una gramática de un lenguaje y se genera un autómata que reconoce cualquier sentencia del lenguaje. A este autómata podemos añadirle código para completar el resto del compilador. Algunos de los metacompiladores conocidos son: YACC, JavaCC, JLex.

Los metacompiladores se suelen dividir entre los que pueden trabajar con gramáticas de contexto libre y los que trabajan con gramáticas regulares. Los primeros se dedican a reconocer la sintaxis del lenguaje y los segundos separan los archivos fuente y los dividen en palabras.

JavaCC es un metacompilador que aúna el reconocimiento de palabras reservadas y la aceptación o rechazo de sentencias de un lenguaje. Este produce código Java y admite descripciones de lenguajes expresadas en notación BNF (Backus-Naur Form).

B.1.3. Estructura de un traductor.

Un traductor divide su labor en dos etapas: una que analiza la entrada y genera estructuras intermedias y otra que sintetiza la salida a partir de dichas estructuras. (Ver figura 46).

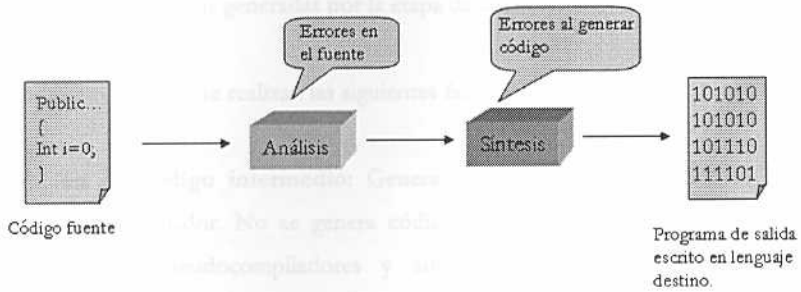


Figura 46.Esquema por etapas de un traductor.
Fuente: Modificada por los autores. Del trabajo de (Gálvez et al., 2005).

B.1.3.1. Etapa de Análisis.

Básicamente los objetivos de la etapa de análisis son:

a) Controlar la corrección del programa fuente.

b) Generar las estructuras necesarias para comenzar la etapa de síntesis.

Para llevar esto a cabo, la etapa de análisis consta de las siguientes fases:

- **Análisis lexicográfico:** Divide el programa fuente en los componentes básicos del lenguaje a compilar. Cada componente básico es una subsecuencia de caracteres del programa fuente, y pertenece a una categoría gramatical: números, identificadores de usuario (variables, constantes, tipos, nombres de procedimientos), palabras reservadas, signos de puntuación.

- **Análisis sintáctico:** Comprueba que la estructura de los componentes básicos sea correcta según las reglas gramaticales del lenguaje que se compila.

- **Análisis semántico:** Comprueba que el programa fuente respeta las directrices del lenguaje que se compila (todo lo relacionado con el significado): chequeo de tipos, rangos de valores, existencia de variables.

Cualquiera de estas tres fases puede emitir mensajes de error derivados de fallos cometidos por el programador en la redacción de los textos fuente. Mientras más errores controle un compilador, menos problemas dará un programa en tiempo de ejecución. (Gálvez et al., 2005).

B.1.3.2. Etapa de Síntesis.

La etapa de síntesis consiste en construir el programa objeto deseado (equivalente semánticamente al fuente) a partir de las estructuras generadas por la etapa de análisis.

Para lograr dicho objetivo se realizan las siguientes fases fundamentales:

- **Generación de código intermedio:** Genera un código independiente de la máquina muy parecido al ensamblador. No se genera código máquina directamente porque esto facilita la construcción de pseudocompiladores y además se provee la optimización de código independientemente del microprocesador.

- **Generación del código máquina.** Crea un bloque de código máquina ejecutable, así como los bloques necesarios destinados a contener los datos.

- **Fase de optimización.** La optimización puede realizarse sobre el código intermedio, sobre el código máquina, o sobre ambos. (Gálvez et al., 2005)

APÉNDICE C. UML

C.1 UML.

UML (Unified Modeling Language) o Lenguaje Unificado de Modelado, es un lenguaje que permite modelar, construir y documentar los elementos que forman un sistema software orientado a objetos. Esta notación ha sido ampliamente aceptada debido a que incorpora las principales ventajas de cada uno de los métodos particulares en los que se basa (Ferré et al., 2004).

Con UML se fusiona la notación de estas técnicas para formar una herramienta compartida entre todos los ingenieros software que trabajan en el desarrollo orientado a objetos. Uno de los objetivos principales de la creación de UML era posibilitar el intercambio de modelos entre las distintas herramientas CASE orientadas a objetos del mercado.

C.2 Diagramas UML.

Un Diagrama representa a un sistema software desde una perspectiva específica. Los tipos de diagramas existentes en la notación UML, permite visualizar cada una de las partes del sistema desde cualquier punto de vista, debido a que muestra su comportamiento tanto estático como dinámico.

C.2.1 Elementos Comunes a los diagramas.

C.2.1.1 Notas.

Una nota sirve para añadir cualquier tipo de comentario a un diagrama o a un elemento de un diagrama. Es un modo de indicar información en un formato libre, cuando la notación del diagrama en cuestión, no nos permite expresar dicha información de manera adecuada (Ver figura 47).

Figura 47. Nota dentro de un diagrama de clases UML.
Fuente: Elaborado por los autores. Adaptada de (Ferré et al., 2004).

C.2.1.2 Dependencias.

La relación de dependencia entre dos elementos de un diagrama significa que un cambio en el elemento destino puede implicar un cambio en el elemento origen (por tanto, si cambia el elemento destino habría que revisar el elemento origen). Una dependencia se representa por medio de una línea de trazo discontinuo entre los dos elementos con una flecha en su extremo. El elemento dependiente es el origen de la flecha y el elemento del que depende es el destino (Figura 48).

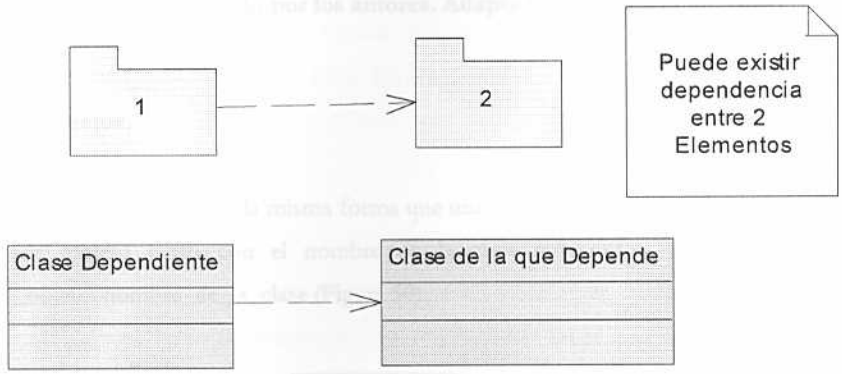


Figura 48. Dependencias dentro de un diagrama de clases UML.
Fuente: Elaborado por los autores Adaptada de (Ferré et al., 2004).

C.2.2. Diagramas de estructura de Clases.

Un diagrama de estructura de clases, muestra el conjunto de clases y objetos importantes que hacen parte de un sistema, junto con las relaciones existentes entre estas clases y objetos. Muestra de una manera estática la estructura de información del sistema y la visibilidad que tiene cada una de las clases, dada por sus relaciones con las demás en el modelo.

Los componentes de un diagrama de estructura de clases, se describirán en los puntos subsiguientes.

C.2.2.1 Clases.

Una clase se representa mediante una caja subdividida en tres partes: En la superior se muestra el nombre de la clase, en la media los atributos y en la inferior las operaciones. Una clase puede representarse de forma esquemática, con los atributos y operaciones suprimidos, siendo entonces tan solo un rectángulo con el nombre de la clase (Figura 49).

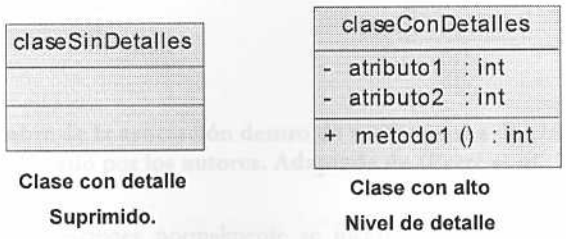


Figura 49.Distintas notaciones para las clases dentro de los diagramas de clases UML. Fuente: Elaborado por los autores. Adaptada de (Ferré et al., 2004).

C.2.2.2 Objetos.

Un objeto se representa de la misma forma que una clase. En el compartimiento superior aparecen el nombre del objeto junto con el nombre de la clase subrayados, según la siguiente sintaxis: nombre_del_objeto: nombre_de_la_clase (Figura 50).

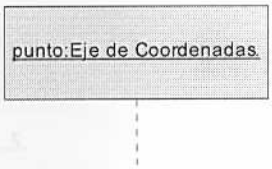


Figura 50. Objeto dentro de un diagrama de clases UML. Fuente: Elaborado por los autores. Adaptada de (Ferré et al., 2004).

C.2.2.3 Asociaciones.

Las asociaciones entre dos clases se representan mediante una línea que las une. La línea puede tener una serie de elementos gráficos que expresan características particulares de la asociación. A continuación se verán los más importantes de entre dichos elementos gráficos.

- **Nombre de la Asociación y Dirección.**

El nombre de la asociación es opcional y se muestra como un texto que está próximo a la línea. Se puede añadir una pequeña flecha que indique la dirección en la cual leer el nombre de la asociación. En el ejemplo (Figura 51) se puede leer la asociación como "Director manda sobre Empleado".

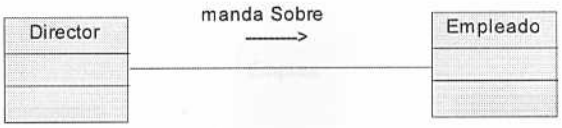


Figura 51. Nombre de la asociación dentro de un diagrama de clases UML.
Fuente: Elaborado por los autores. Adaptada de (Ferré et al., 2004).

Los nombres de las asociaciones normalmente se incluyen en los modelos para aumentar la legibilidad. Sin embargo, en ocasiones pueden hacer demasiado abundante la información que se presenta, con el consiguiente riesgo de saturación. En ese caso se puede suprimir el nombre de las asociaciones consideradas como suficientemente conocidas. En las asociaciones de tipo agregación y de herencia no se suele poner el nombre.

- **Multiplicidad.**

La multiplicidad es una restricción que se pone a una asociación, que limita el número de instancias de una clase que pueden tener esa asociación con una instancia de la otra clase. Puede expresarse de las siguientes formas (Figura 52):

- Con un número fijo: 1.
- Con un intervalo de valores: 2...5.
- Con un rango en el cual uno de los extremos es un asterisco. Significa que es un intervalo abierto.
- Con una combinación de elementos como los anteriores separados por comas: 1, 3..5, 7..*.
- Con un asterisco: * . En este caso indica que puede tomar cualquier valor (cero o más).

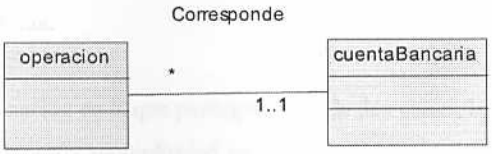


Figura 52. Multiplicidad de una asociación dentro de un diagrama UML.
Fuente: Elaborado por los autores. Adaptada de (Ferré et al., 2004).

- **Roles.**

Para indicar el papel que juega una clase en una asociación se puede especificar un nombre de rol. Se representa en el extremo de la asociación junto a la clase que desempeña dicho rol (Figura 53).

Figura 53. Rol dentro de asociación dentro de un diagrama UML.
Fuente: Elaborado por los autores. Adaptada de (Ferré et al., 2004).

- **Agregación.**

El símbolo de agregación es un diamante colocado en el extremo en el que está la clase que representa el "todo" (Figura 54).

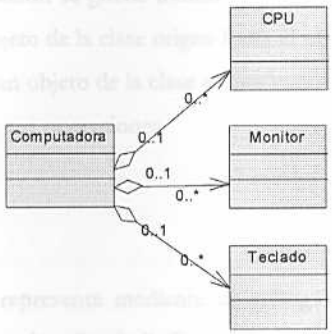


Figura 54. Agregación dentro de un diagrama UML.
Fuente: Elaborado por los autores. Adaptada de (Ferré et al., 2004).

- **Asociaciones N-Arias.**

En el caso de una asociación en la que participan más de dos clases, las clases se unen con una línea a un diamante central. Si se muestra multiplicidad en un rol, representa el número potencial de tuplas de instancias en la asociación cuando el resto de los N-1 valores están fijos (Figura 55).

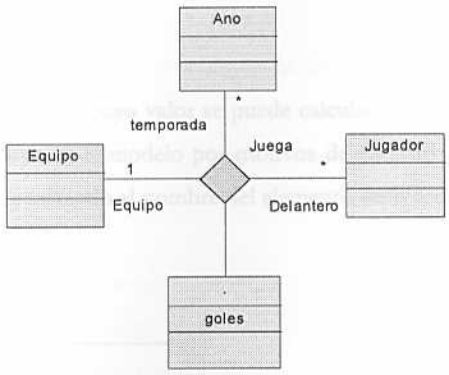


Figura 55. Asociación N-Aria dentro de un diagrama UML.
Fuente: Elaborado por los autores. Adaptada de (Ferré et al., 2004).

- **Navegabilidad.**

En un extremo de una asociación se puede indicar la navegabilidad mediante una flecha. Significa que es posible "navegar" desde el objeto de la clase origen hasta el objeto de la clase destino. Se trata de un concepto de diseño, que indica que un objeto de la clase origen conoce al (los) objeto(s) de la clase destino, y por tanto puede llamar a alguna de sus operaciones.

C.2.2.4 Herencia.

La relación de herencia se representa mediante un triángulo en el extremo de la relación que corresponde a la clase más general o clase "padre". Si se tiene una relación de herencia con varias clases subordinadas, pero en un diagrama concreto no se quieren poner todas, esto se representa mediante puntos suspensivos (Figura 56).

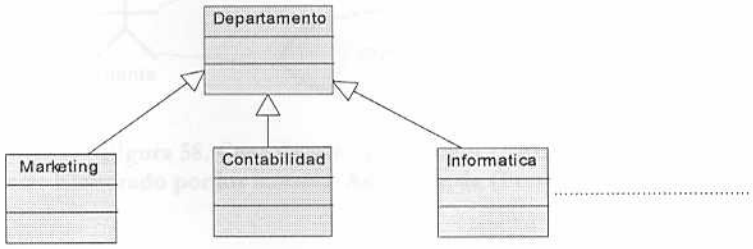


Figura 56. Herencia dentro de un diagrama UML.
Fuente: Elaborado por los autores. Adaptada de (Ferré et al., 2004).

C.2.2.5 Elementos Derivados.

Un elemento derivado es aquel cuyo valor se puede calcular a partir de otros elementos presentes en el modelo, pero que se incluye en el modelo por motivos de claridad o como decisión de diseño. Se representa con una barra "/" precediendo al nombre del elemento derivado (Figura 57).

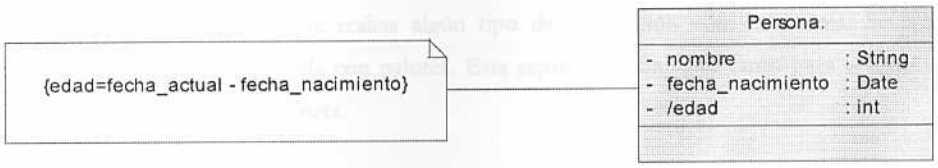


Figura 57. Elemento Derivado dentro de un diagrama UML.
Fuente: Elaborado por los autores. Adaptada de (Ferré et al., 2004).

C.2.3 Diagrama de Casos de Uso

Un Diagrama de Casos de Uso muestra la relación entre los actores y los casos de uso del sistema. Representa la funcionalidad que ofrece el sistema en lo que se refiere a su interacción externa. En el diagrama de casos de uso se representa también el sistema como una caja rectangular con el nombre en su interior. Los casos de uso están en el interior de la caja del sistema, y los actores fuera, y cada actor está unido a los casos de uso en los que participa mediante una línea (Figura 58).

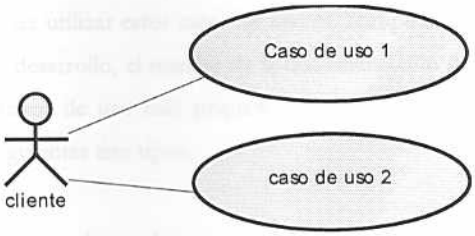


Figura 58. Caso de uso en notación UML.
Fuente: Elaborado por los autores. Adaptada de (Ferré et al., 2004).

C.2.3.1 Elementos.

Los elementos que pueden aparecer en un Diagrama de Casos de Uso son: actores, casos de uso y relaciones entre casos de uso.

- **Actores.**

Un actor es algo con comportamiento, como una persona (identificada por un rol), un sistema informatizado u organización, y que realiza algún tipo de interacción con el sistema. Se representa mediante una figura humana dibujada con palotes. Esta representación sirve tanto para actores que son personas como para otro tipo de actores.

- **Casos de Uso.**

Un caso de uso es una descripción de la secuencia de interacciones que se producen entre un actor y el sistema, cuando el actor usa el sistema para llevar a cabo una tarea específica. Expresa una unidad coherente de funcionalidad, y se representa en el Diagrama de Casos de Uso mediante una elipse con el nombre del caso de uso en su interior. El nombre del caso de uso debe reflejar la tarea específica que el actor desea llevar a cabo usando el sistema.

- **Relaciones entre Casos de Uso.**

Un caso de uso, en principio, debería describir una tarea que tiene un sentido completo para el usuario. Sin embargo, hay ocasiones en las que es útil describir una interacción con un alcance menor como caso de uso. La razón para utilizar estos casos de uso no completos en algunos casos, es mejorar la comunicación en el equipo de desarrollo, el manejo de la documentación de casos de uso. Para el caso de que queramos utilizar estos casos de uso más pequeños, las relaciones entre estos y los casos de uso ordinarios pueden ser de los siguientes tres tipos:

- Incluye (<>): Un caso de uso base incorpora explícitamente a otro caso de uso en un lugar especificado en dicho caso base. Se suele utilizar para encapsular un comportamiento parcial común a varios casos de uso (Figura 59).

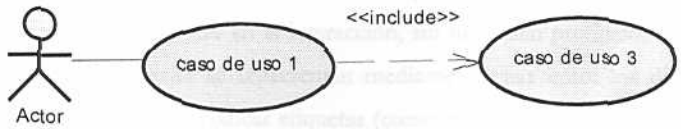


Figura 59. Relación Include dentro de un Caso de Uso UML.
Fuente: Elaborado por los autores. Adaptada de (Ferré et al., 2004).

- Extiende (<>): Cuando un caso de uso base tiene ciertos puntos (puntos de extensión) en los cuales, dependiendo de ciertos criterios, se va a realizar una interacción adicional. El caso de uso que extiende describe un comportamiento opcional del sistema (a diferencia de la relación incluye que se da siempre que se realiza la interacción descrita), ver figura 60.

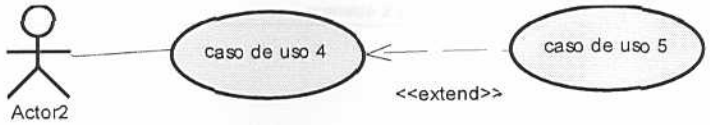


Figura 60. Relación Extend dentro de un Caso de Uso UML.
Fuente: Elaborado por los autores. Adaptada de (Ferré et al., 2004).

Ambos tipos de relación se representan como una dependencia etiquetada con el estereotipo correspondiente (<> o <>), de tal forma que la flecha indique el sentido en el que debe leerse la etiqueta. Junto a la etiqueta <> se puede detallar el/los puntos de extensión del caso de uso base en los que se aplica la extensión.

- Generalización: Cuando un caso de uso definido de forma abstracta se particulariza por medio de otro caso de uso más específico. Se representa por una línea continua entre los dos casos de uso, con el triángulo que simboliza generalización en UML (usado también para denotar la herencia entre clases) pegado al extremo del caso de uso más general. Al igual que en la herencia entre clases, el caso de uso hijo hereda las asociaciones y características del caso de uso padre. El caso de uso padre se trata de un caso de uso abstracto, que no está definido completamente. Este tipo de relación se utiliza mucho menos que las dos anteriores.

C.2.4 Diagrama de Secuencia.

Un diagrama de Secuencia muestra una interacción ordenada según la secuencia temporal de eventos. En particular, muestra los objetos participantes en la interacción y los mensajes que intercambian

ordenados según su secuencia en el tiempo. El eje vertical representa el tiempo, y en el eje horizontal se colocan los objetos y actores participantes en la interacción, sin un orden prefijado. Cada objeto o actor tiene una línea vertical, y los mensajes se representan mediante flechas entre los distintos objetos. El tiempo fluye de arriba abajo. Se pueden colocar etiquetas (como restricciones de tiempo, descripciones de acciones, etc.) bien en el margen izquierdo o bien junto a las transiciones o activaciones a las que se refieren (Figura 61).

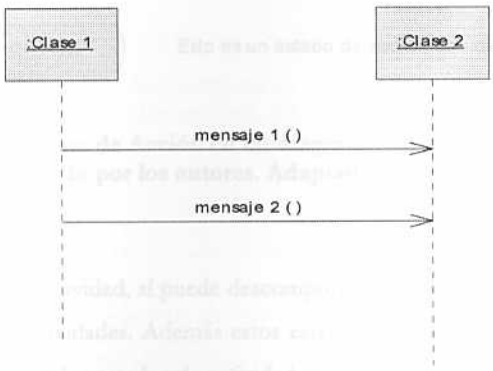


Figura 61. Diagrama de Secuencia en notación UML.
Fuente: Elaborado por los autores. Adaptada de (Ferré et al., 2004).

C.2.5 Diagramas De Actividades.

Los diagramas de actividades sirven fundamentalmente para modelar el flujo de control entre actividades. La idea es generar una especie de diagrama Pert, en el que se puede ver el flujo de actividades que tienen lugar a lo largo del tiempo, así como las tareas concurrentes que pueden realizarse a la vez. El diagrama de actividades sirve para representar el sistema desde otra perspectiva, y de este modo complementa a los anteriores diagramas vistos.

Gráficamente un diagrama de actividades será un conjunto de arcos y nodos. Desde un punto de vista conceptual, el diagrama de actividades muestra cómo fluye el control de unas clases a otras con la finalidad de culminar con un flujo de control total que se corresponde con la consecución de un proceso más complejo. Por este motivo, en un diagrama de actividades aparecerán acciones y actividades correspondientes a distintas clases. Colaborando todas ellas para conseguir un mismo fin.

C.2.5.1 Estados de actividad y estados de acción.

La representación de ambos es un rectángulo con las puntas redondeadas, en cuyo interior se representa bien una actividad o bien una acción. La forma de expresar tanto una actividad como una

acción, no queda impuesta por UML, se podría utilizar lenguaje natural, una especificación formal de expresiones, un metalenguaje, etc. La idea central es la siguiente: "Un estado que represente una acción es atómico, lo que significa que su ejecución se puede considerar instantánea y no puede ser interrumpida" (Figura 62).

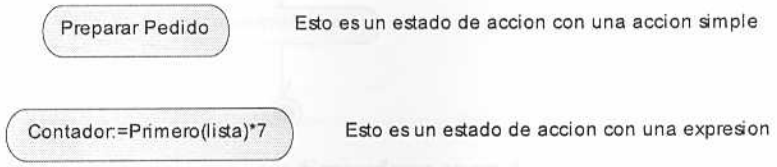


Figura 62. Estados de Acción en un diagrama de actividades UML. Fuente: Elaborado por los autores. Adaptada de (Ferré et al., 2004).

En cambio un estado de actividad, sí puede descomponerse en más sub-actividades representadas a través de otros diagramas de actividades. Además estos estados sí pueden ser interrumpidos y tardan un cierto tiempo en completarse. En los estados de actividad podemos encontrar otros elementos adicionales como son: acciones de entrada (entry) y de salida (exit) del estado en cuestión, así como definición de submáquinas (Figura 63).

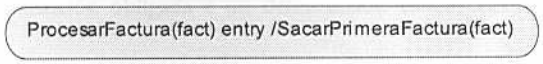


Figura 63. Estados de Actividad en un diagrama de actividades UML. Fuente: Elaborado por los autores. Adaptada de (Ferré et al., 2004).

C.2.5.2 Transiciones.

Las transiciones reflejan el paso de un estado a otro, bien sea de actividad o de acción. Esta transición se produce como resultado de la finalización del estado del que parte el arco dirigido que marca la transición. Como todo flujo de control debe empezar y terminar en algún momento, podemos indicar esto utilizando dos disparadores de inicio (Figura 64).

Figura 64. Transiciones sin disparadores en un diagrama de actividades UML.
Fuente: Elaborado por los autores. Adaptada de (Ferré et al., 2004).

C.2.5.3 Bifurcaciones.

Un flujo de control no tiene porqué ser siempre secuencial, puede presentar caminos alternativos. Para poder representar dichos caminos alternativos o bifurcación se utilizará como símbolo el rombo. Dicha bifurcación tendrá una transición de entrada y dos o más de salida. En cada transición de salida se colocará una expresión booleana que será evaluada una vez al llegar a la bifurcación, las guardas de la bifurcación han de ser excluyentes y contemplar todos los casos ya que de otro modo la ejecución del flujo de control quedaría interrumpida. Para poder cubrir todas las posibilidades se puede utilizar la palabra ELSE, para indicar una transición obligada a un determinado estado cuando el resto de guardas han fallado (Figura 65).

Figura 65. Bifurcación en un diagrama de actividades UML.
Fuente: Elaborado por los autores. Adaptada de (Ferré et al., 2004).

C.2.5.4 División y unión.

No sólo existe el flujo secuencial y la bifurcación, también hay algunos casos en los que se requieren tareas concurrentes. UML representa gráficamente el proceso de división, que representa la concurrencia, y el momento de la unión de nuevo al flujo de control secuencial, por una línea horizontal ancha (Figura 66).

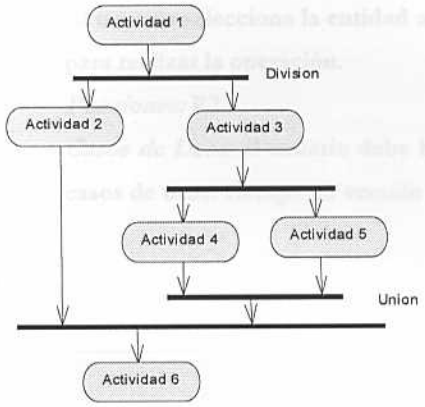


Figura 66. División y Unión en un diagrama de actividades UML.
Fuente: Elaborado por los autores. Adaptada de (Ferré et al., 2004).

APÉNDICE D. CASOS DE USO

Caso de Uso #2:	Eliminar entidad
Actores:	**Usuario (iniciador).**
Propósito:	**Eliminar una entidad de un modelo.**
Precondición:	**El usuario debe haber creado previamente la entidad a eliminar.**
Resumen:	**El usuario selecciona la entidad a eliminar y presiona la "equis" para realizar la operación.**
Referencias cruzadas:	***Funciones:*** **R2.1** ***Casos de Usos:*** **el usuario debe haber terminado alguno de los casos de usos:** ***Recuperar versión o Crear Entidad.***

Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario escoge la entidad que desea eliminar.	**2. El sistema habilita un botón en la barra de herramientas en donde se muestra un icono representativo ("equis") para la eliminación.**
3. El usuario presiona el botón "equis".	**4. El sistema muestra un mensaje de advertencia en donde se pide la confirmación de la operación a realizar.**
5. El usuario confirma la eliminación.	**6. El sistema remueve la entidad del espacio de trabajo, tanto gráficamente como internamente.**

Cursos alternos:

5.a) El usuario cancela la eliminación.

5.a.1) El sistema vuelve al espacio de trabajo y no realiza ningún cambio.

Caso de Uso #3:	Modificar entidad
Actores:	**Usuario (iniciador).**
Propósito:	**Realizar algún cambio a una entidad.**
Precondición:	**El usuario debe haber creado previamente la entidad.**

Resumen: **El usuario selecciona la entidad que desea modificar y realiza los cambios que desee.**

Referencias cruzadas: ***Funciones:*** **R3.1, R1.1, R1.2, R1.3.**

Casos de Usos: **el usuario debe haber terminado alguno de los casos de usos:** ***Recuperar versión o Crear entidad.***

Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario hace doble click en la entidad que desea modificar.	**2. El sistema muestra una ventana, en donde el usuario podrá modificar los datos de la entidad, dicha ventana posee la misma estructura que la que se despliega al momento de crear una entidad.**
3. El usuario modifica los datos que desee tanto generales, como de los atributos y presiona el botón "Finalizar".	**4. El sistema muestra un mensaje para la confirmación de dichos cambios.**
5. El usuario confirma que desea continuar con la operación.	**6. El sistema muestra gráficamente la entidad modificada.**

Cursos alternos:

3.a) El usuario no modifica ninguno de los datos de la entidad y presiona "Finalizar".

3.a.1) El sistema no realiza operación alguna y regresa al espacio de trabajo.

3.b) El usuario deja de llenar alguno de los campos o los llena incorrectamente.

3.b.1) El sistema muestra un mensaje de error según sea el caso.

3.c) El usuario presiona el botón "Cancelar".

3.c.1) El sistema cierra la ventana y regresa al espacio de trabajo.

3.d) El usuario elimina todos los atributo y presiona el botón finalizar.

3.d.1) El sistema muestra un mensaje donde le indica al usuario que al menos debe de existir un atributo clave dentro de la entidad.

3.e) El usuario no selecciona ninguno de los atributos como clave.

3.e.1) El sistema muestra un mensaje de error indicando que al menos un atributo debe ser clave.

Caso de Uso #4:	Crear Modelo
Actores:	**Usuario (iniciador).**
Propósito:	**Establecer un espacio de trabajo.**
Precondición:	**El usuario debe estar autentificado en el sistema.**
Resumen:	**El usuario selecciona en la barra de herramientas, el icono que corresponde a la creación de un nuevo modelo e introduce los datos correspondientes.**
Referencias cruzadas:	***Funciones:* R1.1, R1.2, R1.3, R4.1.** ***Casos de Usos:* el usuario debe haber terminado el caso de uso: *Autentificarse en el sistema.***

Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario presiona el icono "Nuevo modelo", en la barra de herramientas.	**2. El sistema muestra un formulario con los campos necesarios para crear un modelo. Dichos datos son los siguientes:** - *Nombre:* **seudónimo con el que identifica al modelo.** - *Descripción:* **breve explicación del objetivo o la función del modelo.**
3. El usuario introduce los datos del modelo a crear y presiona el botón "Finalizar".	**4. El sistema habilita un espacio de trabajo, con los datos introducidos por el usuario.**

Cursos alternos:

3.a) El usuario presiona el botón "Cancelar".

3.a.1) El sistema cierra la ventana y vuelve a la ventana principal.

3.b) El usuario deja de llenar alguno de los campos o los llena incorrectamente.

3.b.1) El sistema muestra un mensaje de error según sea el caso.

Caso de Uso #5:	Eliminar Modelo
Actores:	**Usuario (iniciador).**

Propósito:	**Remover un modelo.**
Precondición:	**El usuario debe estar autentificado en el sistema y haber creado algún modelo.**
Resumen:	**El usuario selecciona el modelo que desea eliminar y lo remueve de la memoria**
Referencias cruzadas:	***Funciones:*** **R3.1, R2.1.** ***Casos de Usos:*** **el usuario debe haber terminado el caso de uso:** ***Autentificarse en el sistema.***

Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario selecciona el modelo que desea eliminar en el menú "Abrir" de la barra de herramientas.	**2. El sistema habilita el icono que corresponde a una "equis".**
3. El usuario presiona el icono "equis".	**4. El sistema muestra un mensaje de advertencia donde se notifica al usuario que de completarse la transacción los datos se perderán definitivamente. Y solicita su confirmación.**
5. El usuario confirma la eliminación del modelo.	**6. El sistema elimina el modelo de la memoria y todos los datos asociados a este.**

Cursos alternos:

3.a) El usuario no presiona el icono "equis".

3.a.1) El sistema no realiza operación alguna.

5.a) El usuario cancela la transacción.

5.a.1) El sistema vuelve al espacio de trabajo donde se inició la transacción.

Caso de Uso #6:	Modificar Modelo
Actores:	**Usuario (iniciador).**
Propósito:	**Realizar algún cambio al modelo.**
Precondición:	**El usuario debe estar autentificado en el sistema y haber creado**

	algún modelo.
Resumen:	**El usuario selecciona el modelo que desea modificar y realiza los cambios que desee.**
Referencias cruzadas:	***Funciones:*** **R3.1, R1.1, R1.2, R1.3.** ***Casos de Usos:*** **el usuario debe haber terminado el caso de uso:** ***Autentificarse en el sistema.***

Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario hace doble click sobre el modelo que desea modificar en el árbol donde se encuetran cada uno de los objetos creados.	**2. El sistema muestra una ventana, en donde el usuario podrá modificar los datos del modelo, dicha ventana posee la misma estructura que la que se despliega al momento de crear un modelo.**
3. El usuario modifica los datos que desee y presiona el botón "Finalizar".	**4. El sistema muestra un mensaje para la confirmación de dichos cambios.**
5. El usuario confirma que desea continuar con la operación.	**6. El sistema vuelve a la ventana principal.**

Cursos alternos:

3.a) El usuario cancela la transacción.

3.a.1) El sistema vuelve al espacio de trabajo donde se inició la transacción.

3.b) El usuario deja de llenar alguno de los campos o los llena incorrectamente.

3.b.1) El sistema muestra un mensaje de error según sea el caso.

5.a) El usuario aborta la operación.

5.a.1) El sistema cancela la operación y vuelve a la ventana principal.

Caso de Uso #7:	Crear Dominio
Actores:	**Usuario (iniciador).**
Propósito:	**Agregar un dominio al modelo o versión actual.**

Precondición:	**El usuario debe haber creado un modelo o estar trabajando sobre una versión existente.**
Resumen:	**El usuario selecciona desde la barra de herramientas o desde la ventana "atributos", la opción crear dominio e ingresa los datos correspondientes.**
Referencias cruzadas:	***Funciones:*** **R1.1, R1.2, R1.3.** ***Casos de Usos:*** **el usuario debe haber terminado alguno de los casos de usos:** ***Crear modelo o Recuperar versión.***

Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario selecciona desde la barra de herramientas o en la ventana "Atributos", la opción crear dominio.	**2. El sistema muestra un formulario con los campos necesarios para crear un dominio. Dichos datos son los siguientes:** - *Nombre:* **seudónimo con el que identifica el dominio.** - *Descripción:* **breve explicación del objetivo o la función del dominio.** - *Valores:* **lista de posibilidades que puede tomar ese dominio. Ej: M y F, en caso de que el dominio sea "Sexo".**
3. El usuario introduce los datos del dominio a crear y presiona el botón "Finalizar".	**4. El sistema incluye el dominio dentro de los valores posibles para la asignación a los atributos.**

Cursos alternos:

3.a) El usuario presiona el botón "cancelar".

3.a.1) El sistema no realiza operación alguna y vuelve a la ventana "Atributos".

3.b) El usuario deja de llenar alguno de los campos o los llena incorrectamente.

3.b.1) El sistema muestra un mensaje de error según sea el caso.

Caso de Uso #8:	Asignar dominio a atributo
Actores:	**Usuario (iniciador).**
Propósito:	**Asignar un dominio a un atributo de una entidad.**

Precondición:	**El usuario debe haber creado una entidad o estar trabajando sobre una entidad existente.**
Resumen:	**El usuario selecciona desde la ventana "atributos", la opción asignar a dominio y escoge uno entre los dominios existente**
Referencias cruzadas:	***Funciones:*** **R3.1, R1.3.** ***Casos de Usos:*** **el usuario debe haber terminado alguno de los casos de usos:** ***Crear entidad o Recuperar versión.***

Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario selecciona el atributo al que desea asignar el dominio y lo escoge de la lista que despliega el combo "dominio"	**2. El sistema muestra una lista con los dominios pertenecientes al modelo actual.**
3. El usuario escoge el dominio que desea asignar.	**4. El sistema asigna el dominio al atributo.**
5. El usuario presiona el botón "Finalizar" o asigna algún dominio a los demás a atributos.	**6. El sistema muestra gráficamente los cambios en la entidad.**

Cursos alternos:

5.a) El usuario presiona el botón "cancelar".

5.a.1) El sistema no realiza operación alguna y vuelve a la ventana principal.

Caso de Uso #9:	Autentificarse en el sistema
Actores:	**Usuario (iniciador).**
Propósito:	**Identificarse como usuario del sistema.**
Precondición:	**El usuario debe estar registrado en el sistema.**
Resumen:	**El usuario introduce sus datos de *login* y *password* para**

	autentificarse en el sistema.
Referencias cruzadas:	***Funciones:*** **R1.1, R1.2, R5.1.**

Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario introduce los datos en la ventana "autentificación".	**2. El sistema toma los datos del formulario de autentificación y verifica que el usuario exista para darle acceso a la herramienta.**

Cursos alternos:

1.a) El usuario deja de llenar alguno de los campos o los llena incorrectamente.

1.a.1) El sistema muestra un mensaje de error según sea el caso.

1.b) El usuario no está registrado en la herramienta.

1.b.1) El sistema da la opción al usuario de que se registre en el sistema a través de un link en la ventana de autentificación.

2.a) El sistema no encuentra coincidencia entre los datos introducidos por usuario y el registro de los usuarios del sistema.

2.a.1) El sistema muestra un aviso de que no se encontraron coincidencias, por lo que solicita al usuario que verifique los datos o que realice su registro en el sistema.

Caso de Uso #10:	Registrar usuarios.
Actores:	**Usuario (iniciador).**
Propósito:	**Pertenecer a las personas que tienen acceso al sistema.**
Precondición:	
Resumen:	**El usuario introduce los datos que correspondan para registrarse en el sistema.**
Referencias cruzadas:	***Funciones:*** **R1.1, R1.2, R1.3.**

Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario introduce los datos en la ventana "Registro de usuarios". Los cuales son: - *Nombre:* **nombre del usuario que desea registrarse.** - *Login:* **pseudónimo con el que el usuario se identifica en la aplicación.** - *Password:* **clave para ingresar a la aplicación.**	**2. El sistema verifica que el usuario no exista y lo registra en el sistema.**

Cursos alternos:

1.a) El usuario deja de llenar alguno de los campos o los llena incorrectamente.

1.a.1) El sistema muestra un mensaje de error según sea el caso.

1.b) El usuario presiona el botón "cancelar".

1.b.1) El sistema no realiza operación alguna.

2.a) El sistema encuentra un usuario con los mismos atributos claves en la aplicación.

2.a.1) El sistema envía al usuario un mensaje notificándole el problema y solicitándole que introduzca de nuevo los datos.

Caso de Uso #11:	Crear relación.
Actores:	**Usuario (iniciador).**
Propósito:	**Agregar una relación entre dos entidades.**
Precondición:	**Que exista al menos una entidad en el modelo.**
Resumen:	**El usuario introduce los datos que correspondan para crear una relación.**
Referencias cruzadas:	***Funciones:* R1.1, R1.2, R1.3.**

Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario selecciona el icono que corresponde a la relación desde la barra de herramientas.	**2. El sistema despliega una ventana en la que muestran los campos para agregar los datos generales de la relación. Estos son:** - *Nombre:* **seudónimo con el que**

	identifica la relación. - *Descripción:* **breve explicación del objetivo o la función de la relación.**
3. El usuario introduce los datos de la relación generales de la relación y presiona el botón "continuar".	**4. El sistema muestra otro formulario en se permite al usuario agregar los siguientes datos:** - *Cardinalidad:* **representa las posibilidades de que se encuentren los registros en la entidad. Ejemplo: uno a muchos, uno a uno, etc.** - *Opcionalidad:* **indica si la relación es mandatoria o no.** - *Entidad origen:* **es la entidad de donde sale la relación.** - *Entidad fin:* **es la entidad a donde llega la relación.**
4. El usuario llena los campos con los datos de la relación y presiona el botón "finalizar".	**5. El sistema muestra gráficamente la relación.**

Cursos alternos:

3.a) El usuario deja de llenar alguno de los campos o los llena incorrectamente.

1.a.1) El sistema muestra un mensaje de error según sea el caso.

3.b) El usuario presiona el botón "cancelar".

1.b.1) El sistema no realiza operación alguna.

4.a) El usuario deja de llenar alguno de los campos o los llena incorrectamente.

1.a.1) El sistema muestra un mensaje de error según sea el caso.

4.b) El usuario presiona el botón "cancelar".

1.b.1) El sistema muestra un mensaje de advertencia en donde se indica al usuario que de cancelar la operación perderá los datos generales de la relación.

4.c) El usuario escoge como cardinalidad muchos a muchos.

4.c.1) El sistema muestra un mensaje en el que se le informa al usuario que la forma correcta de hacer una relación de este tipo es creando una entidad intersección.

Caso de Uso #12:	Eliminar relación.
Actores:	**Usuario (iniciador).**
Propósito:	**Remover una relación.**

Precondición:	**Que exista una relación en el modelo.**
Resumen:	**El usuario selecciona la relación que desea eliminar y la remueve.**
Referencias cruzadas:	***Funciones:*** **R2.1.** ***Casos de Usos:*** **el usuario debe haber terminado alguno de los casos de usos:** ***Recuperar versión o Crear Relación.***

Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario escoge la relación que desea eliminar.	**2. El sistema habilita un botón en la barra de herramientas en donde se muestra un icono representativo ("equis") para la eliminación.**
3. El usuario presiona el botón "equis".	**4. El sistema muestra un mensaje de advertencia en donde se pide la confirmación de la operación a realizar.**
5. El usuario confirma la eliminación.	**6. El sistema remueve la relación del espacio de trabajo, tanto gráficamente como internamente.**

Cursos alternos:

3.a) El usuario no presiona el botón "equis".

3.a.1) No se genera ningún cambio en el sistema.

5.a) El usuario cancela la eliminación.

5.a.1) El sistema vuelve al espacio de trabajo y no realiza ningún cambio.

Caso de Uso #13:	Modificar Relación
Actores:	**Usuario (iniciador).**
Propósito:	**Realizar algún cambio a una relación.**
Precondición:	**El usuario debe haber creado previamente la relación.**
Resumen:	**El usuario selecciona la relación que desea modificar y realiza los cambios que desee.**

Referencias cruzadas: ***Funciones:*** **R3.1, R1.1, R1.2, R1.3.**
Casos de Usos: **el usuario debe haber terminado alguno de los casos de usos: *Recuperar versión o Crear relación.***

Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario hace doble clic sobre la relación que desea modificar.	**2. El sistema muestra una ventana, en donde el usuario podrá modificar los datos de la relación, dicha ventana posee la misma estructura que la que se despliega al momento de crear una relación.**
3. El usuario modifica los datos que desee tanto generales, como específicos y presiona el botón "Finalizar".	**4. El sistema muestra un mensaje para la confirmación de dichos cambios.**
5. El usuario confirma que desea continuar con la operación.	**6. El sistema muestra gráficamente la relación modificada.**

Cursos alternos:

3.a) El usuario no modifica ninguno de los datos de la relación y presiona "Finalizar".

3.a.1) El sistema no realiza operación alguna y regresa al espacio de trabajo.

3.b) El usuario deja de llenar alguno de los campos o los llena incorrectamente.

3.b.1) El sistema muestra un mensaje de error según sea el caso.

3.c) El usuario presiona el botón "Cancelar".

3.c.1) El sistema cierra la ventana y regresa al espacio de trabajo.

Caso de Uso #14: Modificar Dominio.

Actores: **Usuario (iniciador).**

Propósito: **Realizar algún cambio al dominio.**

Precondición: **El usuario debe haber creado previamente el dominio.**

Resumen: **El usuario selecciona el dominio que desea modificar y realiza los cambios que desee.**

Referencias cruzadas: ***Funciones:*** **R3.1, R1.1, R1.2, R1.3.**

Casos de Usos: **el usuario debe haber terminado alguno de los casos de usos:** ***Recuperar versión o Crear dominio.***

Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario escoge el dominio que desea modificar del árbol donde se encuentran cada uno de los objetos creados por el usuario.	**2. El sistema muestra una ventana, en donde el usuario podrá modificar los datos del dominio, dicha ventana posee la misma estructura que la que se despliega al momento de crearlo.**
3. El usuario modifica los datos que desee y presiona el botón "Finalizar".	**4. El sistema muestra un mensaje para la confirmación de dichos cambios.**
5. El usuario confirma que desea continuar con la operación.	**6. El sistema vuelve a la ventana principal.**

Cursos alternos:

3.a) El usuario no modifica ninguno de los datos de la relación y presiona "Finalizar".

3.a.1) El sistema no realiza operación alguna y regresa al espacio de trabajo.

3.b) El usuario deja de llenar alguno de los campos o los llena incorrectamente.

3.b.1) El sistema muestra un mensaje de error según sea el caso.

3.c) El usuario presiona el botón "Cancelar".

3.c.1) El sistema cierra la ventana y regresa al espacio de trabajo.

5.a) El usuario aborta la operación.

5.a.1) El sistema vuelve al espacio donde se inició la actividad.

Caso de Uso #15: Crear versión.

Actores: **Usuario (iniciador).**

Propósito: **Crear una versión de un modelo en específico.**

Precondición: **Que exista al menos un modelo.**

Resumen: **El usuario introduce los datos que correspondan para crear una**

	versión.
Referencias cruzadas:	***Funciones:*** **R1.1, R1.2, R1.3.**

Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario selecciona el icono que corresponde a la versión desde la barra de herramientas.	**2. El sistema despliega una ventana en la que muestran los campos para agregar los datos generales de la versión. Estos son:** - *Nombre:* **seudónimo con el que identifica a la versión.** - *Descripción:* **breve explicación del objetivo o la función de la versión.**
3. El usuario introduce los datos de la versión y presiona el botón "finalizar".	**4. El sistema crea la nueva versión.**

Cursos alternos:

3.a) El usuario deja de llenar alguno de los campos o los llena incorrectamente.

1.a.1) El sistema muestra un mensaje de error según sea el caso.

3.b) El usuario presiona el botón "cancelar".

1.b.1) El sistema no realiza operación alguna.

Caso de Uso #16:	Crear usuario.
Actores:	**Usuario (iniciador).**
Propósito:	**Crear usuario para la manipulación del modelo.**
Precondición:	**Que exista al menos un modelo.**
Resumen:	**El usuario introduce los datos que correspondan para crear un usuario para la manipulación del modelo en la base de datos.**
Referencias cruzadas:	***Funciones:*** **R1.1, R1.2, R1.3.**

Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario selecciona la opción "crear	**2. El sistema despliega una ventana en la que**

usuario" desde la barra de herramientas.	**muestran los campos para agregar los datos generales del usuario. Estos son:** - *Nombre:* **nombre de la persona que se va a identificar como usuario.** - *Login:* **seudónimo con el que se identifica al usuario.** - *Password:* **clave para identificarse.**
3. El usuario introduce los datos generales y presiona el botón "continuar".	**4. El sistema despliega una ventana en la que da la posibilidad de asignar roles y privilegios al usuario a crear.**
5. El usuario asigna los roles y privilegios que desee y presiona el botón "Finalizar".	**6. El sistema asigna los usuarios creados al modelo.**

Cursos alternos:

3.a) El usuario deja de llenar alguno de los campos o los llena incorrectamente.

1.a.1) El sistema muestra un mensaje de error según sea el caso.

3.b) El usuario presiona el botón "cancelar".

1.b.1) El sistema no realiza operación alguna.

5.a) El usuario presiona el botón "cancelar".

5.a.1) El sistema despliega un mensaje de advertencia en donde se le informa al usuario que si cancela la operación, perderá los datos del usuario.

Caso de Uso #17:	Crear rol.
Actores:	**Usuario (iniciador).**
Propósito:	**Crear un rol para poder asignarlo a un usuario.**
Precondición:	**Que exista al menos un modelo.**
Resumen:	**El usuario introduce los datos que correspondan para crear un rol.**
Referencias cruzadas:	***Funciones:* R1.1, R1.2, R1.3.**

Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario selecciona la opción "crear rol" desde la ventana "Usuarios".	**2. El sistema despliega una ventana en la que muestran los campos para agregar los datos generales del rol. Estos son:** - *Nombre:*. **seudónimo con el que se identifica al rol.** - *Privilegios:* **lista de funcionalidades que se le asignan al rol. Estos privilegios están predefinidos por el sistema.**
3. El usuario introduce los datos y presiona el botón "finalizar".	**4. El sistema adopta el nuevo rol entre la lista de roles disponibles para asignarle a los usuarios del modelo.**

Cursos alternos:

3.a) El usuario deja de llenar alguno de los campos o los llena incorrectamente.

1.a.1) El sistema muestra un mensaje de error según sea el caso.

3.b) El usuario presiona el botón "cancelar".

1.b.1) El sistema no realiza operación alguna.

Caso de Uso #18:	Modificar rol.
Actores:	**Usuario (iniciador).**
Propósito:	**Alterar un rol.**
Precondición:	**Que el rol haya sido creado anteriormente.**
Resumen:	**El usuario introduce los datos que desea modificar.**
Referencias cruzadas:	***Funciones:*** **R3.1,R1.1, R1.2, R1.3.**

Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario hace doble click en el rol que desea modificar en el árbol donde se encuentrn cada uno de los objetos creados.	**2. El sistema muestra una ventana, en donde el usuario podrá modificar los datos del dominio, dicha ventana posee la misma estructura que la que se despliega al momento de crearlo.**

3. El usuario modifica los datos que desee y presiona el botón "Finalizar".	**4. El sistema muestra un mensaje para la confirmación de dichos cambios.**
5. El usuario confirma que desea continuar con la operación.	**6. El sistema vuelve a la ventana principal.**

Cursos alternos:

3.a) El usuario no modifica ninguno de los datos de la relación y presiona "Finalizar".

3.a.1) El sistema no realiza operación alguna y regresa al espacio de trabajo.

3.b) El usuario deja de llenar alguno de los campos o los llena incorrectamente.

3.b.1) El sistema muestra un mensaje de error según sea el caso.

3.c) El usuario presiona el botón "Cancelar".

3.c.1) El sistema cierra la ventana y regresa al espacio de trabajo.

5.a) El usuario aborta la operación.

5.a.1) El sistema vuelve al espacio donde se inició la actividad.

Caso de Uso #19:	Eliminar rol.
Actores:	**Usuario (iniciador).**
Propósito:	**Remover un rol.**
Precondición:	***Que exista una rol en el modelo.***
Resumen:	**El usuario selecciona rol que desea eliminar y lo remueve.**
Referencias cruzadas:	***Funciones:*** **R2.1.** ***Casos de Usos:*** **el usuario debe haber terminado alguno de los casos de usos:** ***Recuperar versión o Crear Rol.***

Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario escoge el rol que desea eliminar del árbol ubicado a la izquierda de la ventana principal.	**2. El sistema habilita un botón en la barra de herramientas en donde se muestra un icono representativo ("equis") para la eliminación.**

3. El usuario presiona el botón "equis".	**4. El sistema muestra un mensaje de advertencia en donde se pide la confirmación de la operación a realizar.**
5. El usuario confirma la eliminación.	**6. El sistema remueve el rol del modelo.**

Cursos alternos:

3.a) El usuario no presiona el botón "equis".

3.a.1) No se genera ningún cambio en el sistema.

5.a) El usuario cancela la eliminación.

5.a.1) El sistema vuelve al espacio de trabajo y no realiza ningún cambio.

Caso de Uso #20:	Generar modelo lógico.
Actores:	**Usuario (iniciador).**
Propósito:	**Crear la representación del modelo lógico.**
Precondición:	**Que el modelo conceptual exista.**
Resumen:	**El usuario selecciona la opción generar modelo lógico y el sistema lo muestra gráficamente.**
Referencias cruzadas:	***Funciones:*** **R6.1** ***Casos de Usos:*** **el usuario debe haber terminado alguno de los casos de usos:** ***Crear modelo o Recuperar modelo.***

Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario selecciona la opción "Generar modelo lógico" en el menú principal.	**2. El sistema despliega una ventana en donde se le da la opción al usuario de que escoja la versión a la que le desea realizar el proceso.**
3. El usuario escoge la versión.	**4. El sistema muestra gráficamente el modelo lógico.**

Cursos alternos:

3.a) El usuario presiona el botón "cancelar".

3.a.1) No se genera ningún cambio en el sistema.

Caso de Uso #21:	Generar modelo físico.
Actores:	**Usuario (iniciador).**
Propósito:	**Crear el script para el manejador de base de datos .**
Precondición:	**Que el modelo conceptual exista.**
Resumen:	**El usuario selecciona la opción generar modelo físico y el sistema lo muestra gráficamente.**
Referencias cruzadas:	***Funciones:*** **R6.2** ***Casos de Usos:*** **el usuario debe haber terminado alguno de los casos de usos:** ***Crear modelo o Recuperar modelo.***

Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario selecciona la opción "Generar modelo físico" en el menú principal.	**2. El sistema despliega una ventana en donde se permite al usuario seleccionar el manejador de base de datos, para el cual desea generar el script y la version a la cual quiere generarlo.**
3. El usuario escoge la opción de su preferencia y presiona el botón "Siguiente".	**4. El sistema muestra otra ventana donde se muestran el tipo de archivo que se puede generar.**
5. El usuario escoge el tipo de archivo y presiona el botón "Siguiente".	**6. El sistema despliega una ventana para que el usuario seleccione la ruta donde desea guardar el archivo sql.**
7. El usuario selecciona la ruta y presiona el botón "Finalizar".	**8. El sistema genera el script sql.**

Cursos alternos:

3.a) El usuario presiona el botón "cancelar".

3.a.1) El sistema no realiza operación alguna.

5.a) El usuario presiona el botón "cancelar".

5.a.1) El sistema no realiza operación alguna.

7.a) El usuario presiona el botón "cancelar".

7.a.1) El sistema no realiza operación alguna.

Caso de Uso #22:	Generar reportes.
Actores:	**Usuario (iniciador).**
Propósito:	**Crear el reporte del modelo que desee.**
Precondición:	**Que el modelo conceptual exista.**
Resumen:	**El usuario selecciona la opción del reporte que desee crear.**
Referencias cruzadas:	***Funciones:*** **R6.2** ***Casos de Usos:*** **el usuario debe haber terminado alguno de los casos de usos:** ***Crear modelo o Recuperar modelo.***

Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario selecciona la opción "Generar reporte" en el menú principal.	**2. El sistema despliega una ventana en donde se permite al usuario seleccionar la versión sobre la cual desea generar el reporte de diccionario de datos.**
3. El usuario escoge la opción de su preferencia.	**4. El sistema genera el reporte en el formato HTML.**

Cursos alternos:

3.a) El usuario presiona el botón "cancelar".

3.a.1) El sistema no realiza operación alguna.

Caso de Uso #23:	Realizar Ingeniería Reversa.
Actores:	**Usuario (iniciador).**
Propósito:	**Crear el modelo lógico del script SQL.**
Precondición:	**Que el modelo conceptual exista.**
Resumen:	**El usuario selecciona la opción "Ingeniería reversa" y le proporciona al sistema el script que desea pasar a modelo lógico.**
Referencias cruzadas:	***Funciones:*** **R6.2**

Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario selecciona la opción "Ingeniería Reversa" en el menú principal.	**2. El sistema despliega una ventana en donde se permite al usuario introducir la ruta donde se encuentra el archivo sql que se desea procesar.**
3. El usuario selecciona la ruta del archivo sql.	**4. El sistema genera el modelo lógico y lo muestra gráficamente al usuario.**

Cursos alternos:

3.a) El usuario presiona el botón "cancelar".

3.a.1) El sistema no realiza operación alguna.

4.a) El sistema no genera el modelo lógico.

4.a.1) El sistema muestra un mensaje en el que se especifica la excepción por la cual no pudo ser generado el modelo.

Caso de Uso #24:	Recuperar versión.
Actores:	**Usuario (iniciador).**
Propósito:	**Activa una versión de un modelo en específico.**
Precondición:	**Que exista al menos una versión.**
Resumen:	**El usuario selecciona la versión del modelo que desea activar.**
Referencias cruzadas:	***Funciones:* R3.1**

Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario selecciona la opción "Abrir" en el menú principal.	**2. El sistema despliega una ventana en donde se muestra al usuario cada uno de sus modelos y las versiones asociadas a dichos modelos.**

3. El usuario selecciona la versión del modelo que desea activar.	**4. El sistema activa la versión y la muestra al usuario.**

Cursos alternos:

3.a) El usuario presiona el botón "cancelar".

3.a.1) El sistema no realiza operación alguna.

Caso de Uso #25:	Realizar consulta.
Actores:	**Usuario (iniciador).**
Propósito:	**Realizar consultas sobre los objetos creados.**
Precondición:	**Que exista un modelo de datos y que se haya generado una versión lógica del mismo.**
Resumen:	**El usuario selecciona los datos que desee consultar en la base de datos.**
Referencias cruzadas:	***Funciones:* R3.1, R1.3.**

Curso normal de los eventos:

Acción de los actores	Respuesta del sistema
1. El usuario selecciona la opción "Consulta" en la paleta.	**2. El sistema despliega una ventana en donde se muestra al usuario un formulario donde este debe de seleccionar las tablas, los campos y las condiciones sobre las cuales desea realizar la consulta.**
3. El usuario selecciona los datos.	**4. El sistema muestra la consulta en el lenguaje que propio de la herramienta (Lenguaje de recuperación).**

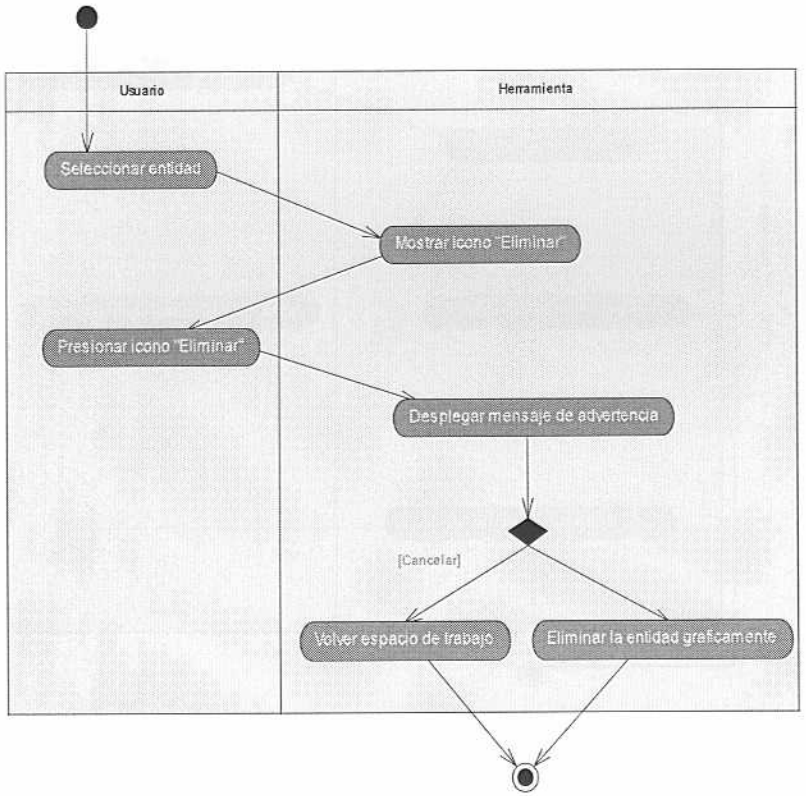
Cursos alternos:

3.a) El usuario presiona el botón "cancelar".

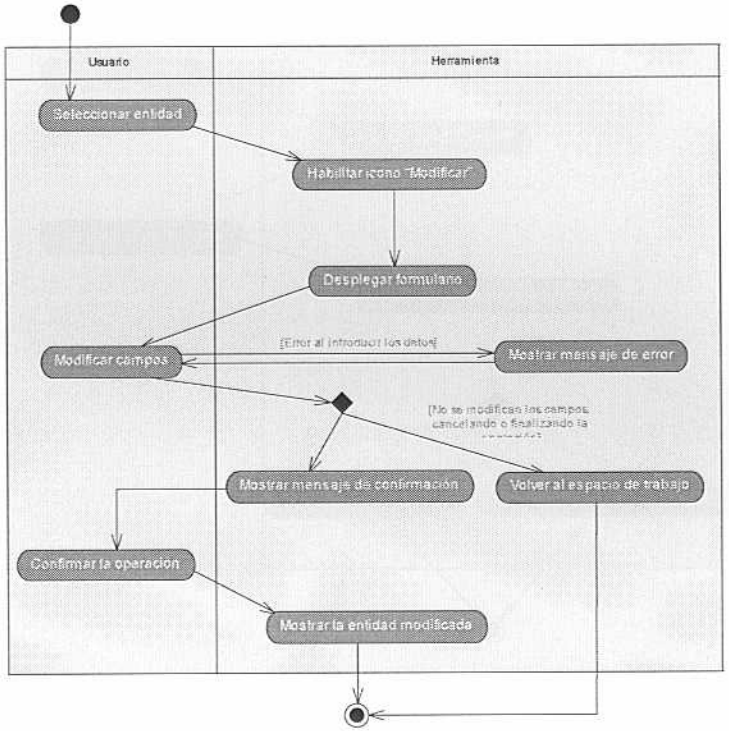
3.a.1) El sistema no realiza operación alguna.

APÉNDICE E. DIAGRAMAS DE ACTIVIDADES

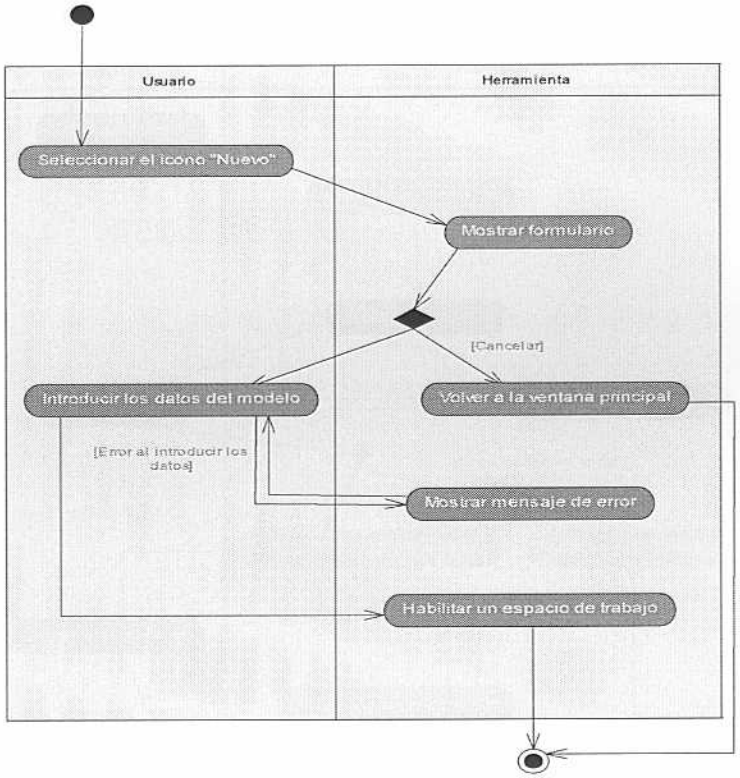
Eliminar Entidad.



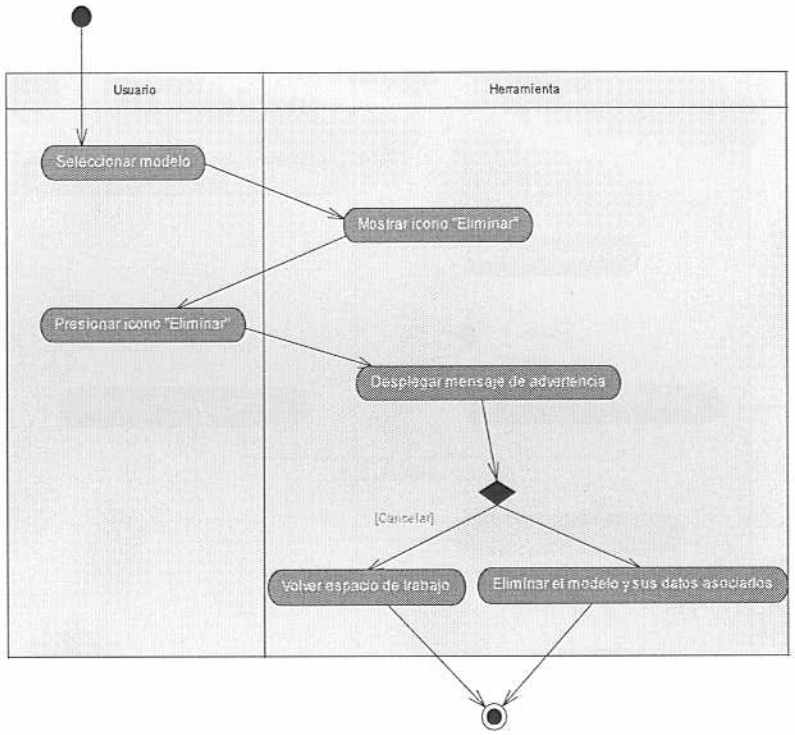
Modificar Entidad.



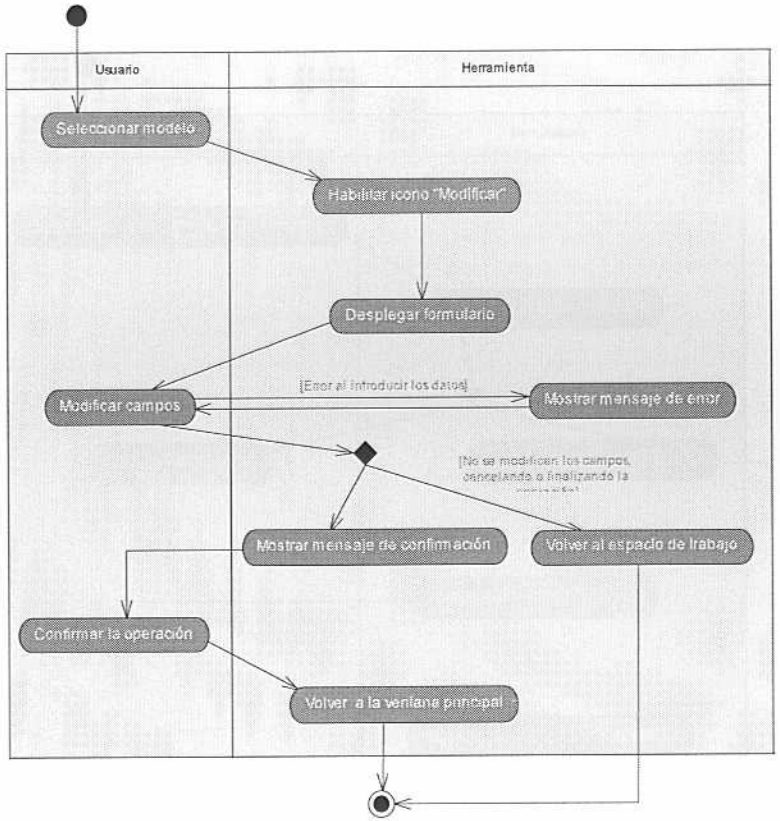
Crear Modelo.



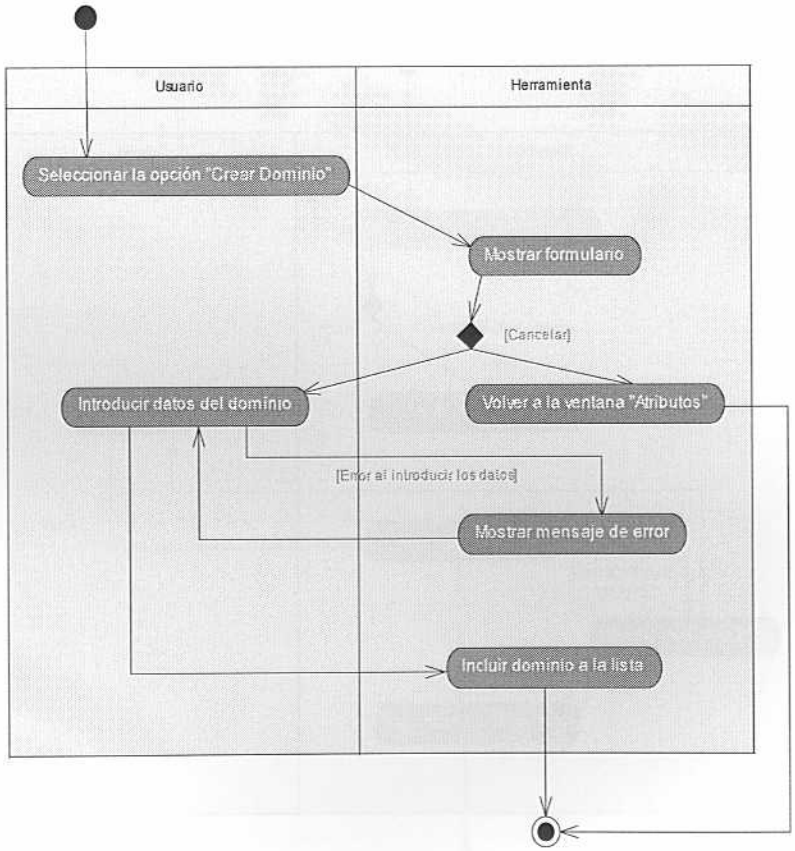
Eliminar Modelo.



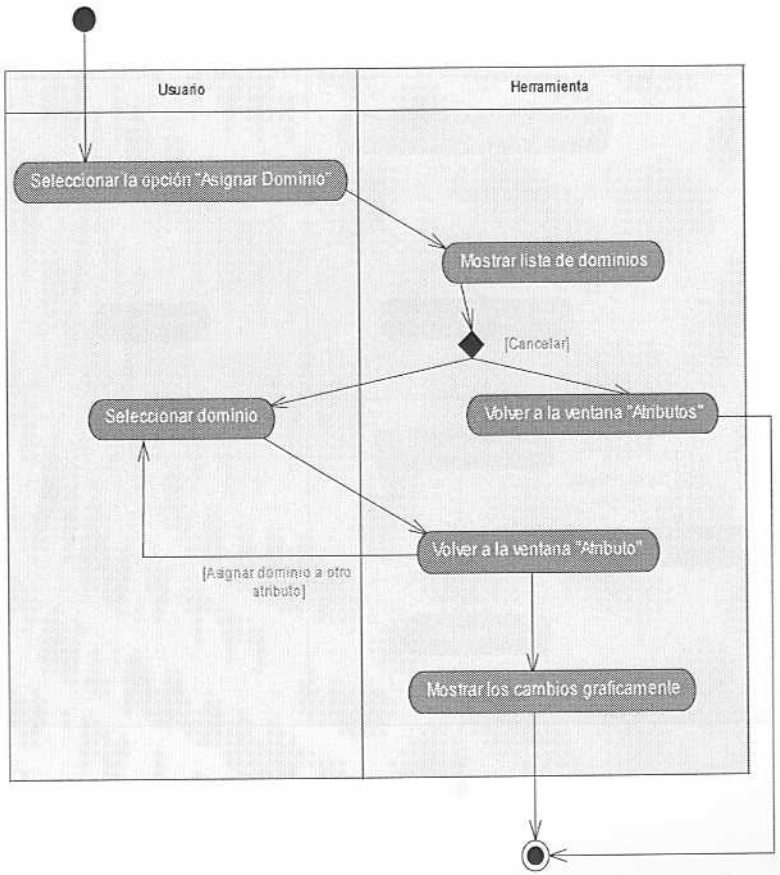
Modificar Modelo.



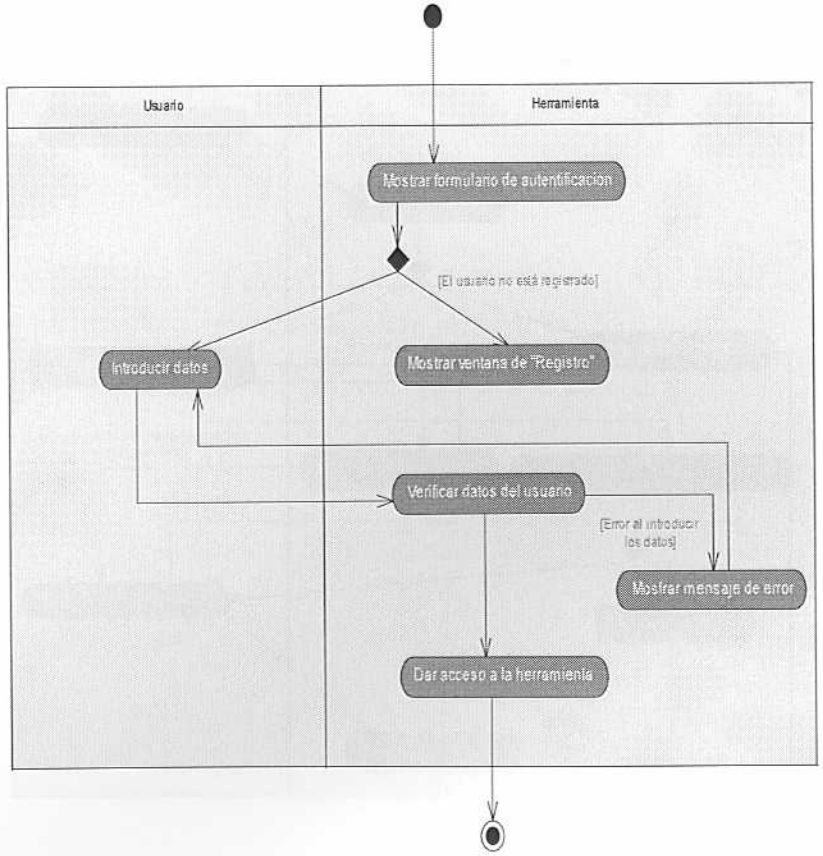
Crear Dominio.



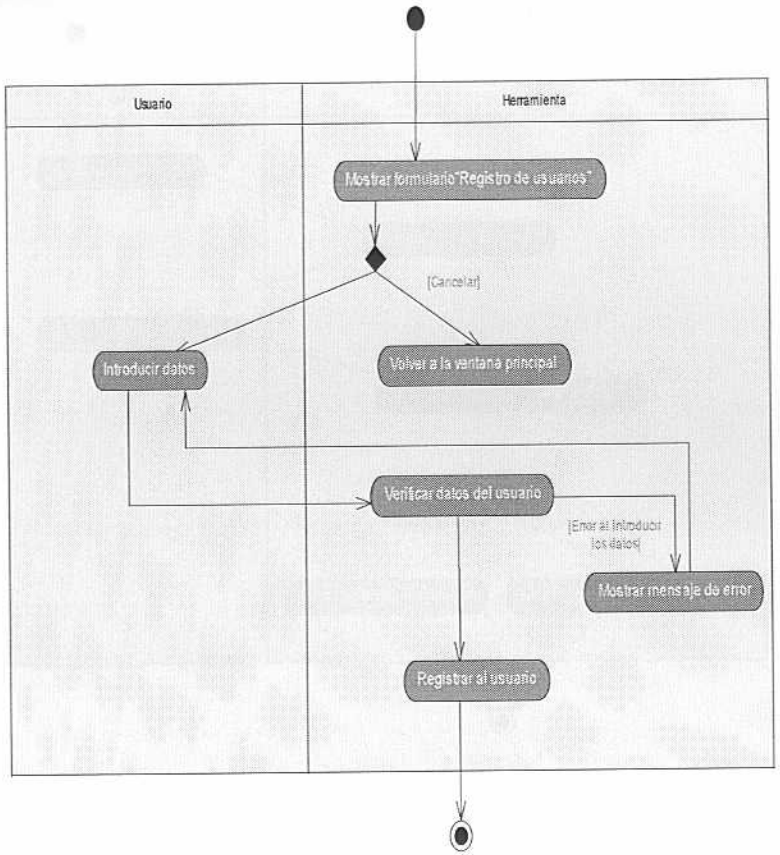
Asignar dominio a atributo.



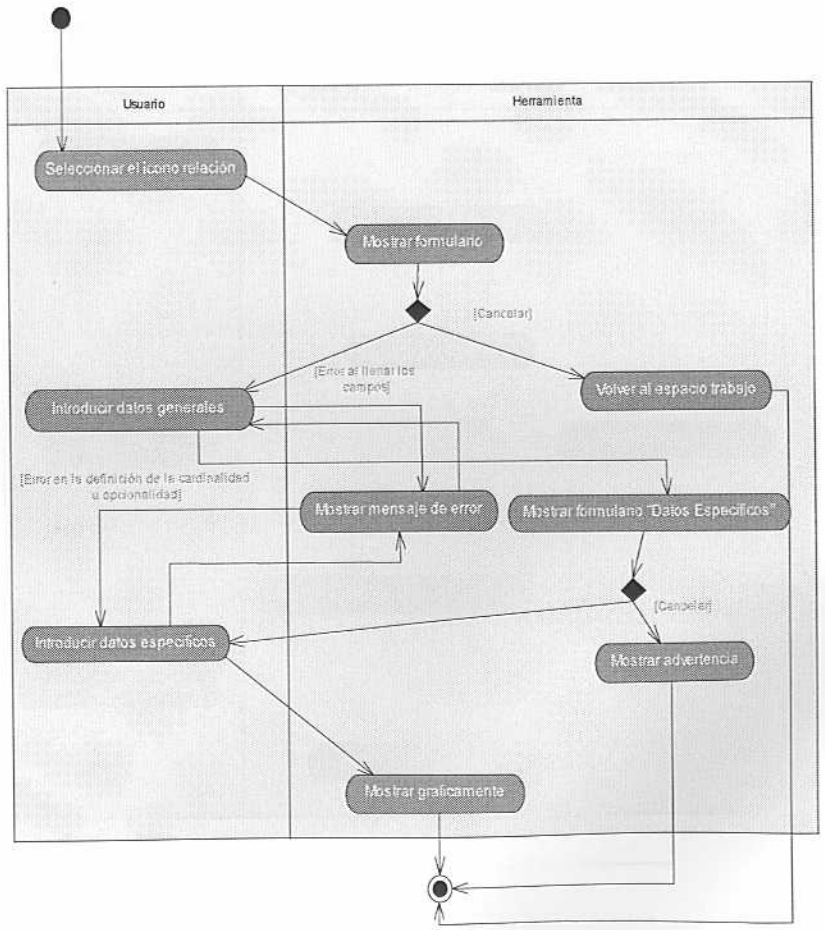
Autentificarse en el sistema.



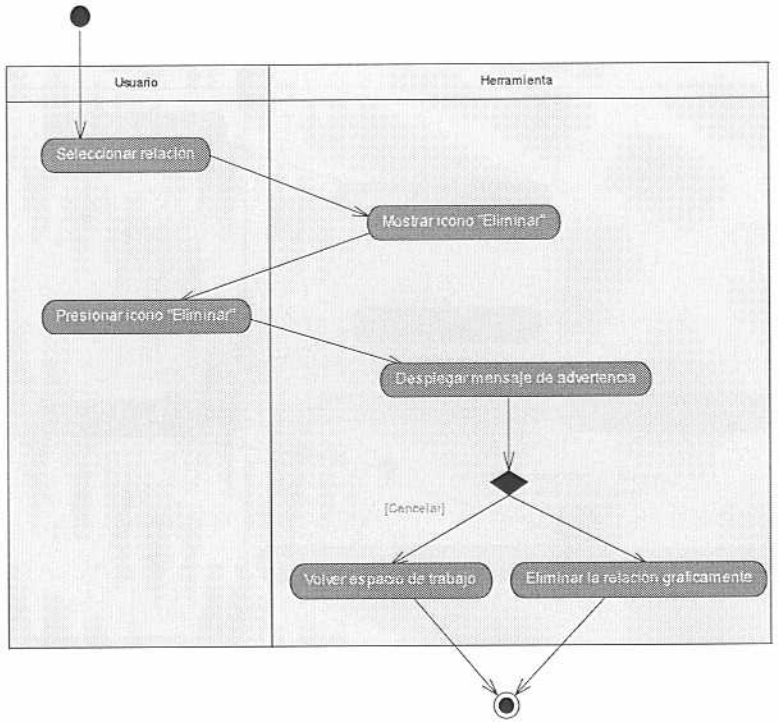
Registrar usuarios.



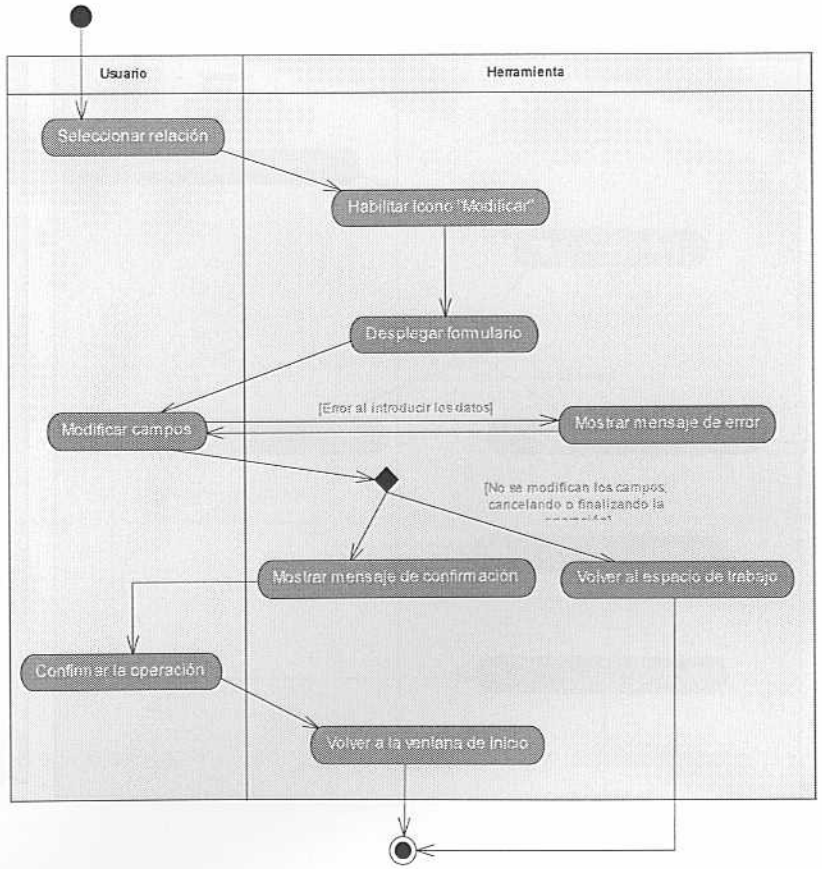
Crear relación.



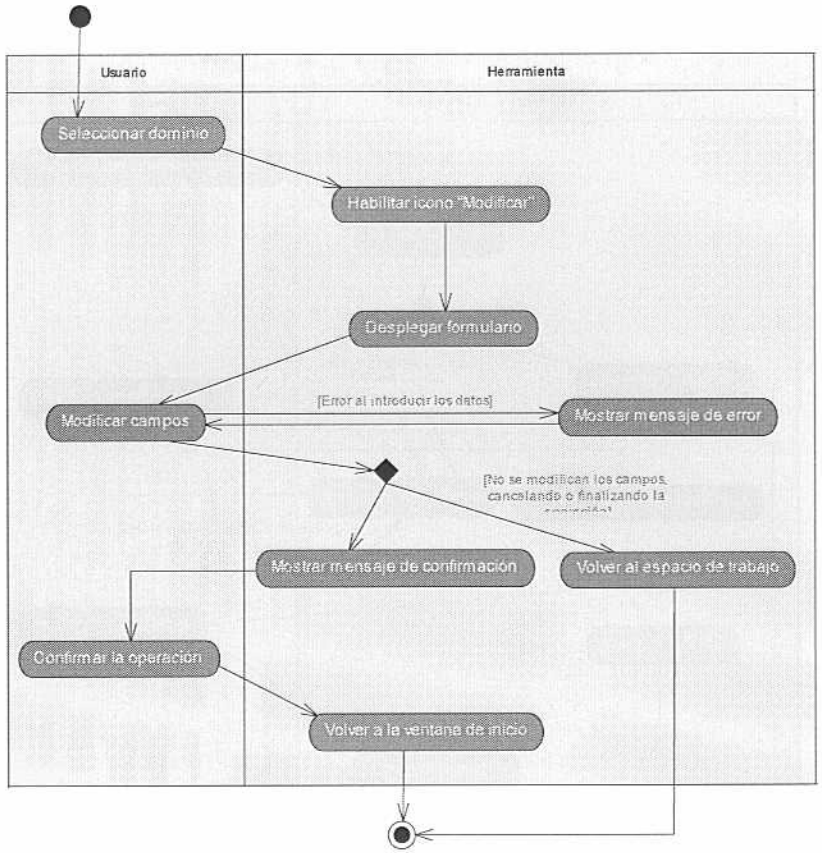
Eliminar relación.



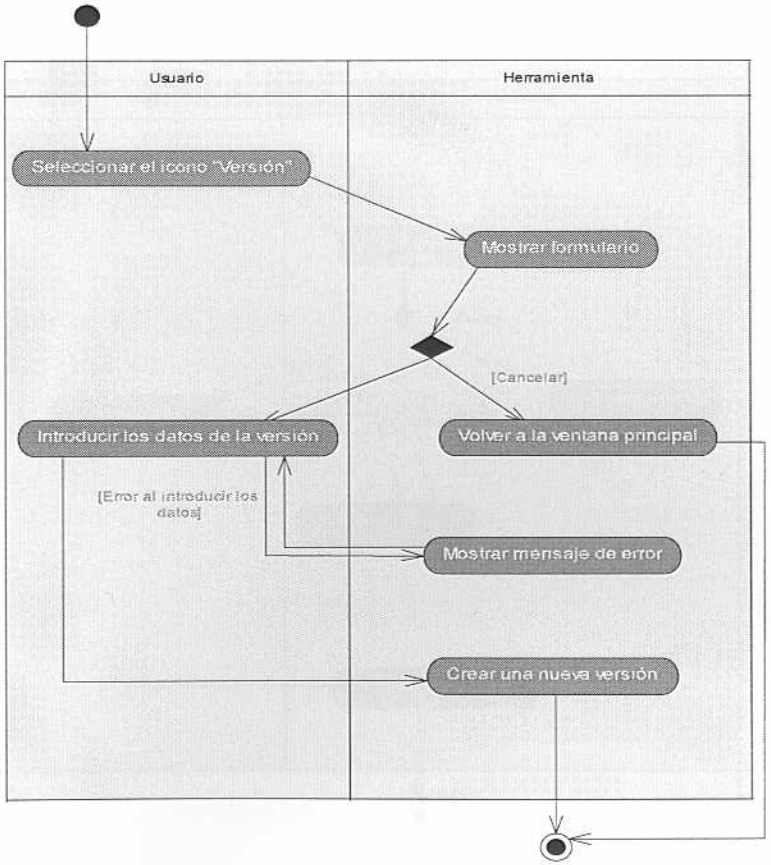
Modificar relación.



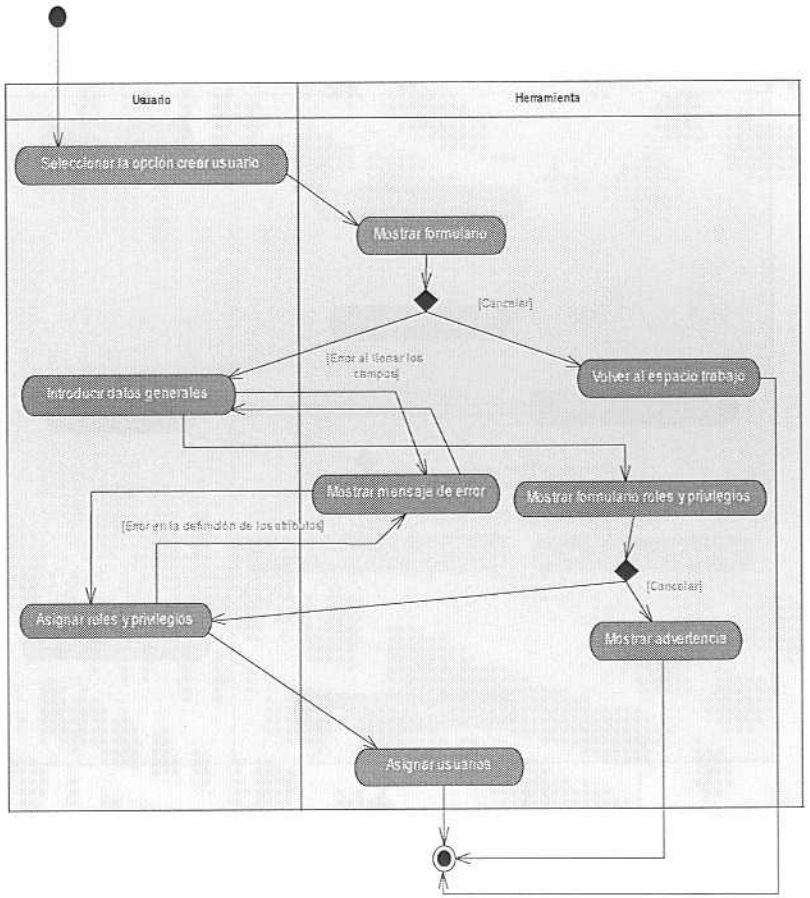
Modificar Dominio.



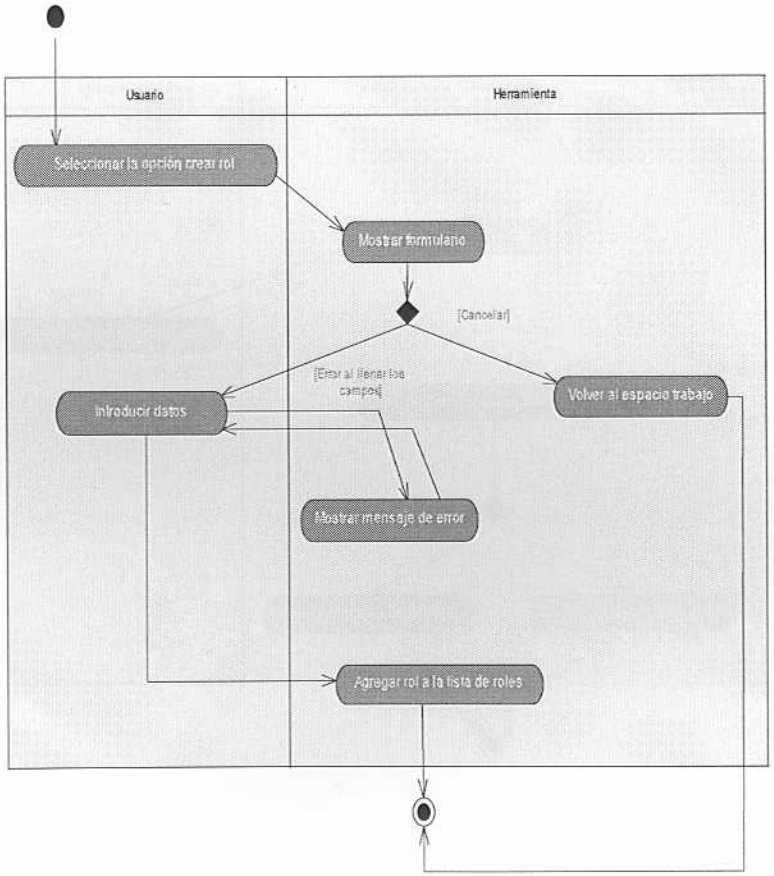
Crear Versión.



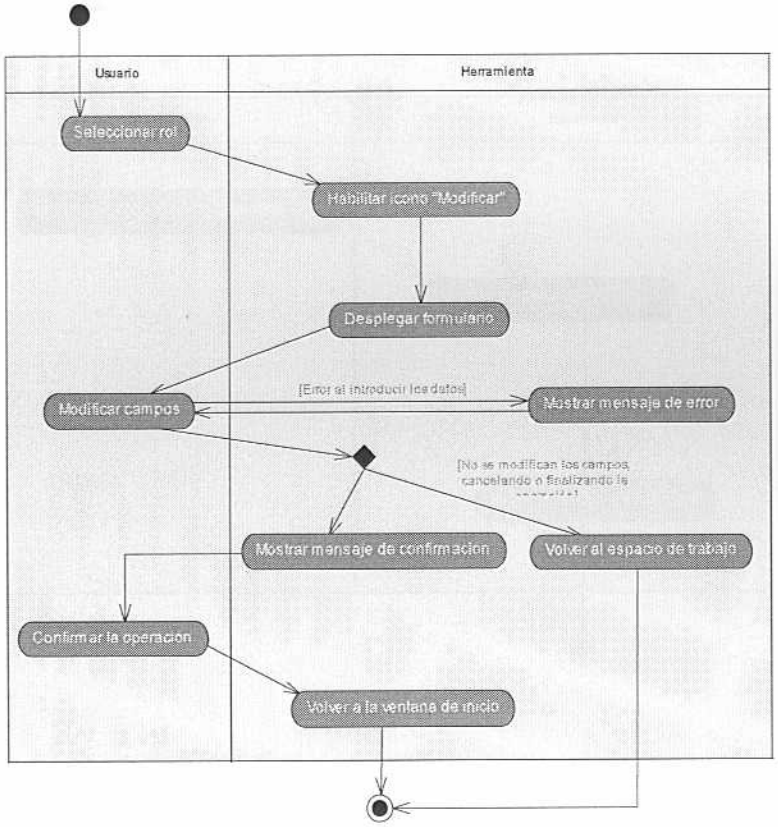
Crear Usuario.



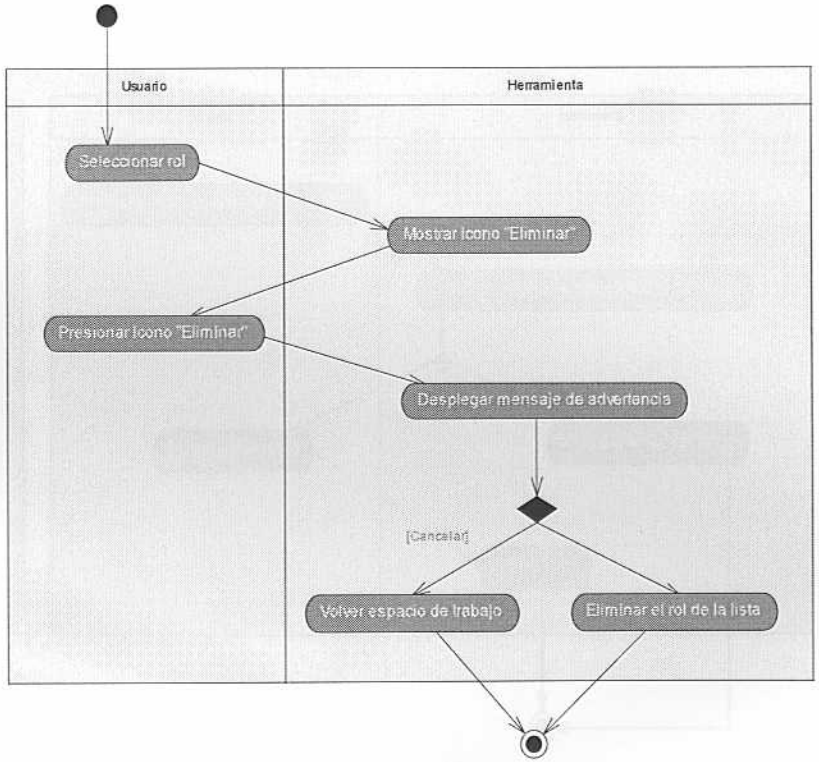
Crear Rol.



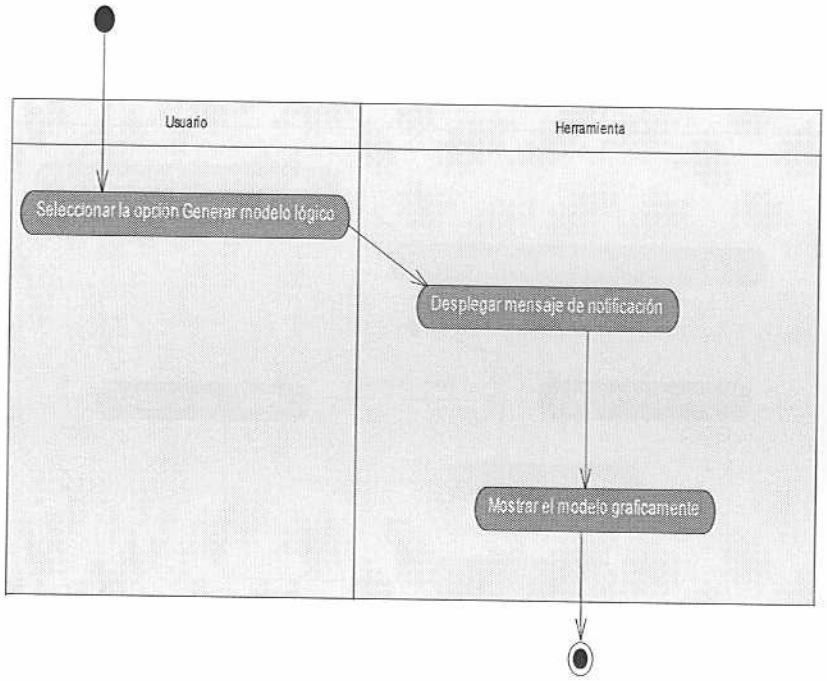
Modificar Rol.



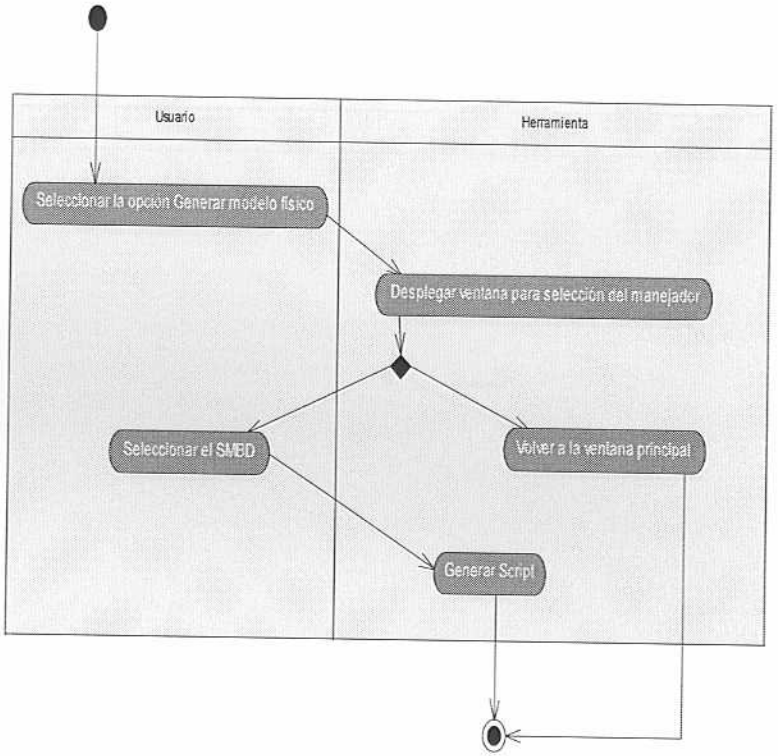
Eliminar Rol.



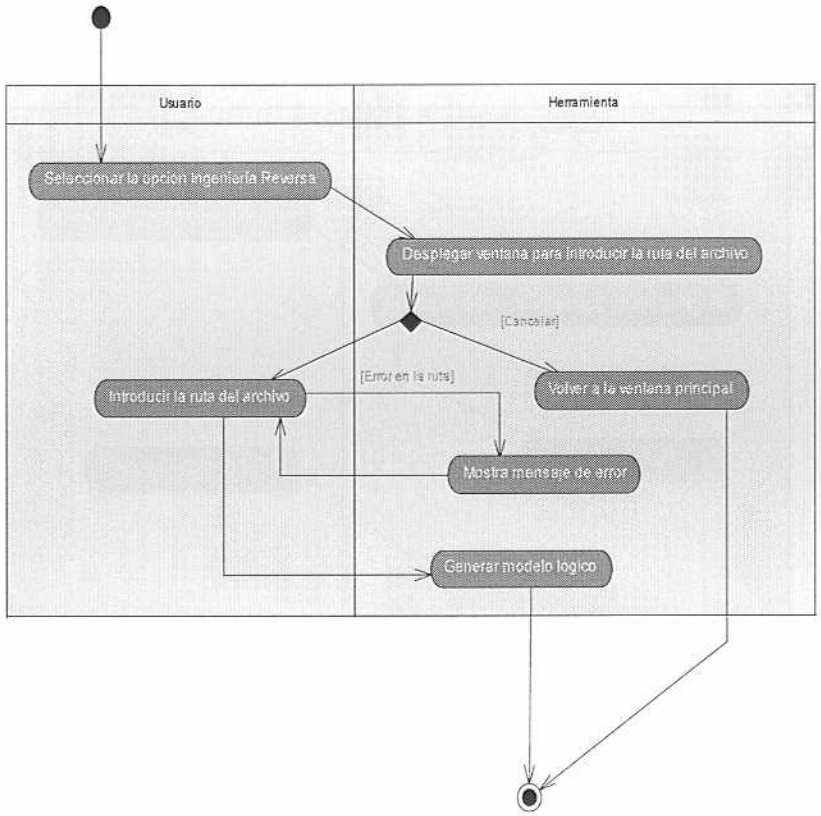
Generar modelo lógico.



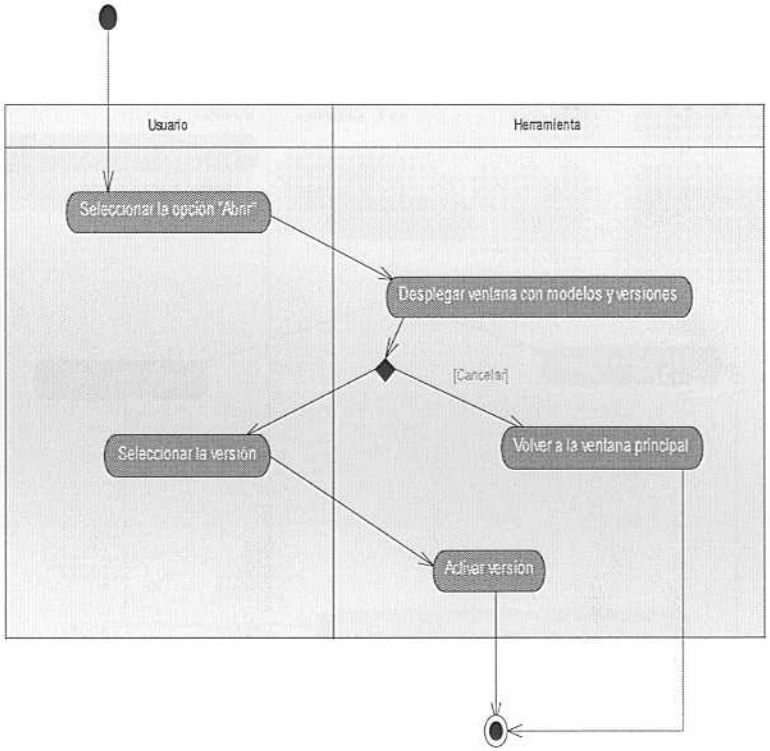
Generar modelo físico.



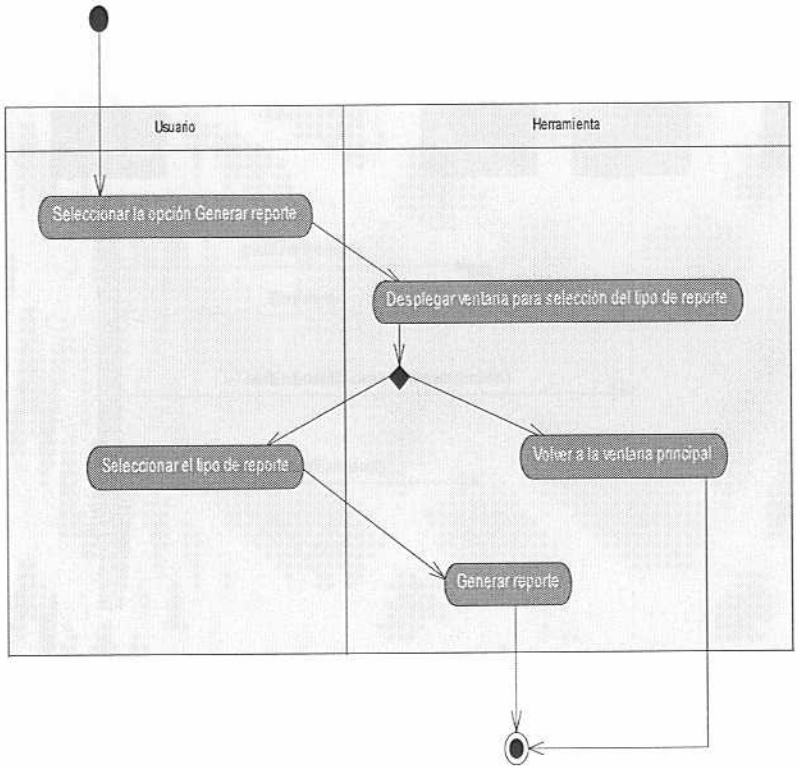
Realizar Ingeniería Reversa.



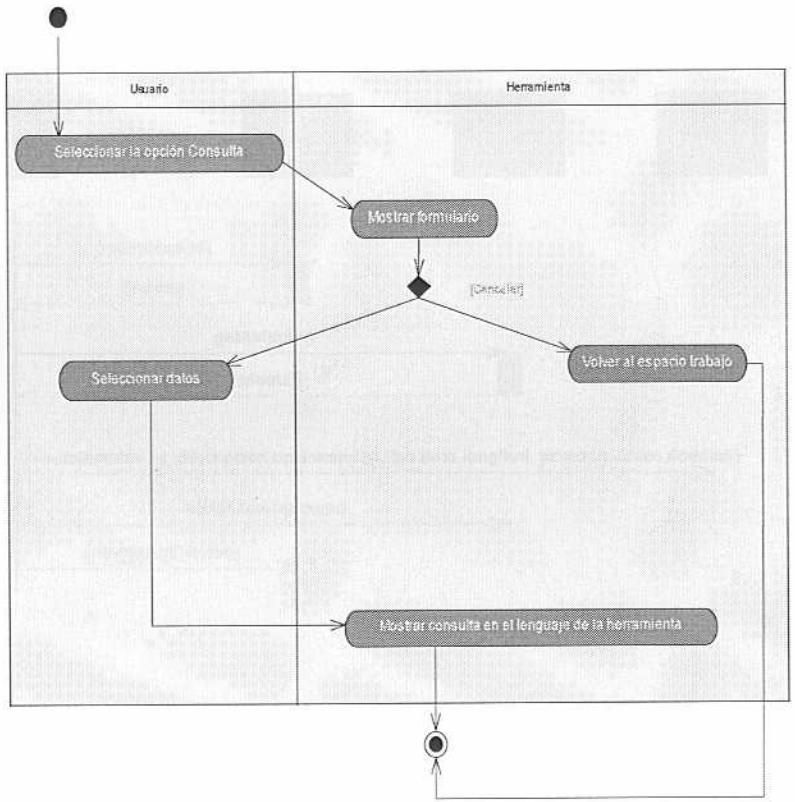
Recuperar Versión.



Generar Reporte

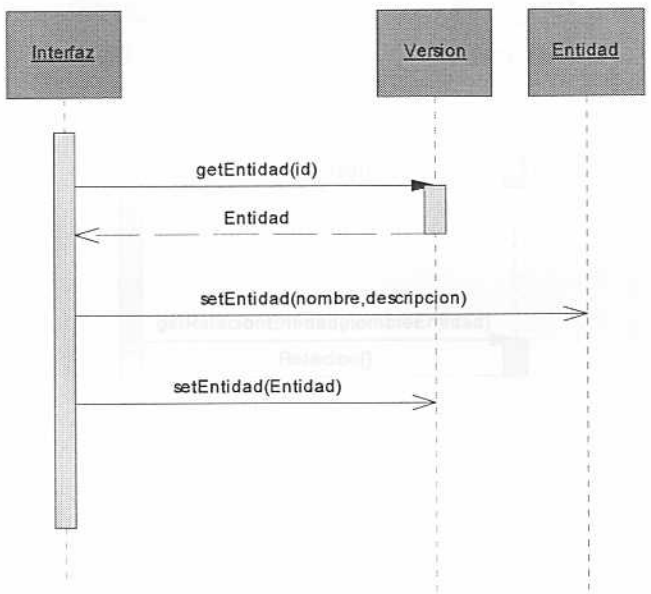


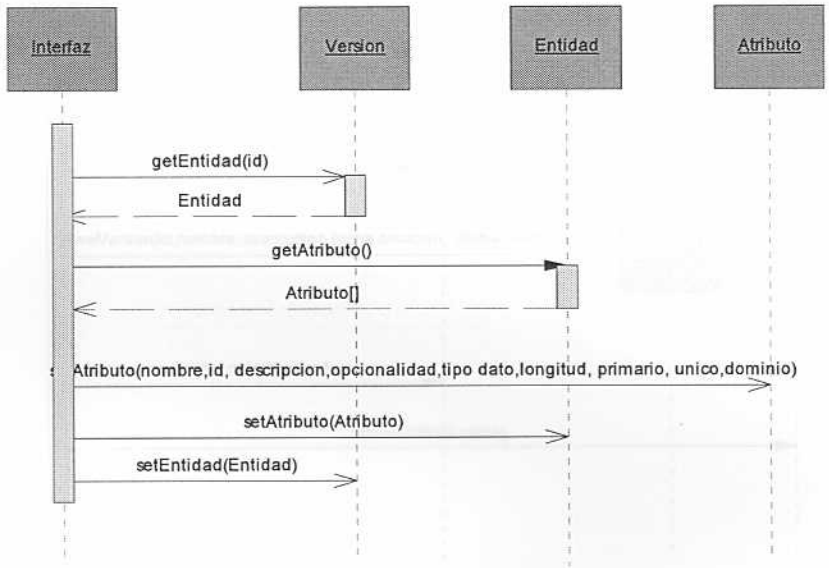
Realizar Consultas



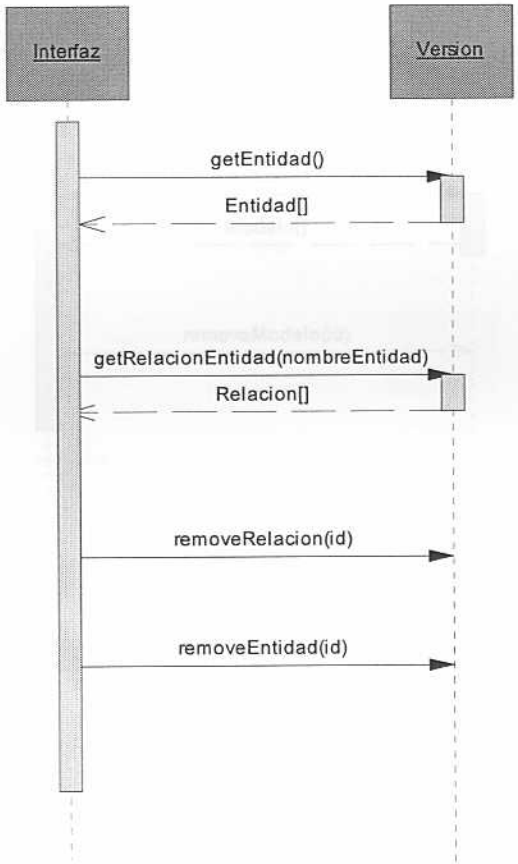
APÉNDICE F. DIAGRAMAS DE SUCUENCIA

Modificar Entidad.

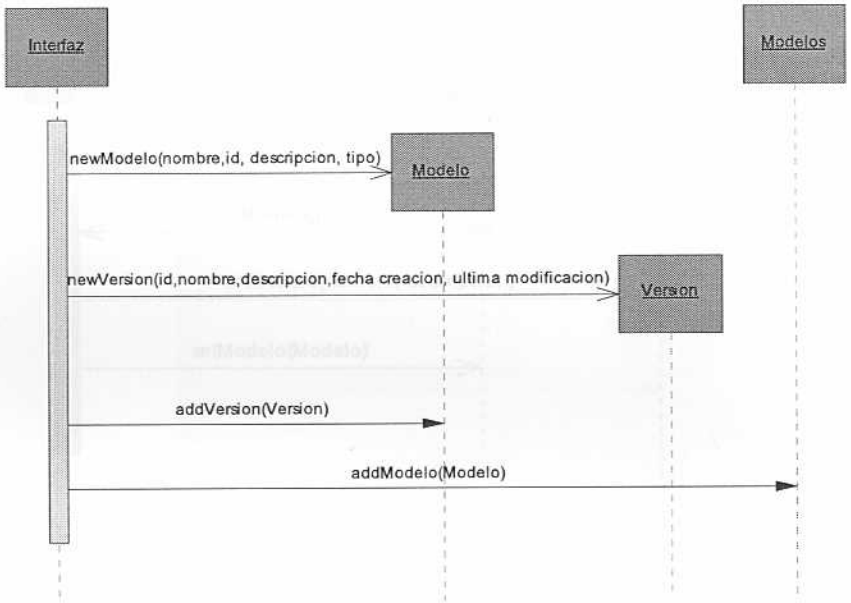




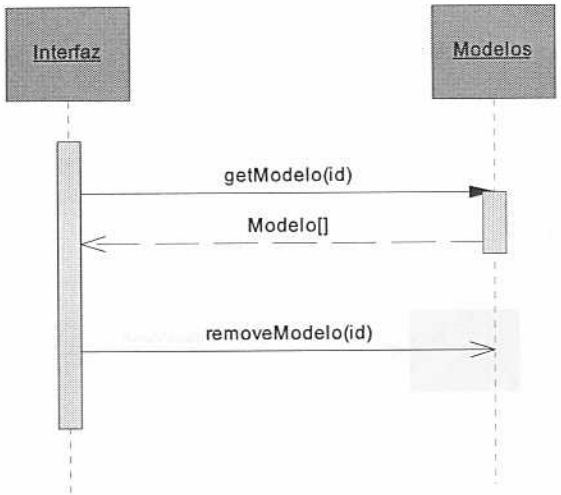
Eliminar Entidad.



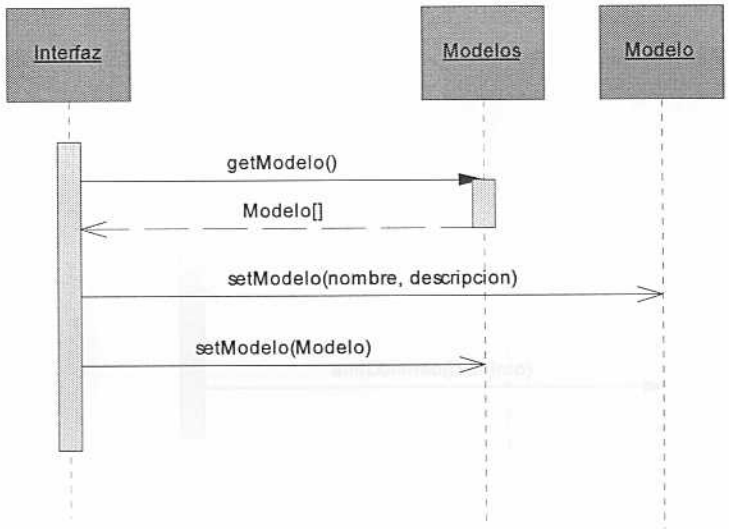
Crear Modelo.



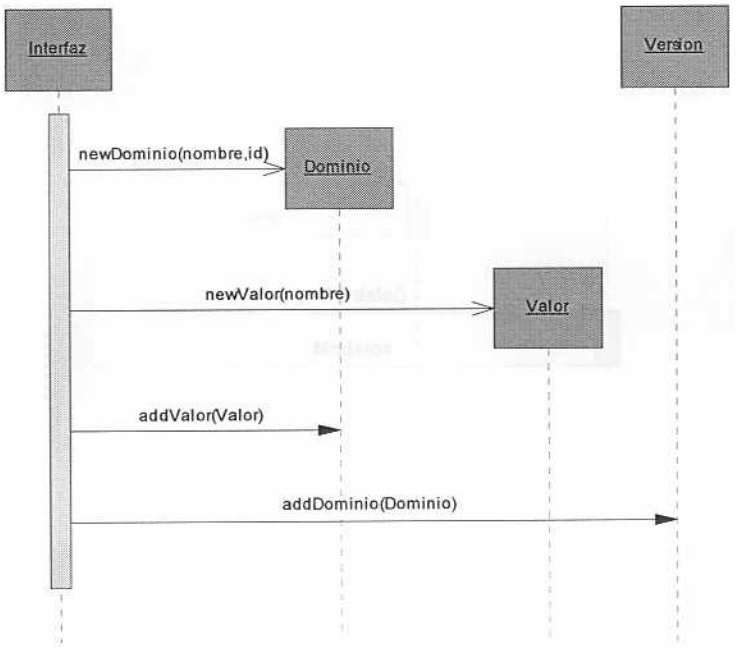
Eliminar Modelo.



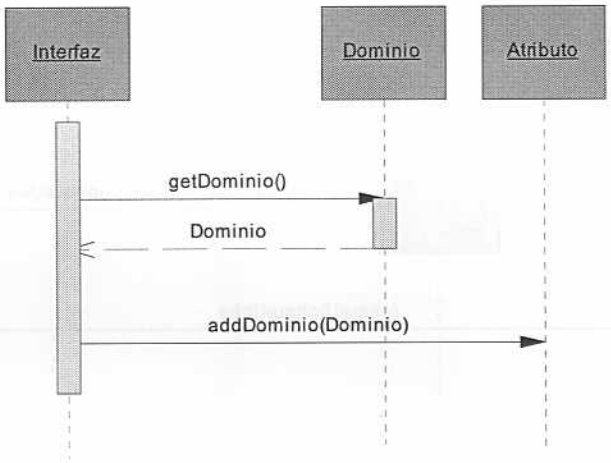
Modificar Modelo.



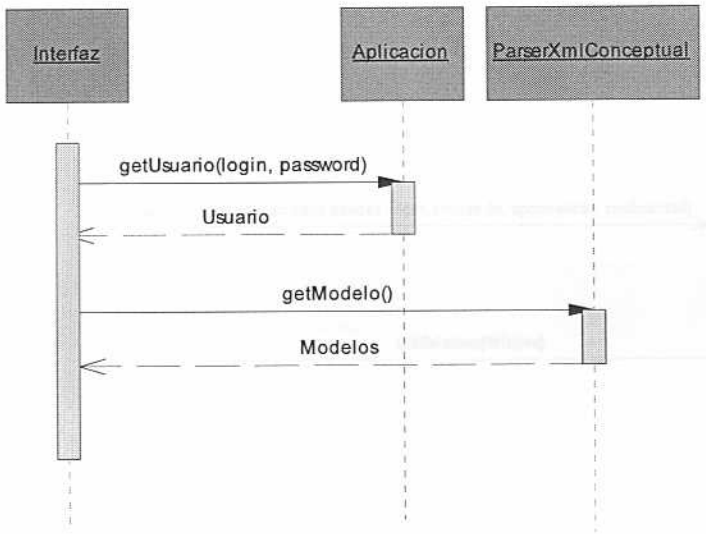
Crear Dominio.



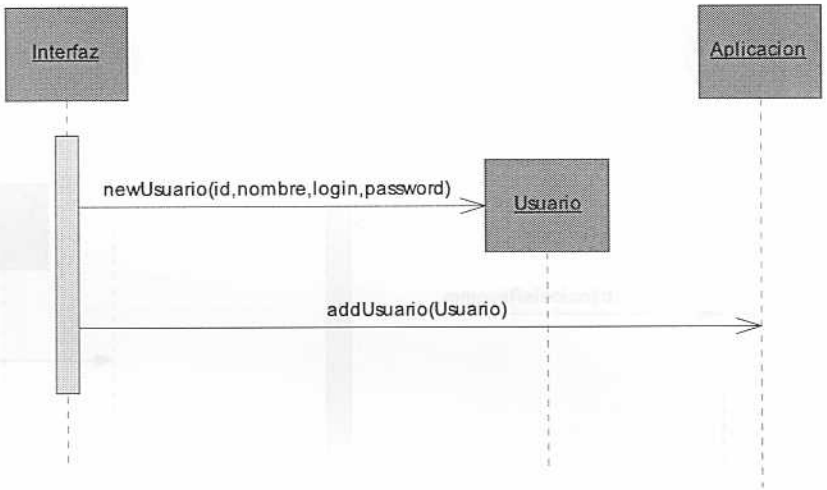
Asignar dominio a atributo



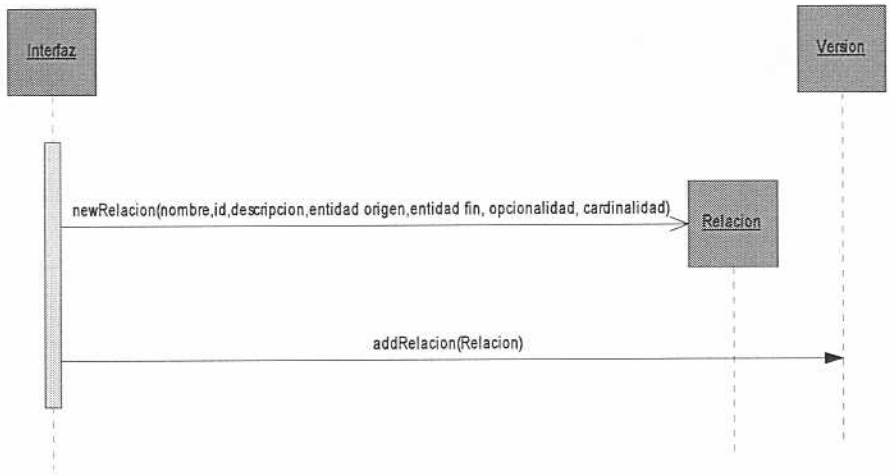
Autentificarse en el sistema



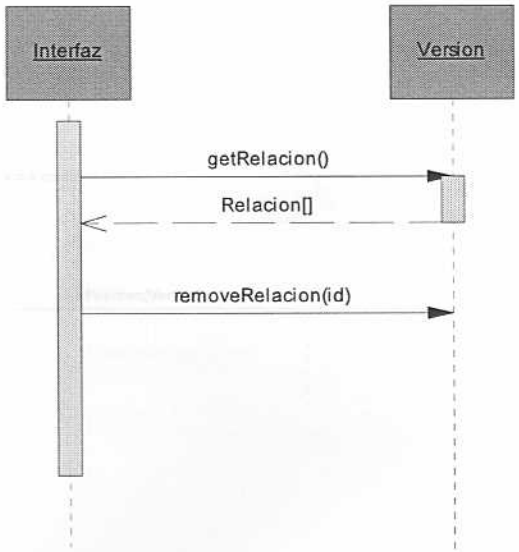
Registrar usuarios



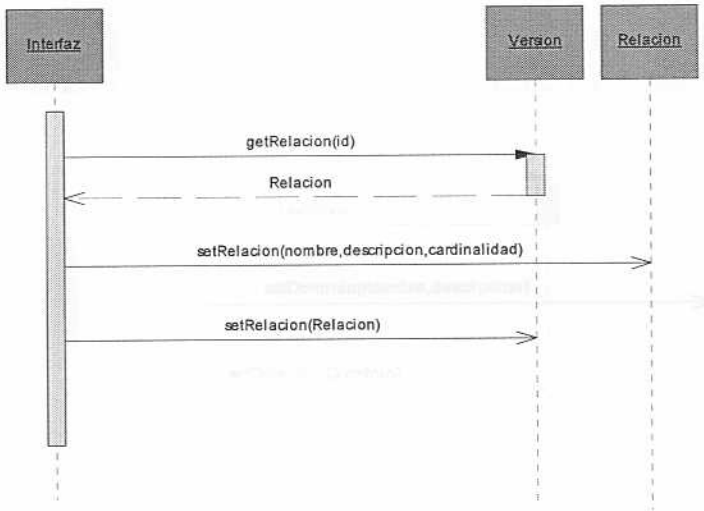
Crear relación



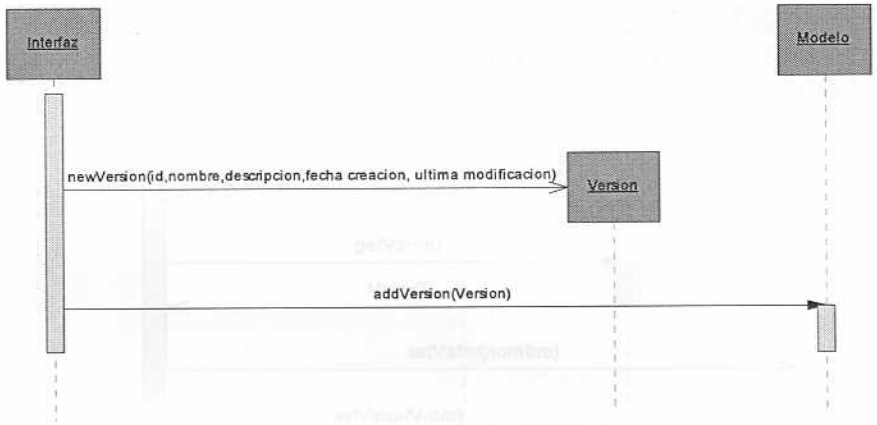
Eliminar Relación



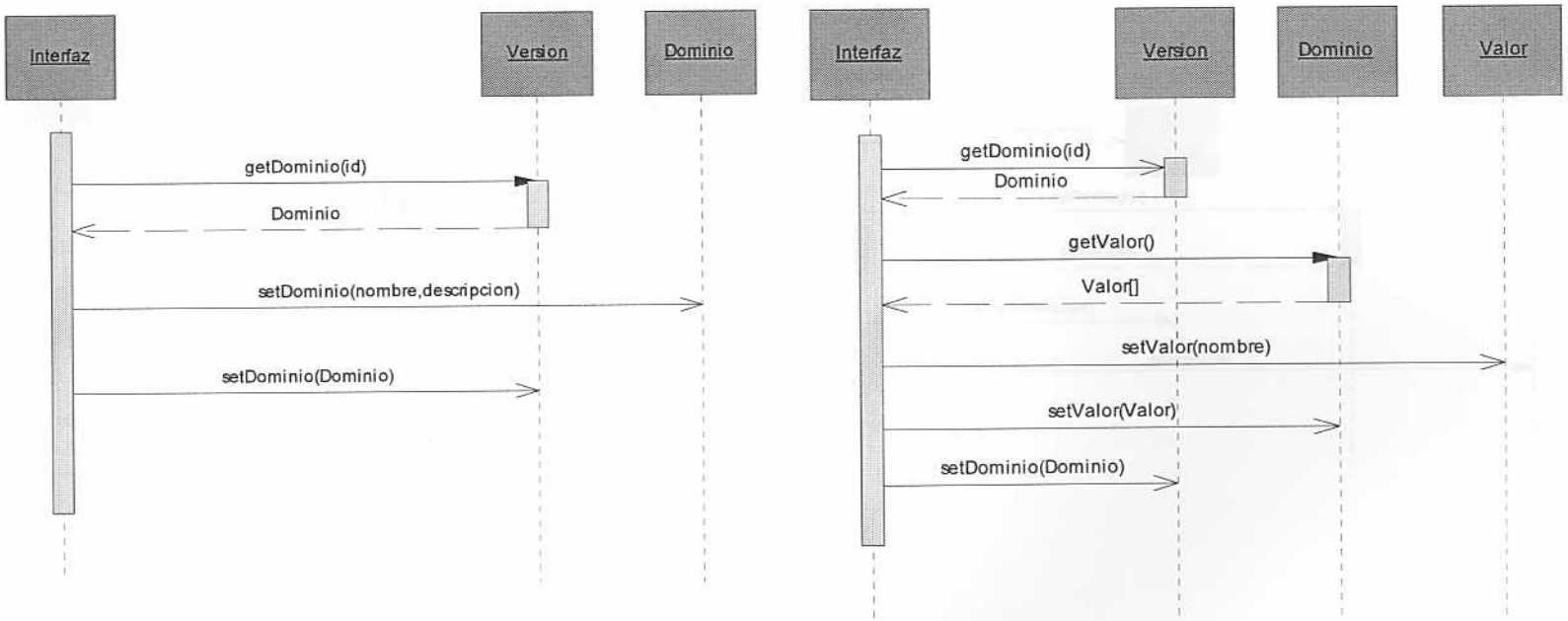
Modificar relación



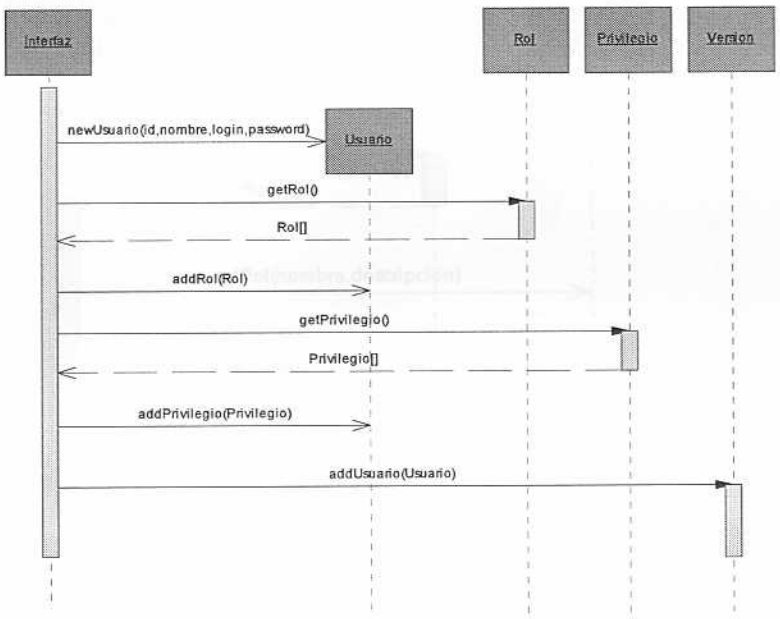
Crear Versión



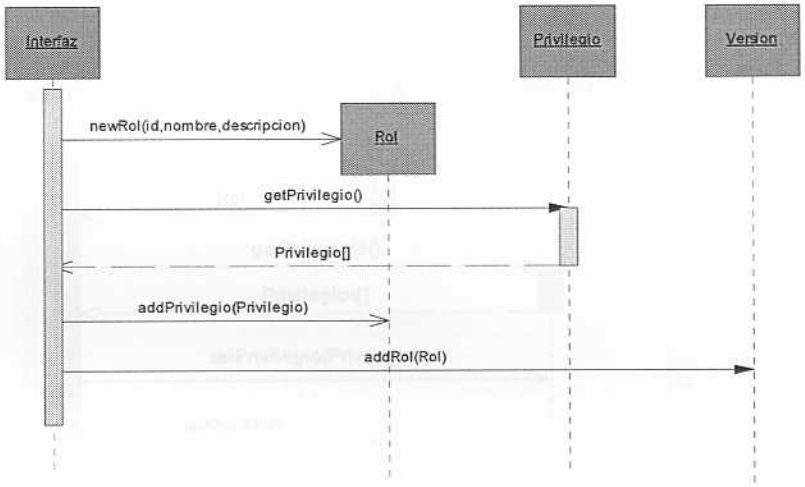
Modificar dominio



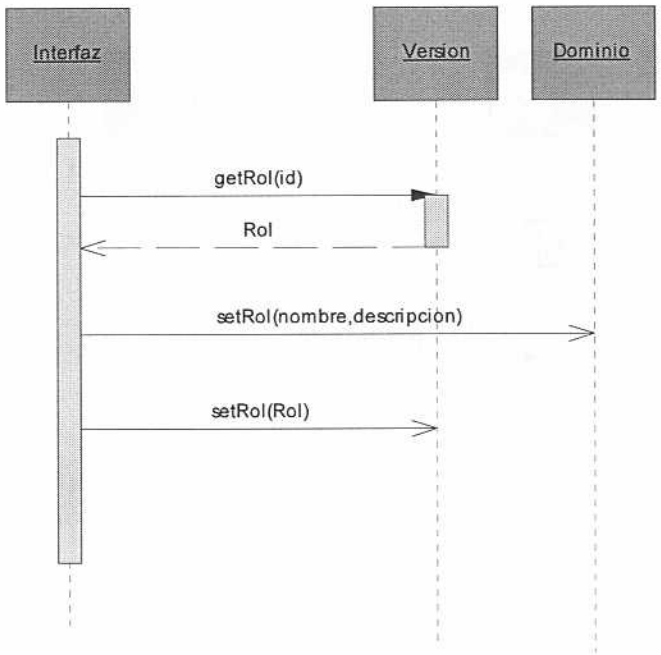
Crear Usuario

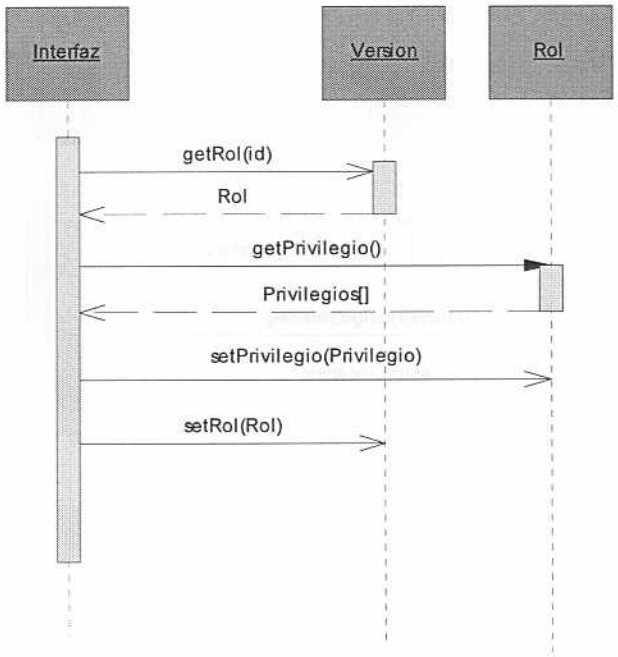


Crear Rol

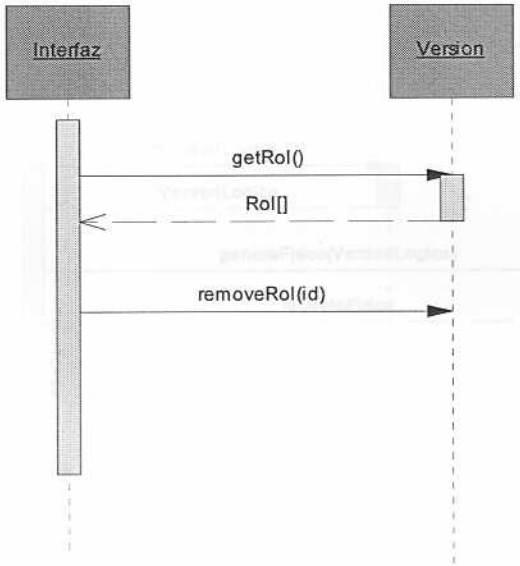


Modificar Rol

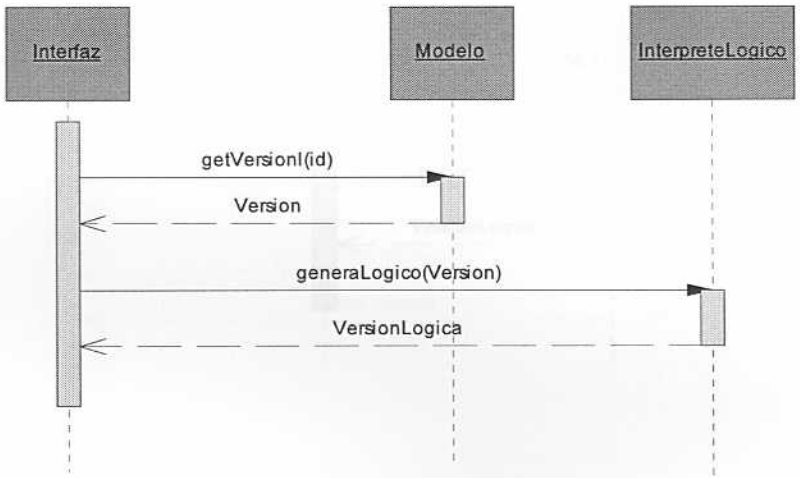




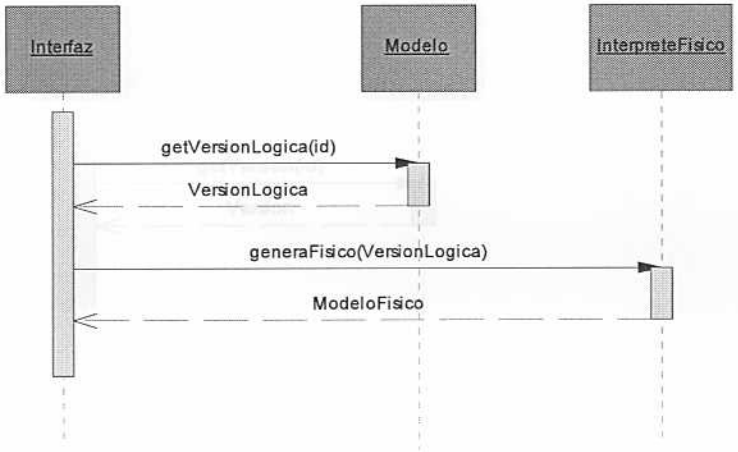
Eliminar Rol



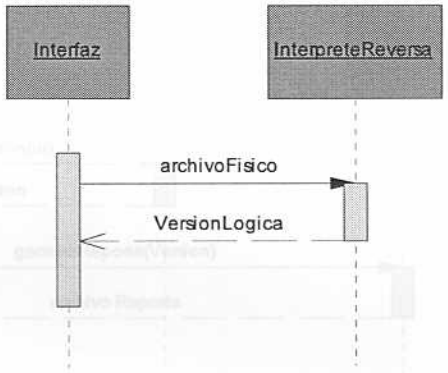
Generar modelo lógico



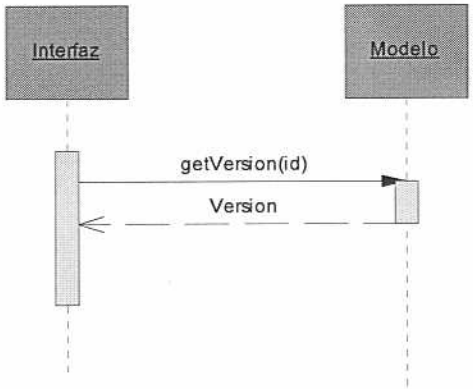
Generar modelo físico



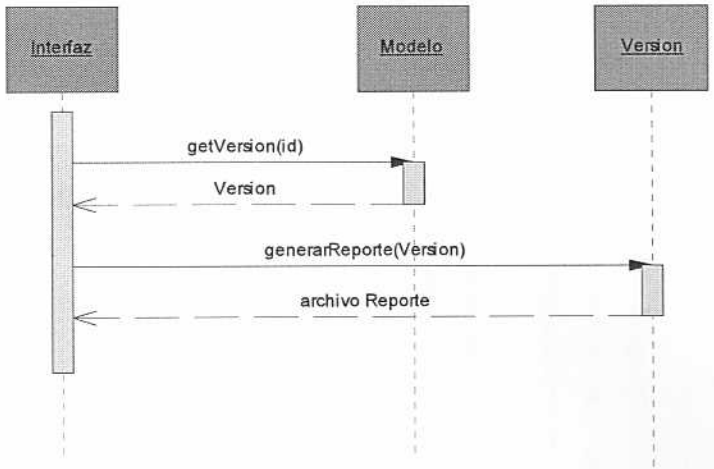
Realizar Ingeniería Reversa.



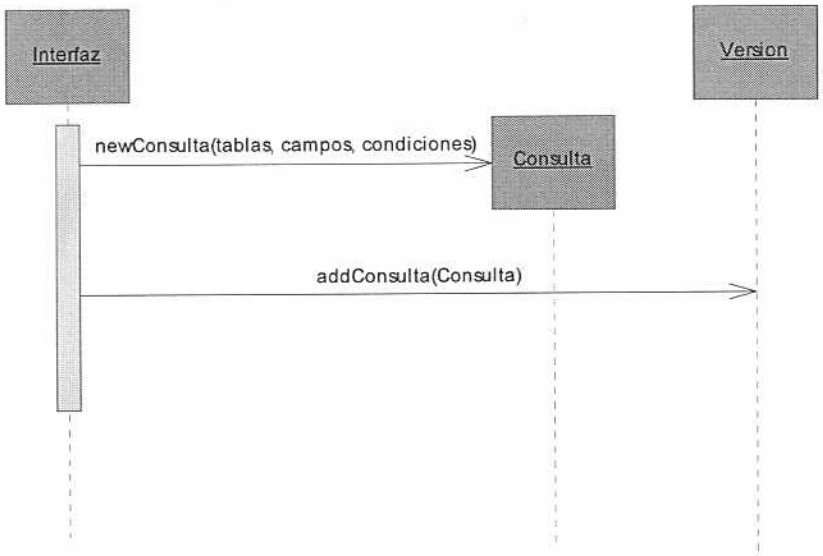
Recuperar Versión



Generar Reportes



Realizar Consulta



APÉNDICE G. XML & DOM

G.1. XML

XML, es el estándar de Extensible Markup Language. No es más que un conjunto de reglas para definir etiquetas semánticas que organizan un documento en diferentes partes. XML es un metalenguaje que define la sintaxis utilizada para definir otros lenguajes de etiquetas estructurados.

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como lenguaje de bajo nivel (a nivel de aplicación, no de programación) para intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo, y casi cualquier cosa que podamos pensar (Osmosis Latina, 2005).

G.1. 1.Objetivos de XML.

Los objetivos de XML se pueden resumir en:

- XML debe ser directamente utilizable sobre Internet.
- XML debe soportar una amplia variedad de aplicaciones.
- XML debe ser compatible con SGML.
- Debe ser fácil la escritura de programas que procesen documentos XML.
- El número de características opcionales en XML debe ser absolutamente mínima, idealmente cero.
- Los documentos XML deben ser legibles por humanos y razonablemente claros.
- El diseño de XML debe ser preparado rápidamente.
- El diseño de XML debe ser formal y conciso.
- Los documentos XML deben ser fácilmente creables.
- La concisión en las marcas XML es de mínima importancia.

Esta especificación, junto con los estándares asociados (Unicode e ISO/IEC 10646 para caracteres, Internet RFC 1766 para identificación de lenguajes, ISO 639 para códigos de nombres de lenguajes, e ISO 3166 para códigos de nombres de países), proporciona toda la información necesaria para entender la Versión 1.0 de XML y construir programas de computador que los procesen.

G.1.2. Características de XML.

- Es una arquitectura más abierta y extensible. No se necesita versiones para que puedan funcionar en futuros navegadores. Los identificadores pueden crearse de manera simple y ser adaptados en el acto en internet/intranet por medio de un validador de documentos (parser).
- Mayor consistencia, homogeneidad y amplitud de los identificadores descriptivos del documento con XML (los RDF Resource Description FrameWork), en comparación a los atributos de la etiqueta <META> del HTML.
- Integración de los datos de las fuentes mas dispares. Se podrá hacer el intercambio de documentos entre las aplicaciones tanto en el propio PC como en una red local o extensa.
- Datos compuestos de múltiples aplicaciones. La extensibilidad y flexibilidad de este lenguaje nos permitirá agrupar una variedad amplia de aplicaciones, desde páginas web hasta bases de datos.
- Gestión y manipulación de los datos desde el propio cliente web.
- Los motores de búsqueda devolverán respuestas más adecuadas y precisas, ya que la codificación del contenido web en XML consigue que la estructura de la información resulte más accesible.
- Se desarrollarán de manera extensible las búsquedas personalizables y subjetivas para robots y agentes inteligentes. También conllevará que los clientes web puedan ser más autónomos para desarrollar tareas que actualmente se ejecutan en el servidor.
- Se permitirá un comportamiento más estable y actualizable de las aplicaciones web, incluyendo enlaces bidireccionales y almacenados de forma externa (El famoso epígrafe "404 file not found" desaparecerá).
- El concepto de "hipertexto" se desarrollará ampliamente (permitirá denominación independiente de la ubicación, enlaces bidireccionales, enlaces que pueden especificarse y gestionarse desde fuera del documento, hiperenlaces múltiples, enlaces agrupados, atributos para los enlaces, etc. Creado a través del Lenguaje de enlaces extensible (XLL).
- Exportabilidad a otros formatos de publicación (papel, web, cd-rom, etc.). El documento maestro de la edición electrónica podría ser un documento XML que se integraría en el formato deseado de manera directa.

Se puede establecer un cuadro-resumen con algunas de las diferencias significativas con respecto a los otros lenguajes que hemos mencionado:

	HTML/DHTML	XML	SGML
Gramática	Fija y no ampliable	Extensible	Extensible
Estructura	Monolítica	Jerárquica	Jerárquica
Nº de marcas	Fijas	Sin límite	Sin límite
Complejidad	Baja	Mediana	Alta
Diseño de páginas	Fijado por tags. Etiquetas con atributos CSS en DHTML	CSS o XSL	DSSSL
Enlaces	Simples enlaces	Poderosos enlaces (XLL)	HyTime
Exportabilidad (formatos/aplicaciones)	No	Sí	Sí
Validación	Sin validación	Pueden validarse	Obligatorio DTD
Búsquedas	Simple y a veces resuelta por *scripts* o CGI	Potente busqueda. Con capacidad para personalizarla	Son posibles potentes búsquedas.
Indización/Catalogación de páginas web	Sólo lo permite los atributos de la etiqueta <META>, e implementaciones como DC.	Una descripción abierta y personalizable con el RDF.	Algún proyecto como TEI, DLI, etc.

Tabla 17. Comparación entre HTML, XML y SGML.
Fuente: Elaborado por los autores. Adaptado de (García, María. 2004).

G.1.3.Estructura del XML

Un documento XML tiene dos estructuras, una **lógica** y otra **física**. Físicamente, el documento está compuesto por unidades llamadas entidades. Una entidad puede hacer referencia a otra entidad, causando que esta se incluya en el documento. Cada documento comienza con una entidad documento, también llamada raíz. Lógicamente, el documento está compuesto de declaraciones, elementos, comentarios, referencias a caracteres e instrucciones de procesamiento, todos los cuales están indicados por una marca explícita. Las estructuras lógica y física deben encajar de manera adecuada:

Los documentos XML se dividen en dos grupos, documentos bien formados y documentos válidos.

- **Bien formados:** Son todos los que cumplen las especificaciones del lenguaje respecto a las reglas sintácticas sin estar sujetos a unos elementos fijados en un DTD. De hecho los documentos XML deben tener una estructura jerárquica muy estricta y los documentos bien formados deben cumplirla.

- **Válidos:** Además de estar bien formados, siguen una estructura y una semiautentica determinada por un DTD: sus elementos y sobre todo la estructura jerárquica que define el DTD, además de los atributos, deben ajustarse a lo que el DTD dicte.

Ejemplo.

```
<?xml version=" 1.0 " encoding=" UTF-8 " standalone= " yes "?>
<ficha>
<nombre> Jose </nombre>
<apellido> Martinez </apellido>
<direccion> Calle 60, San Bernardino </direccion>
</ficha>
```

El *metalenguaje* XML consta de cuatro especificaciones (el propio XML sienta las bases sintácticas y el alcance de su implementación):

DTD (*Document Type Definition*): Definición del tipo de documento. Es, en general, un archivo/s que encierra una definición formal de un tipo de documento y , a la vez, especifica la estructura lógica de cada documento. Define tanto los elementos de una página como sus atributos. El DTD del XML es opcional. En tareas sencillas no es necesario construir una DTD, entonces se trataría de un documento "bien formado"(*well-formed*) y si lleva DTD será un documento "validado" (*valid*).

XSL (*eXtensible Stylesheet Language*): Define o implementa el lenguaje de estilo de los documentos escritos para XML. Desde el verano de 1997 varias empresas informáticas como Arbortext, Microsoft e Inso vienen trabajando en una propuesta de XSL (antes llamado "xml-style") que presentaron a W3C. Permite modificar el aspecto de un documento. Se puede lograr múltiple columnas, texto girado, orden de visualización de los datos de una tabla, múltiples tipos de letra con amplia variedad en los tamaños. Este estándar está basado en el lenguaje de semántica y especificación de estilo de documento (DSSSL, *Document Style Semantics and Specification Language*, ISO/IEC 10179) y, por otro lado, se considera más potente que las hojas de estilo en cascada (CSS, *Cascading Style Sheets*), usado en un principio con el lenguaje DHTML. "Se espera que el CSS sea usado para visualizar simples estructuras de documentos XML (actualmente se ha conseguido mayor integración en XML con el protocolo CSS2 (*Cascading Style Sheets, level 2*) ofreciendo nuevas formas de composición y una más rápida visualización) y, por otra parte, XSL pueda ser utilizado donde se requiera más potencia de diseño como documentos XML que encierran datos estructurados (tablas, organigramas, etc.)[2]".

XLL (*eXtensible Linking Language*): Define el modo de enlace entre diferentes enlaces. Se considera que es un subconjunto de HyTime (*Hipermedia/Timed-based structuring Language* o Lenguaje de estructuración

hipermedia/basado en el tiempo, ISO 10744) y sigue algunas especificaciones del TEI (*Text Encoding Initiative* o Iniciativa de codificación de texto). Desde marzo de 1998 el W3C trabajo en los enlaces y direccionamientos del XML. Provisionalmente se le renombró como *Xlink* y a partir de junio se le denomina XLL. Este lenguaje de enlaces extensible tiene dos importantes componentes: *Xlink* y el *Xpointer*. Va más allá de los enlaces simples que sólo soporta el HTML. Se podrá implementar con enlaces extendidos. Jon Bosak establece los siguientes mecanismos hipertextuales que soportará esta especificación:

- Denominación independiente de la ubicación.
- Enlaces que pueden ser también bidirecccionales.
- Enlaces que pueden especificarse y gestionarse desde fuera del documento a los que se apliquen (Esto permitirá crear en un entorno intranet/extranet un banco de datos de enlaces en los que se puede gestionar y actualizar automáticamente. No habrá más errores del tipo "404 Not Found").
- Hiperenlaces múltiples (anillos, múltiples ventanas, etc.).
- Enlaces agrupados (múltiples orígenes).
- Transclusión (el documento destino al que apunta el enlace aparece como parte integrante del documento orígen del enlace).
- Se pueden aplicar atributos a los enlaces (tipos de enlaces).

XUA (*XML User Agent*): Estandarización de navegadores XML. Todavía está en proceso de creación de borradores de trabajo. Se aplicará a los navegadores para que compartan todos las especificaciones XML.

G.2. DOM

DOM ("Document Object Model") es solamente una *especificación* definida por el "World Wide Web Consortium", es muy similar al J2EE de Sun, ya que permite a diversas compañías u organizaciones definir **"parsers"** alrededor de esta especificación. La especificación DOM más reciente es 2.0 y la gran mayoría de los "parsers" DOM disponibles ya cumplen con ella.

DOM genera un **árbol jerárquico en memoria** del documento o información en XML, basándose del documento anterior cada elemento `<Amigos>` , `<Nombre>` , `<Apellido>` ..etc. es considerado un **nodo** dentro del árbol. Este **árbol jerárquico de información en memoria** permite que a través del **"parser** sea manipulada la información, las ventajas serian las siguientes:

- Puede ser agregado un *nodo* (Información) en cualquier punto del árbol.

- Puede ser eliminada información de un *nodo* en cualquier punto del árbol.
- Lo anterior se ejecuta *sin incurrir* en las penalidades o limitaciones de manipular un archivo de alguna otra manera.

Debido a que DOM es solo una *especificación* existen diversos "parsers" DOM.

Un detalle notorio de cualquier "Parser" es que la mayoría están escritos en Java , esta no es ninguna coincidencia ya que Java es uno de los lenguajes que permite mayor portabilidad entre Sistemas Operativos. Ahora bien, a pesar de esta portabilidad en Java, DOM es solo una especificación y por ende existen diversas

El utilizar un "Parser" implica aprender a utilizar sus clases | funciones o API ("Application Programming Interface"), *ciertas* clases o funciones difieren un poco en los diversos "Parsers" , por eso se recomienda (si utiliza Java) que los programas o aplicaciones que requieran de las funcionalidades de un "Parser" sean diseñadas alrededor de JAXP ("Java API for XML Processing").

APÉNDICE H. ESQUEMAS XML DE ARCHIVOS DE ALMACENAMIENTO DE LA HERRAMIENTA.

H.1.Archivos de sintaxis para SMDB.

H.1.1. ORACLE

```
<?xml version="1.0"?>
        <config nombreManejador="ORACLE">
        <sentencia>
           <create>
                <createTable>
                <abreTabla>CREATE TABLE
          %NOMBRE_TABLA%(</abreTabla>
                <columna> %NOMBRE_ATRIBUTO%
          %TIPO_DATO%  (%LONGITUD%)
                %OPCIONALIDAD%</columna>
                        <cierraTabla>);</cierraTabla>
                </createTable>
                <indice>CREATE INDEX
          %NOMBRE_RESTRICCION_UNIQUE% ON

          %NOMBRE_TABLA%(ATRIBUTO);</indice>
                 <consulta>
                        <select>SELECT
          %LISTA_ATRIBUTOS%</select>
                        <from>FROM
          %LISTA_TABLAS%</from>
                        <where>WHERE
          %LISTA_CONDICIONES%</where>
                 </consulta>
                <user>CREATE USER %NOMBRE_USUARIO%
          IDENTIFIED BY %PASSWORD%;</user>
                  <rol>CREATE ROLE
          %NOMBRE_ROL%;</rol>
          </create>
         <alter>
           <alterAbre>ALTER TABLE %NOMBRE_TABLA%
          ADD(</alterAbre>
           <constraint>
          <pk>CONSTRAINT %NOMBRE_CONSTRAINT%
          PRIMARY KEY(%NOMBRE_ATRIBUTO%)</pk>
               <fk>CONSTRAINT
          %NOMBRE_CONSTRAINT% FOREIGN
               KEY(%NOMBRE_ATRIBUTO%)REFERENCES
               %TABLA_REFERENCIA%(%ATRIBUTO_REFER
          ENCIA%)</fk>
          <ck>CONSTRAINT %NOMBRE_CONSTRAINT%
          CHECK (%NOMBRE_ATRIBUTO% IN
               (%LISTA_VALORES%))</ck>
           </constraint>
           <alterCierra>);</alterCierra>
         </alter>
         <drop>
           <dropTable>DROP TABLE      %NOMBRE_TABLA%
CASCADE
CONSTRAINTS;</dropTable>
                <dropUser>DROP USER
               %NOMBRE_USUARIO%
               CASCADE;</dropUser>
                <dropRole>DROP ROLE
               %NOMBRE_ROL%
               CASCADE;</dropRole>
            </drop>
            <rYp>
                <grants>
                <grantSistema>
               <grantPri_to_Usr>GRANT
               %LISTA_PRIVILEGIO% TO
               %LISTA_NOMBRE_USUARIO% $WITH
                    ADMIN OPTION$;
               </grantPri_to_Usr>
               <grantRol_to_Usr>GRANT
               %LISTA_NOMBRE_ROL% TO
               %LISTA_NOMBRE_USUARIO% $WITH
               ADMIN OPTION$;</grantRol_to_Usr>
               <grantPri_to_Rol>GRANT
               %LISTA_NOMBRE_PRIVILEGIO% TO
               %LISTA_NOMBRE_ROL% $WITH
                    ADMIN
               OPTION$;</grantPri_to_Rol>
               </grantSistema>
               <grantObjeto>
                    <grantO_to_Usr>GRANT
               %LISTA_PRIVILEGIO_SISTEMA% ON
               %TABLA% TO
                    %LISTA_NOMBRE_USUARIO%
               $WITH ADMIN
               OPTION$;</grantO_to_Usr>
               <grantO_to_Rol>GRANT
               %LISTA_PRIVILEGIO_SISTEMA% ON
               %TABLA% TO
                    %LISTA_NOMBRE_ROL% $WITH
               ADMIN OPTION$;</grantO_to_Rol>
              </grantObjeto></grants>
              <revoke>
               <revokePri_to_Usr>REVOKE
               %LISTA_PRIVILEGIO% TO
               %LISTA_NOMBRE_USUARIO%;
                    </revokePri_to_Usr>
               <revokeRol_to_Usr>REVOKE
               %LISTA_NOMBRE_ROL% TO
                    %LISTA_NOMBRE_USUARIO%;</re
               vokeRol_to_Usr>
               <revokePri_to_Rol>REVOKE
               %LISTA_NOMBRE_PRIVILEGIO% TO
                    %LISTA_NOMBRE_ROL%;</revokeP
               ri_to_Rol>
             </revoke>
               </rYp>
            </sentencia></config>
```

H.1.2. SYBASE

```
<?xml version="1.0"?>
        <config nombreManejador="SYBASE">
          <sentencia>
          <create>
                    <createTable>
                    <abreTabla>CREATE TABLE
        %NOMBRE_TABLA%(</abreTabla>
                    <columna>
        %NOMBRE_ATRIBUTO% %TIPO_DATO%
        (%LONGITUD%)
            %OPCIONALIDAD%</columna>
                    <cierraTabla>);</cierraTabla>
         </createTable>
                    <indice>CREATE INDEX
        %NOMBRE_RESTRICCION_UNIQUE% ON

        %NOMBRE_TABLA%(ATRIBUTO); </indice>
            <consulta>
            <select>SELECT
        %LISTA_ATRIBUTOS%</select>
            <from>FROM %LISTA_TABLAS%</from>
            <where>WHERE
        %LISTA_CONDICIONES%</where>
             </consulta>
        </create>
<alter>
          <constraint>
          <pk>PRIMARY
          KEY(%NOMBRE_ATRIBUTO%)</pk>
          <fk>ALTER TABLE %NOMBRE_TABLA% ADD
          CONSTRAINT %NOMBRE_CONSTRAINT%
              FOREIGN
          KEY(%NOMBRE_ATRIBUTO%)REFERENCES
              %TABLA_REFERENCIA%(%ATRIBUTO_REF
          ERENCIA%);</fk>
          <ck>ALTER TABLE %NOMBRE_TABLA% ADD
          CONSTRAINT %NOMBRE_CONSTRAINT%
              CHECK (%NOMBRE_ATRIBUTO% IN
          (%LISTA_VALORES%));</ck>
            </constraint>
        </alter>
        <drop>
            <dropTable>DROP TABLE
            %NOMBRE_TABLA%;</dropTable>
        </drop>
         </sentencia>
</config>
```

H.1.3 MYSQL

```
<?xml version="1.0"?>

        <config nombreManejador="MYSQL">
        <sentencia>
          <create>
        <createTable>
        <abreTabla>CREATE TABLE
        %NOMBRE_TABLA%(</abreTabla>
            <columna> %NOMBRE_ATRIBUTO%
            %TIPO_DATO%
            (%LONGITUD%)%OPCIONALIDAD%</c
            olumna>
        <cierraTabla>);</cierraTabla>
      </createTable>
<consulta>
          <select>SELECT
          %LISTA_ATRIBUTOS%</select>
          <from>FROM %LISTA_TABLAS%</from>
          <where>WHERE
          %LISTA_CONDICIONES%</where>
        </consulta>
        </create>
         <alter>
         <constraint>
        <pk>PRIMARY KEY
          (%NOMBRE_ATRIBUTO%)</pk>
         </constraint>
         </alter>
        </sentencia>
        </config>
```

H.2.Archivo de configuración de usuarios de la herramienta.

```
<applicationConfig appRoot="""">
    <usersInfo>
        <appUser idUsuario=" " nombreUsuario=" " loginUsuario=" " passwordUsuario=" ">
            <conceptualPath></conceptualPath>
            <logicalPath></logicalPath>
            <physicalPath></physicalPath>
        </appUser>
    </usersInfo>
</applicationConfig>
```

H.3.Modelos de configuración XML conceptual.

```
<?xml version="1.0"?>
<MODELOS idUsuarioModelos="">
    <MODELO idModelo="">
        <nombreModelo></nombreModelo>
        <descripcionModelo></descripcionModelo>
        <tipoModelo></tipoModelo>
        <version idVersion="" nombreVersion=""
        descripcionVersion="" fechaCreacionVersion=""
            fechaModificacionVersion="">
        <ENTIDADES>
            <entidad  nombreEntidad="" idEntidad=""
        descripcionEntidad="">
        <atributos>
            <atributo nombreAtributo="" idAtributo="">
            <descripcionAtributo></descripcionAtributo>
         <opcionalidadAtributo></opcionalidadAtributo>
            <tipoDatoAtributo></tipoDatoAtributo>
            <tamanoAtributo></tamanoAtributo>
            <primario></primario>
            <unico></unico>
            <dominio></dominio>
                    </atributo>
        </atributos>
            </entidad>
</ENTIDADES>
<RELACIONES>
        <relacion nombreRelacion="" idRelacion="">
        <descripcionRelacion></descripcionRelacion>
        <entidadOrigen></entidadOrigen>
        <entidadFin></entidadFin>
        <cardinalidadRelacion></cardinalidadRelacion>
        <opcionalidadRelacion></opcionalidadRelacion>
            </relacion>
        </RELACIONES>

<dominios>
    <dominio nombreDominio="" idDominio="">
        <valor></valor></dominio>
 </dominios>

 <usuario idUsuario="" nombreUsuario="" loginUsuario=""
        passwordUsuario="">
 <privilegios>
        <privilegio nombrePrivilegio=""idPrivilegio=""
            descripcionPrivilegio=""/>
 </privilegios>
 <roles>
<rol nombreRol="" idRol="" descripcionRol="" >
        <privilegios>
            <privilegio nombrePrivilegio="" idPrivilegio=""
            descripcionPrivilegio=""/>
        </privilegios>
        </rol>
 </roles>
 </usuario>
    <consulta nombreConsulta="" idConsulta=""
            descripcionConsulta="">
        <campos>
        <campo nombreTabla=""></campo></campos>
         <tablas>
            <tabla alias=""></tabla>
        </tablas>
            <condiciones>
                <condicion></condicion>
            </condiciones>
        </consulta>
 </version>
</MODELO>
</MODELOS>
```

H.4.Modelos de configuración XML lógicos.

```
<?xml version=""?>
<MODELOS idUsuarioModelos="">
 <MODELO idModelo="">
  <nombreModelo>   </nombreModelo>
  <descripcionModelo>  </descripcionModelo>
  <tipoModelo></tipoModelo>
  <VersionLogica idVersionLogica=""
nombreVersionLogica="" descripcionVersionLogica=""
fechaCreacionVersionLogica=""
    fechaModificacionVersionLogica="">
        <TABLAS>
        <Tabla nombreTabla="" idTabla=""
descripcionTabla="">
        <atributos>
        <atributo nombreAtributo="" idAtributo="">
        <descripcionAtributo></descripcionAtributo>
        <opcionalidadAtributo></opcionalidadAtributo>
        <tipoDatoAtributo></tipoDatoAtributo>
        <tamanoAtributo></tamanoAtributo>
        <dominio></dominio>
        <restricciones>

        <primary idRestriccion="" nombreRestriccion=""/>

        <foraneo idRestriccion="" nombreRestriccion=""
tablaRef="" atribRef=""/>

        <unique idRestriccion="" nombreRestriccion=""/>

        <check idRestriccion="" nombreRestriccion="">

<dominio></dominio>

        </check>
        </restricciones>
        </atributo>
        </atributos>
        </Tabla>

        </TABLAS>
        <dominios>
        <dominio nombreDominio="" idDominio="">
                <valor></valor>
                <valor></valor>
        </dominio>
         </dominios>

        <usuario idUsuario="" nombreUsuario=""
loginUsuario="" passwordUsuario="">
        <privilegios>
        <privilegio nombrePrivilegio="" idPrivilegio=""
descripcionPrivilegio=""/>                    </privilegios>
        <roles>
        <rol nombreRol="" idRol="" descripcionRol="" >
        <privilegios>
        <privilegio nombrePrivilegio="" idPrivilegio=""
descripcionPrivilegio=""/>
        </privilegios>
        </rol>
        </roles>
        </usuario>


        <consulta nombreConsulta="" idConsulta=""
descripcionConsulta="">
        <campos>
        <campo nombreTabla=""></campo>
        </campos>
        <tablas>
        <tabla alias=""></tabla>
        </tablas>
        <condiciones>
        <condicion></condicion>
        </condiciones>
        </consulta>

   </VersionLogica>

  </MODELO>

 </MODELOS>
```

H.5.Archivo SQL generado por la herramienta (Modelos Físicos).

```
-- ------------Archivo de Creacion de Objetos de la Base de Datos------------

-- ------------Datos del Modelo----------------------
-- Nombre de la Version: VERSION_CONCEPTUAL_LOGICA
-- Fecha de Creacion: fecha Creacion
-- Nombre del Manejador: ORACLE
-- ------------------------------------------------

-- ------------ELIMINACION DE OBJETOS------------

DROP TABLE muchos_muchos CASCADE CONSTRAINTS;
```

```
DROP TABLE carnet CASCADE CONSTRAINTS;
DROP TABLE empresa CASCADE CONSTRAINTS;
DROP TABLE departamento CASCADE CONSTRAINTS;
DROP TABLE empleado CASCADE CONSTRAINTS;

-- ------------Creando Tabla: empleado------------

CREATE TABLE empleado(
 nombre_empleado varchar(30) NOT NULL,
 cedula_empleado varchar(8) NOT NULL,
 pasaporte_empleado varchar(8) NOT NULL,
 id_carnet varchar(8) NOT NULL,
 id_departamento varchar(8) NOT NULL,
CONSTRAINT PK_empleado PRIMARY KEY(cedula_empleado,pasaporte_empleado)
);

-- ------------Creando Tabla: departamento------------

CREATE TABLE departamento(
 nombre_departamento varchar(30) NOT NULL,
 id_departamento varchar(8) NOT NULL,
 rif_empresa varchar(8) NOT NULL,
 id_departamento_0 varchar(8) NOT NULL,
CONSTRAINT PK_departamento PRIMARY KEY(id_departamento)
);

-- ------------Creando Tabla: empresa------------

CREATE TABLE empresa(
 nombre_empresa varchar(30) NOT NULL,
 rif_empresa varchar(8) NOT NULL,
CONSTRAINT PK_empresa PRIMARY KEY(rif_empresa)
);

-- ------------Creando Tabla: carnet------------

CREATE TABLE carnet(
 nombre_carnet varchar(30) NOT NULL,
 id_carnet varchar(8) NOT NULL,
CONSTRAINT PK_carnet PRIMARY KEY(id_carnet)
);

-- ------------Creando Tabla: muchos_muchos------------

CREATE TABLE muchos_muchos(
 rif_empresa varchar(8) NOT NULL,
```

```
cedula_empleado varchar(8) NOT NULL,
pasaporte_empleado varchar(8) NOT NULL,
CONSTRAINT PK_muchos_muchos PRIMARY KEY(rif_empresa,cedula_empleado,pasaporte_empleado)
);
-- ------------Creacion de Restricciones------------


-- ------------Alterando Tabla: empleado------------
ALTER TABLE empleado ADD(

CONSTRAINT CHK_empleado_pasaporte_dominio CHECK (pasaporte_empleado IN ('V','E')),
CONSTRAINT FK_pertenece_departamento_departamento FOREIGN KEY(id_departamento)REFERENCES
        departamento(id_departamento),
CONSTRAINT FK_tiene_carnet_carnet FOREIGN KEY(id_carnet)REFERENCES carnet(id_carnet)
);

-- ------------Alterando Tabla: departamento------------
ALTER TABLE departamento ADD(

CONSTRAINT FK_departamento_departamento_0 FOREIGN KEY(id_departamento)REFERENCES
        departamento(id_departamento),
CONSTRAINT FK_departamento_empresa_empresa FOREIGN KEY(rif_empresa)REFERENCES
        empresa(rif_empresa)
);


-- ------------Alterando Tabla: muchos_muchos------------
ALTER TABLE muchos_muchos ADD(

CONSTRAINT FK_muchos_muchos_empresa FOREIGN KEY(rif_empresa)REFERENCES empresa(rif_empresa),
CONSTRAINT FK_muchos_muchos_empleado FOREIGN
        KEY(cedula_empleado,pasaporte_empleado)REFERENCES
        empleado(cedula_empleado,pasaporte_empleado)
);

-- ------------Creacion de Indices------------

CREATE INDEX UNQ_empleado ON empleado(pasaporte_empleado);
CREATE INDEX UNQ_empresa ON empresa(rif_empresa);
CREATE INDEX UNQ_carnet ON carnet(id_carnet);
COMMIT;


-- ------------ELIMINACION DE OBJETOS------------

DROP USER Luis Martinez 4 CASCADE;
```

```
DROP USER Luis Martinez 3 CASCADE;
DROP USER Luis Martinez 2 CASCADE;
DROP USER Luis Martinez 1 CASCADE;

-- ------------Creacion de Objetos------------

CREATE USER Luis Martinez 1 IDENTIFIED BY 111111;
CREATE USER Luis Martinez 2 IDENTIFIED BY 222222;
CREATE USER Luis Martinez 3 IDENTIFIED BY 333333;
CREATE USER Luis Martinez 4 IDENTIFIED BY 444444;

-- ------------Asignacion de Roles y Privilegios------------

CREATE ROLE Rol1;
CREATE ROLE Rol2;
CREATE ROLE Rol3;
CREATE ROLE Rol4;

-- ------------Asignacion de Privilegios a Roles------------

GRANT Pri1 TO Rol3 ;
GRANT Pri4,Pri5 TO Rol2 ;
GRANT Pri1,Pri2,Pri3,Pri4 TO Rol1 ;

-- ------------Asignacion de Roles a Usuarios------------

GRANT Rol1,Rol2,Rol3,Rol4 TO lumartin 1 ;
GRANT Rol1,Rol2 TO lumartin 2 ;
GRANT Rol3,Rol4 TO lumartin 3 ;

-- ------------Asignacion de Privilegios a Usuarios--------

GRANT Pri1,Pri5 TO lumartin 2 ;
GRANT Pri1,Pri2,Pri3,Pri4,Pri5 TO lumartin 1 ;
GRANT Pri2,Pri4 TO lumartin 3 ;


-- ------------Consulta: nombreConsulta1----------------------
-- Descripcion:descripcionConsulta1

SELECT CARN.idCarnet,DEPA.gerente,EMPL.nombre
FROM DEPARTAMENTO DEPA,EMPLEADO EMPL,CARNET CARN
WHERE condicion1 AND condicion2 AND condicion3 AND condicion4


-- ------------Consulta: nombreConsulta2----------------------
```

```
-- Descripcion:descripcionConsulta2

SELECT DEPA.idDepartamento,CARN.fechaVencimiento,EMPL.apellido
FROM DEPARTAMENTO DEPA,EMPLEADO EMPL,CARNET CARN
WHERE condicion1 AND condicion2 AND condicion3 AND condicion4


-- ------------Consulta: nombreConsulta3----------------------
-- Descripcion:descripcionConsulta3

SELECT EMPL.CEDULA
FROM EMPLEADO EMPL
WHERE condicion1 AND condicion2 AND condicion3 AND condicion4
```

APÉNDICE I. HERRAMIENTAS DE PROGRAMACIÓN.

I.1. Java.

Java se creó como parte de un proyecto de investigación de Sun Microsystems® para el desarrollo de software avanzado para una amplia variedad de dispositivos de red y sistemas embebidos. La meta era diseñar una plataforma operativa sencilla, fiable, portable, distribuida y de tiempo real. Cuando se inició el proyecto, C++ era el lenguaje del momento. Pero a lo largo del tiempo, las dificultades encontradas con C++ crecieron hasta el punto en que se pensó que los problemas podrían resolverse mejor creando una plataforma de lenguaje completamente nueva. Se extrajeron decisiones de diseño y arquitectura de una amplia variedad de lenguajes como Eiffel, SmallTalk, Objetive C y Cedar/Mesa. El resultado es un lenguaje que se ha mostrado ideal para desarrollar aplicaciones de usuario final seguras, distribuidas y basadas en red en un amplio rango de entornos desde los dispositivos de red embebidos hasta los sistemas de sobremesa e Internet.(Murias,2005)

I.1.1. Objetivos de Java.

Los principales objetivos para los que el lenguaje java fue creado se pueden puntualizar en:

- **Sencillo, orientado a objetos y familiar:** Sencillo, para que no requiera grandes esfuerzos de entrenamiento para los desarrolladores. Orientado a objetos, porque la tecnología de objetos se considera madura y es el enfoque más adecuado para las necesidades de los sistemas distribuidos y/o cliente/servidor. Familiar, porque aunque se rechazó C++, se mantuvo Java lo más parecido posible a C++, eliminando sus complejidades innecesarias, para facilitar la migración al nuevo lenguaje. (Murias, 2005).
- **Robusto y seguro:** Robusto, simplificando la gestión de memoria y eliminando las complejidades de la gestión explicita de punteros y aritmética de punteros del C. Seguro para que pueda operar en un entorno de red.
- **Independiente de la arquitectura y portable:** Java está diseñado para soportar aplicaciones que serán instaladas en un entorno de red heterogéneo, con hardware y sistemas operativos diversos. Para hacer esto posible el interprete de Java genera "bytecodes", un formato de código independiente de la plataforma diseñado para transportar código eficientemente a través de múltiples plataformas de hardware y software. Es además portable en el sentido de que es rigurosamente el mismo lenguaje en todas las plataformas. El "bytecode" es traducido a código máquina y ejecutado por la Maquina Virtual de Java (JVM), que es la implementación Java para cada plataforma hardware-software concreta. .(Murias,2005)

- **Alto rendimiento:** A pesar de ser interpretado, Java tiene en cuenta el rendimiento, y particularmente en las últimas versiones dispone de diversas herramientas para su optimización. Cuando se necesitan capacidades de proceso intensivas, pueden usarse llamadas a código nativo.
- **Interpretado, multi-hilo y dinámico:** El intérprete Java puede ejecutar "bytecodes" en cualquier máquina que disponga de una Máquina Virtual Java (JVM). Además Java incorpora capacidades avanzadas de ejecución multi-hilo (ejecución simultánea de más de un flujo de programa) y proporciona mecanismos de carga dinámica de clases en tiempo de ejecución.

I.1.2.Características generales de Java.

- Lenguaje de propósito general.
- Lenguaje Orientado a Objetos.
- Sintaxis inspirada en la de C/C++.
- **Lenguaje multiplataforma:** Los programas Java se ejecutan sin variación (sin recompilar) en cualquier plataforma soportada (Windows, UNIX, Mac).
- **Lenguaje interpretado:** El intérprete a código máquina (dependiente de la plataforma) se llama Java Virtual Machine (JVM). El compilador produce un código intermedio independiente del sistema denominado bytecode.
- **Lenguaje gratuito:** Creado por SUN Microsystems, que distribuye gratuitamente el producto base, denominado JDK (Java Development Toolkit) o actualmente J2SE (Java 2 Standard Edition).
- API (Application Program Interface) distribuida con el J2SE muy amplia.
- Código fuente de la API disponible.

I.1.3. JRE (Java Runtime Environment)

JRE es el entorno mínimo para ejecutar programas escritos en lenguaje Java. Incluye la Maquina Virtual de Java (JVM) y la API. Está incluida en el J2SE aunque puede descargarse e instalarse separadamente. En aquellos sistemas donde se vayan a ejecutar programas Java, pero no compilarlos, el JRE es suficiente.

El JRE incluye el Java Plug-in, que es el "añadido" que necesitan lo navegadores (Internet Explorer o Netscape Navigator) para poder ejecutar programas Java. Es decir que instalando el JRE se tiene soporte completo Java, tanto para aplicaciones convencionales (denominadas "standalone") como para Applets (programas Java que se ejecutan en una página Web, cuando esta es accedida desde un navegador).(Ver Figura 67).

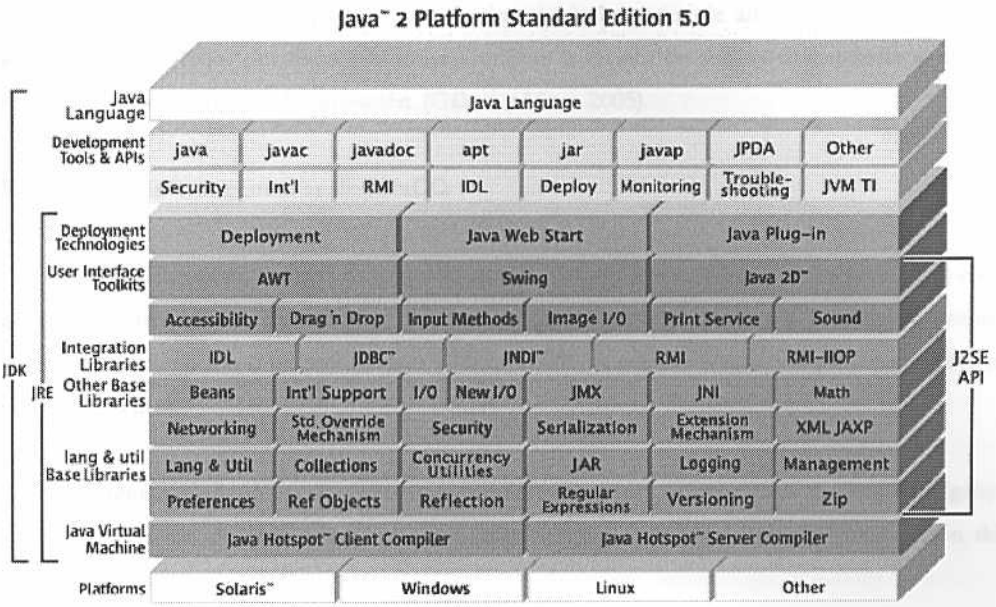


Figura 67.Esquema de la plataforma Java 2 Standard Edition 5.0.
Fuente: Java Documentation (http://java.sun.com/j2se/1.5.0/docs/index.html)

I.1.4.Recursos Necesarios para el desarrollo de aplicaciones Java.

El entorno mínimo necesario para escribir, compilar y ejecutar programas Java es el siguiente:

- J2SE (Java 2 Standard Edition) y la documentación (Se descargan por separado). Esto incluye el compilador Java, la maquina virtual de Java, el entorno de tiempo de ejecución (JRE) y varias herramientas de ayuda. La documentación contiene la referencia completa de la API.
- Un editor de textos.
- De forma opcional puede usarse un Entorno de Desarrollo Integrado para Java (IDE). Una herramienta de este tipo resulta aconsejable como ayuda para desarrollar aplicaciones o componentes. Sin embargo, en las primeras etapas del aprendizaje de Java no resulta necesario.

I.2. JavaCC.

JavaCC (Java Compiler Compiler - Metacompilador en Java) es el principal metacompilador en existente, tanto por sus posibilidades, como por su ámbito de difusión. Se trata de una herramienta basada en tecnología Java que facilita la construcción de analizadores léxicos y sintácticos mediante la utilizacion

de funciones recursivas, aunque permite una notación relajada muy parecida a la BNF (Backus-Naur Form que representa una notación que describe la sintaxis de lenguajes de alto nivel). De esta manera, los analizadores generados por dicha herramienta utilizan la técnica de análisis descendente al momento de obtener un árbol sintáctico de derivación. (Gálvez y Mora, 2005).

I.2.1.Características generales de JavaCC.

JavaCC integra en una misma herramienta al analizador lexicográfico y al sintáctico, y el código que genera es independiente de cualquier librería externa, lo que le confiere una interesante propiedad de independencia respecto al entorno(Gálvez y Mora, 2005).. A grandes rasgos, sus principales características son las siguientes:

- Genera analizadores descendentes, permitiendo el uso de gramáticas de propósito general y la utilización de atributos tanto sintetizados como heredados durante la construcción del árbol sintáctico.

- Las especificaciones léxicas y sintácticas se ubican en un solo archivo. De esta manera la gramática puede ser leída y mantenida más fácilmente. No obstante, cuando se introducen acciones semánticas, recomendamos el uso de ciertos comentarios para mejorar la legibilidad.

- Admite el uso de estados léxicos y la capacidad de agregar acciones léxicas incluyendo un bloque de código Java tras el identificador de un token.

- Incorpora distintos tipos de tokens: normales (TOKEN), especiales (SPECIAL_TOKEN), espaciadores (SKIP) y de continuación (MORE). Ello permite trabajar con especificaciones más claras, a la vez que permite una mejor gestión de los mensajes de error y advertencia por parte de la herramienta en tiempo de metacompilación.

- Los tokens especiales son ignorados por el analizador generado, pero están disponibles para poder ser procesados por el desarrollador.

- La especificación léxica puede definir tokens de manera tal que no se diferencien las mayúsculas de las minúsculas bien a nivel global, bien en un patrón concreto.

- Adopta una notación BNF propia mediante la utilización de símbolos propios de expresiones regulares, tales como (A)*, (A)+.

- Genera por defecto un analizador sintáctico LL(1). No obstante, puede haber porciones de la gramática que no sean LL(1), lo que es resuelto en JavaCC mediante la posibilidad de resolver las ambigüedades desplazar/desplazar localmente en el punto del conflicto. En otras palabras, permite que se transforme en LL(k) sólo en tales puntos, pero se conserva LL(1) en el resto de las reglas mejorando la eficiencia.

- De entre los generadores de analizadores sintácticos descendentes, JavaCC es uno de los que poseen mejor gestión de errores. Los analizadores generados por JavaCC son capaces de localizar exactamente la ubicación de los errores, proporcionando información diagnóstica completa para su corrección.

- Permite entradas codificadas en Unicode, de forma que las especificaciones léxicas también pueden incluir cualquier carácter Unicode. Esto facilita la descripción de los elementos del lenguaje, tales como los identificadores Java que permiten ciertos caracteres Unicode que no son ASCII (American Standard Code for Information Interchange o Código Estadounidense Estándar para el Intercambio de Información). (Gálvez y Mora, 2005).

- JavaCC ofrece muchas opciones diferentes para personalizar su comportamiento y el comportamiento de los analizadores generados.

- Incluye la herramienta JJTree, un preprocesador para el desarrollo de árboles con características muy potentes.

- Incluye la herramienta JJDoc que convierte los archivos de la gramática en archivos de documentación.

- Es altamente eficiente, lo que lo hace apto para entornos profesionales y lo ha convertido en uno de los metacompiladores más extendidos.

I.2.2. Características de programas escritos en JavaCC.

Los programas JavaCC se suelen almacenar en archivos con extensión .jj, como por ejemplo "Analizador.jj", y al realizar la compilación del mismo se producen los siguientes archivos de salida:

- **Analizador.java:** es el analizador sintáctico.
- **AnalizadorTokenManager.java:** es el analizador lexicográfico.

- **AnalizadorConstants.java:** interface que asocia un código a cada token. Asimismo, almacena en forma de String el nombre dado por el desarrollador a cada token, con el propósito de que el analizador sintáctico pueda emitir mensajes de errores claros y explicativos.

Además de los archivos nombrados anteriormente se generan las clases necesarias para el complemento del analizador los cuales son puntualizados a continuación:

- **ParseException.java:** clase que implementa los errores sintácticos. Cuando el analizador sintáctico encuentra uno de estos errores genera una excepción que puede ser capturada en cualquier nodo ancestro del árbol sintáctico que se estaba construyendo. (Gálvez y Mora, 2005).
- **SimpleCharStream.java:** clase que implementa los canales de entrada de los cuales puede leer el analizador léxico. Incorpora constructores para convertir los canales de tipo tradicional: InputStream y Reader.
- **Token.java:** clase que implementa el objeto a través del cual se comunican el analizador léxico y el sintáctico.
- **TokenMgrError.java:** clase que implementa los errores lexicográficos. Este tipo de errores no se puede capturar de forma fácil, por lo que el desarrollador debe recoger todos los lexemas posibles y generar sus propios errores ante los incorrectos (Ver figura 68).

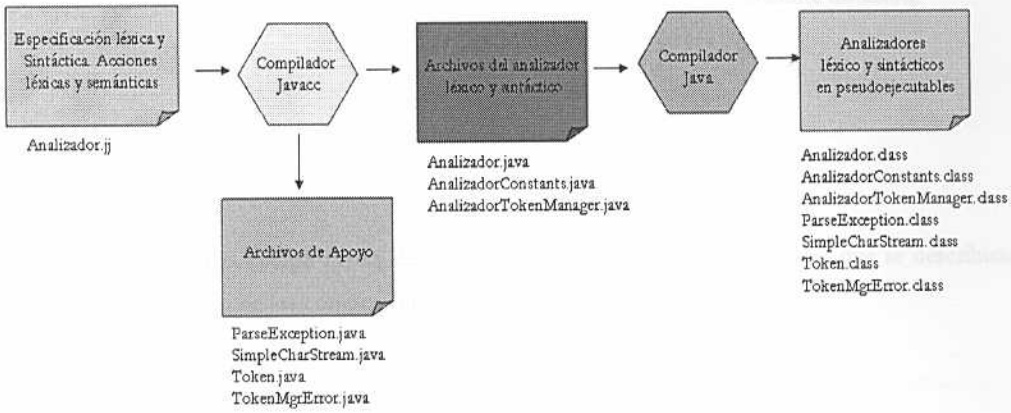


Figura 68.Proceso de Metacompilacion de JavaCC.
Fuente: Modificado por los autores del trabajo de (Gálvez y Mora, 2005).

APÉNDICE J. ESTÁNDARES DE CODIFICACIÓN UTILIZADOS PARA LA IMPLEMENTACIÓN DE LA HERRAMIENTA.

Para la implementación de la herramienta desarrollada, se utilizaron diversos estándares de codificación, de manera de dar mayor mantenibilidad al código fuente.

El lenguaje Java usa los siguientes tipos de sufijos:

Tipo de Archivo	Extensión
Código Fuente Java	.java
Bytecode Java	.class

Tabla 18. Sufijos del lenguaje Java.
Fuente: Elaborado por los autores.

Para la codificación de cada uno de los componentes del código fuente (variables, estructuras, métodos) se utilizaron para nombrarlos alias nemotécnicos con el fin al momento de realizar el mantenimiento de la aplicación, estuviesen escritas de la forma mas natural posible. Para los componentes que requerían ser llamados con nombres de más de una palabra, se utilizo la siguiente notación:

nombre**V**ariable.
nombre**M**etodo.
nombre**E**structura.

Para estandarizar el código fuente de la aplicación se utilizaron las notaciones que se describirán a continuación en cada uno de los componentes.

Clases:

Los nombres utilizados para las clases de la aplicación, fueron escritos con la letra inicial en mayúscula y el resto de la palabra en letra minúscula para diferenciarlas de los otros componentes del código fuente (variables métodos).

Ejemplo:

```
public class Entidad{Bloques de código}
public class Atributo{Bloques de código}
```

Variables:

Los nombres de variables están relacionados con el valor al que representan.

Ejemplo:

String nombreModelo; (Atributo que almacena el nombre de un modelo especifico).

int idModelo; (Atributo que almacena el identificador de referencia de un modelo).

Métodos:

Los nombres de métodos están relacionados a la operación que realiza.

Ejemplo:

public boolean getModelById(); (Método que recupera un Modelo dado su identificador de referencia).

Documentación.

Para realizar la documentación del código fuente de la aplicación, se utilizaron 2 tipos de comentarios permitidos en Java.

Para la documentación interna del código se utilizo el carácter "//", que permite congelar la línea sobre la cual se aplique, con el fin de escribir comentarios acerca de la lógica utilizada para la manipulación de los objetos de esa clase.

Ejemplo:

//Comentario de línea simple

Para la documentación de bloques de código, se utilizaron los caracteres de comentarios múltiple línea "/**/".

Ejemplo:

/*Comentario de Múltiples líneas en la que se pueden incluir
Bloques completos de código*/

A fin de utilizar el comando Javadoc de Java (Comando para generación de API en formato HTML), las convenciones de documentación utilizadas fueron las siguientes:

- Sobre la línea que contiene la declaración de la clase a implementar se coloco la siguiente estructura:

```
/**
* Comentario acerca de la clase que se desea documentar.
*@author Maria A. Pinto y Luis S. Martinez.
*@versión 1.0
*/
public class Ejemplo{Bloques  de código}
```

- Para los atributos de la clase.

```
public class Ejemplo{

 /**
   * Comentario acerca del atributo.
   */
  public String atributo1;

..............................}
```

- Para el constructor de la clase.

```
public class Ejemplo{
..........................
  /**
   * Comentario acerca de la descripción del constructor.
   *@param param1 Descripción del parámetro 1.
   *@param param2 Descripción del parámetro 2.
   **/
       public Ejemplo(String param1, int param2){}
..............................}
```

- Métodos de la clase.

```
public class Ejemplo{
..............................
 /**
  * Comentario acerca del método de la clase.
  *@param param1 Comentario acerca del parámetro que recibe el método.
  *@return ret1 Comentario acerca del valor que retorna el método.
  **/

 public void nombreMetodo(String param1){
 return ret1;
 }

 ...............................}
```

Luego de realizar toda la documentación como se presento en los puntos anteriores, se ejecuto en la comando Javadoc para cada una de las clases, mediante el cual se genero el API de la documentación de la herramienta desarrollada.

APÉNDICE K. GRAMÁTICA PARA LOS PROCESOS DE INGENIERÍA REVERSA.

Para la construcción del analizador sintáctico y léxico requerido para realizar los procesos de ingeniería reversa dentro de la herramienta, fue necesaria la definición de una gramática, mediante la cual serian reconocidas las sentencias leídas del archivo fuente escrito en lenguaje de definición de datos SQL.

Entre los elementos necesarios para dicho analizador, se encuentra las constantes y palabras reservadas del lenguaje mediante las cuales se realizaría el reconocimiento de los comandos obtenidos mediante el archivo SQL. (Ver Tabla 19).

Constante	Carácter Utilizado
<parA>	"("
<parC>	")"
< pYc>	";"
<coma>	","
<SentenciaCreateTable>	"CREATE TABLE"
<SentenciaConstraint>	"CONSTRAINT"
<Primary>	"PRIMARY KEY"
<Foreign>	"FOREIGN KEY"
<Check>	"CHECK"
<SentenciaAlterTable>	"ALTER TABLE"
<SentenciaDropTable>	"DROP TABLE"
<SentenciaCreateIndex>	"CREATE INDEX"
<In>	"IN"
<Add>	"ADD"
<On>	"ON"
<CascadeConstraints>	"CASCADE CONSTRAINTS"
<References>	"REFERENCES"
< String>	(<Caracter>)+
<Carácter>	{a....z | (A....Z)}
<Digito>	{0...9}
<Numero>	(<Digito>)+
<NotNull>	"NOT NULL"

Tabla 19. Constantes y palabra reservadas de la gramatica.
Fuente: Elaborado por los autores.

K.1.Estructura de la gramática.

Basándose en las constantes definidas anteriormente, se construyó una gramática que establece las reglas para el proceso de ingeniería reversa de la herramienta. Se utilizaron expresiones regulares que siguen las reglas sintácticas del lenguaje SQL.

La gramática construida, posee una notación mediante la cual se establece cuales son sus elementos no terminales, así como también los caracteres de repetición de cada uno de ellos, indicándole así al traductor cual será la sintaxis correcta que deberá reconocer mediante el proceso de análisis tanto sintáctico como semántico de cada oración de entrada. (Ver Tabla 20).

Elemento	Notación
Elemento No Terminal	**noTerminal()**
Elemento Terminal	**<Terminal>**
Elemento de Repetición	*** cero o mas veces**
	+ una o mas veces
	? cero o una vez

Tabla 20. Notación de la gramática.
Fuente: Elaborado por los autores.

A continuación se presenta la lista de producciones que complementan la gramática construida para el analizador sintáctico y léxico del lenguaje SQL.

Axioma ( ) → createTable () | alterTable () | createIndex () | dropTable ()

createTable ( ) → <SentenciaCreateTable>< String><parA> (Atributo( ))+ (constraint( ))? <parC>< pYc>

Atributo ( ) → < String>< String> (<parA> <Numero> <parC>)? (<NotNull>)? (<coma>)?

constraint ( ) → <SentenciaConstraint>< String> (pk () | fk () | ck ())

pk ( ) → <Primary><parA>< String><parC>

fk ( ) → <Foreign><parA>< String><parC><References>< String><parA>< String><parC>

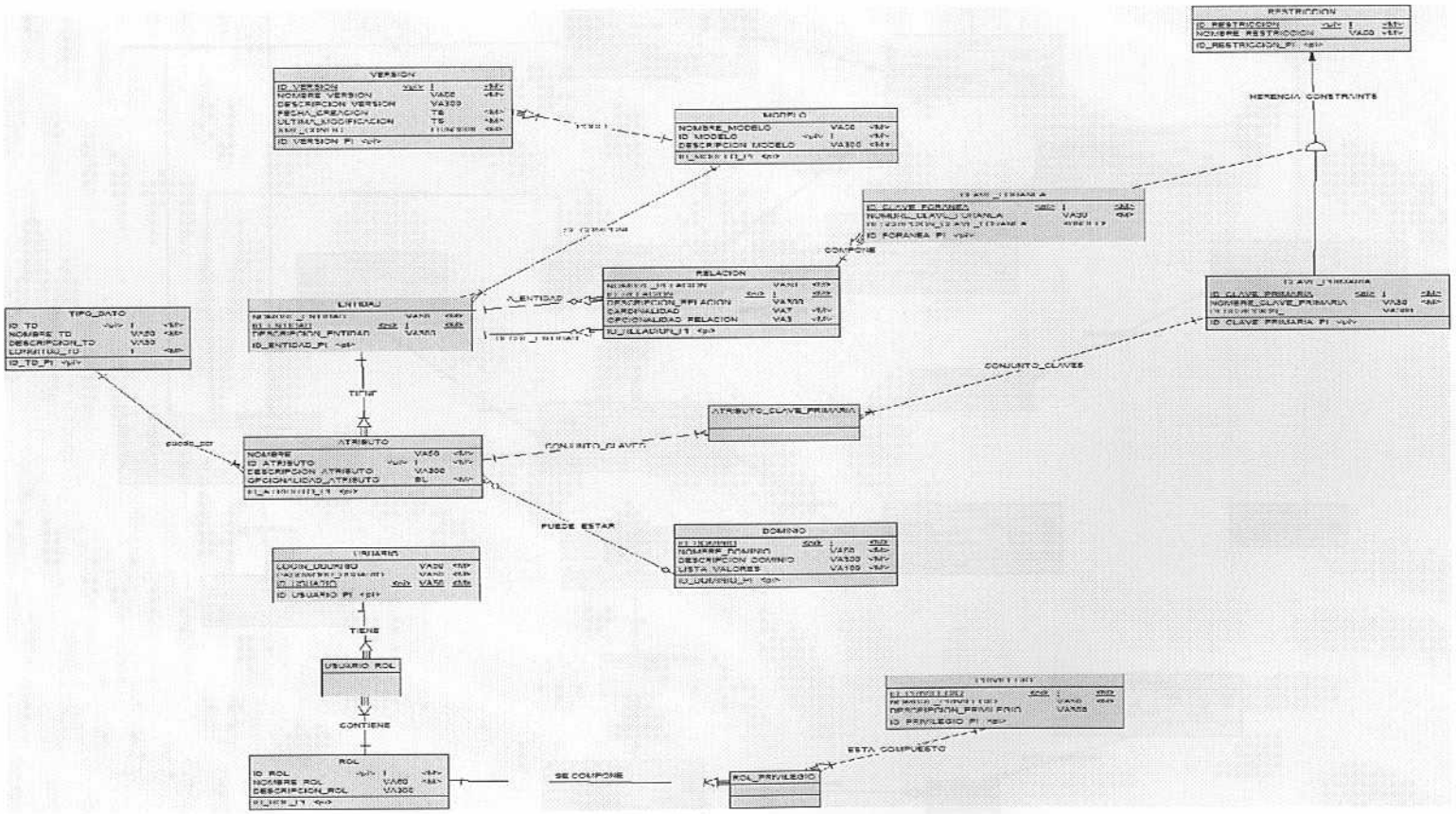
ck ( ) → <Check><parA>< String><In><parA>< String><parC><parC>

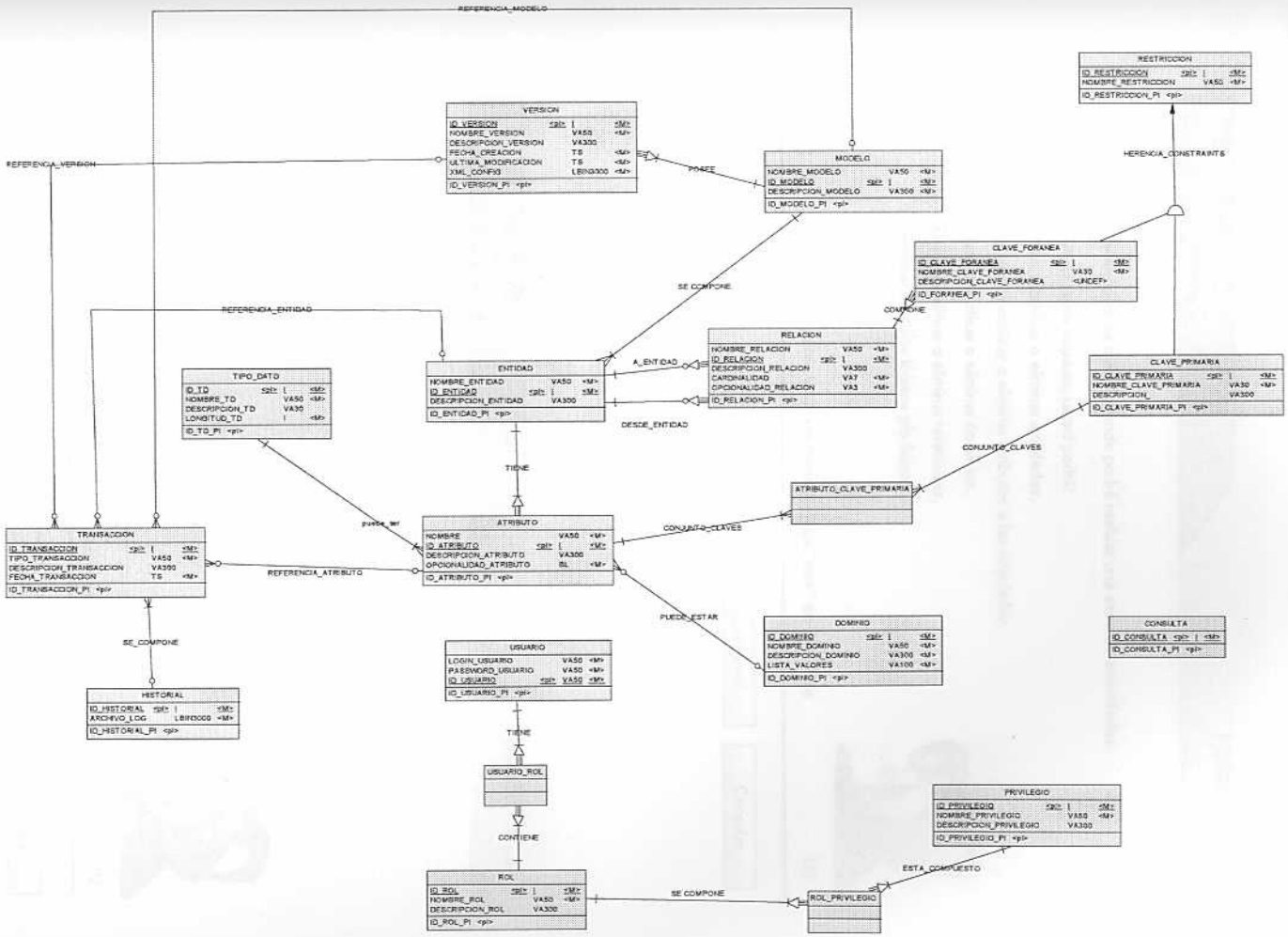
alterTable ( ) → <SentenciaAlterTable>< String><Add><parA> (constraint ( ))* <parC>< pYc>

createIndex ( ) →<SentenciaCreateIndex>< String><On>< String><parA><parC>< pYc>

dropTable ( ) → <SentenciaDropTable>< String><CascadeConstraints>< pYc>

APENDICE L.VERSIONES DE LOS MODELOS




REFERENCIA_MODELO
RESTRICCION
ID_RESTRICCION <pk> I <M>
NOMBRE_RESTRICCION VA50 <M>
ID_RESTRICCION_PI <pi>
VERSION
ID_VERSION <pk> I <M>
NOMBRE_VERSION VA50 <M>
DESCRIPCION_VERSION VA300
FECHA_CREACION TS <M>
ULTIMA_MODIFICACION TS <M>
XML_CONFIG LBINO300 <M>
ID_VERSION_PI <pi>
REFERENCIA_VERSION
POSEE
MODELO
NOMBRE_MODELO VA50 <M>
ID_MODELO <pk> I <M>
DESCRIPCION_MODELO VA300 <M>
ID_MODELO_PI <pi>
HERENCIA_CONSTRAINTS
CLAVE_FORANEA
ID_CLAVE_FORANEA <pk> I <M>
NOMBRE_CLAVE_FORANEA VA30 <M>
DESCRIPCION_CLAVE_FORANEA <UNDEF>
ID_FORANEA_PI <pi>
SE COMPONE
COMPONE
REFERENCIA_ENTIDAD
RELACION
NOMBRE_RELACION VA50 <M>
ID_RELACION <pk> I <M>
DESCRIPCION_RELACION VA300
CARDINALIDAD VA7 <M>
OPCIONALIDAD_RELACION VA3 <M>
ID_RELACION_PI <pi>
A_ENTIDAD
TIPO_DATO
ID_TD <pk> I <M>
NOMBRE_TD VA50 <M>
DESCRIPCION_TD VA30
LONGITUD_TD I <M>
ID_TD_PI <pi>
ENTIDAD
NOMBRE_ENTIDAD VA50 <M>
ID_ENTIDAD <pk> I <M>
DESCRIPCION_ENTIDAD VA300
ID_ENTIDAD_PI <pi>
DESDE_ENTIDAD
CLAVE_PRIMARIA
ID_CLAVE_PRIMARIA <pk> I <M>
NOMBRE_CLAVE_PRIMARIA VA30 <M>
DESCRIPCION_ VA300
ID_CLAVE_PRIMARIA_PI <pi>
TIENE
CONJUNTO_CLAVES
ATRIBUTO_CLAVE_PRIMARIA
puede ser
ATRIBUTO
NOMBRE VA50 <M>
ID_ATRIBUTO <pk> I <M>
DESCRIPCION_ATRIBUTO VA300
OPCIONALIDAD_ATRIBUTO BL <M>
ID_ATRIBUTO_PI <pi>
CONJUNTO_CLAVES
TRANSACCION
ID_TRANSACCION <pk> I <M>
TIPO_TRANSACCION VA50 <M>
DESCRIPCION_TRANSACCION VA300
FECHA_TRANSACCION TS <M>
ID_TRANSACCION_PI <pi>
REFERENCIA_ATRIBUTO
PUEDE_ESTAR
DOMINIO
ID_DOMINIO <pk> I <M>
NOMBRE_DOMINO VA50 <M>
DESCRIPCION_DOMINO VA300 <M>
LISTA_VALORES VA100 <M>
ID_DOMINO_PI <pi>
CONSULTA
ID_CONSULTA <pk> I <M>
ID_CONSULTA_PI <pi>
SE_COMPONE
HISTORIAL
ID_HISTORIAL <pk> I <M>
ARCHIVO_LOG LBINO300 <M>
ID_HISTORIAL_PI <pi>
USUARIO
LOGIN_USUARIO VA50 <M>
PASSWORD_USUARIO VA50 <M>
ID_USUARIO <pk> VA50 <M>
ID_USUARIO_PI <pi>
TIENE
USUARIO_ROL
CONTIENE
PRIVILEGIO
ID_PRIVILEGIO <pk> I <M>
NOMBRE_PRIVILEGIO VA50 <M>
DESCRIPCION_PRIVILEGIO VA300
ID_PRIVILEGIO_PI <pi>
ESTA_COMPUESTO
ROL
ID_ROL <pk> I <M>
NOMBRE_ROL VA50 <M>
DESCRIPCION_ROL VA300
ID_ROL_PI <pi>
SE COMPONE
ROL_PRIVILEGIO

APÉNDICE M.PANTALLAS

M.1. Crear Modelo.

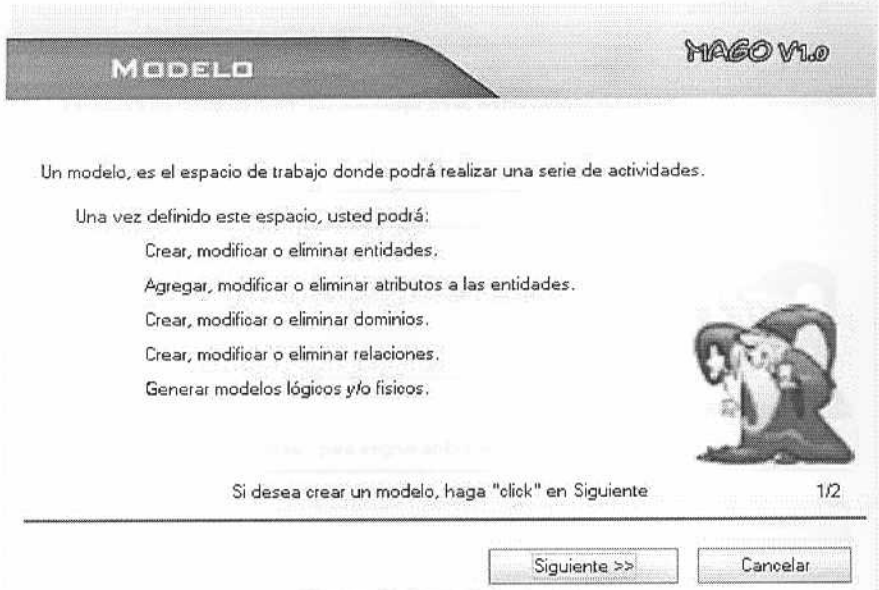


Figura 69.Pantalla modelo1.
Fuente: Elaborado por los autores.

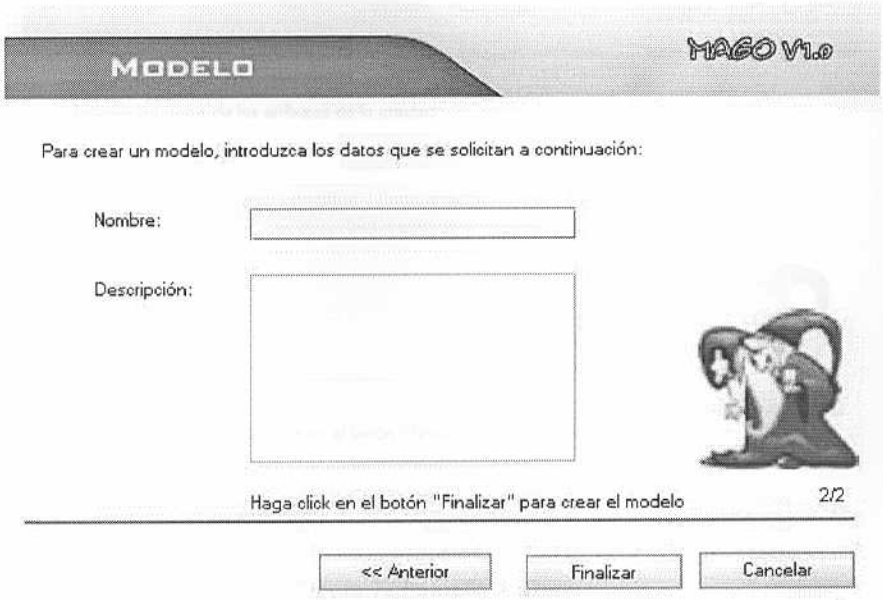


Figura 70.Pantalla modelo 2.
Fuente: Elaborado por los autores.

M.2. Crear Entidad.

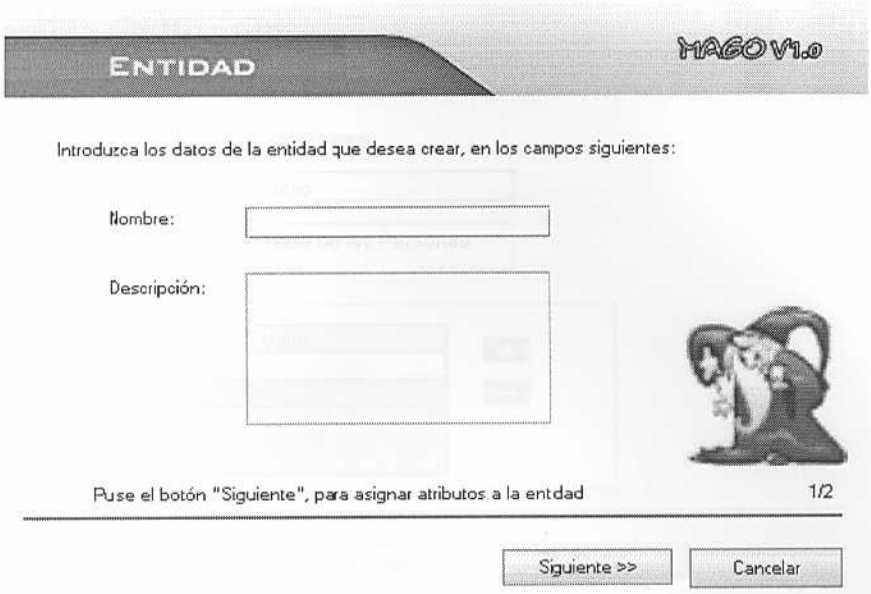


Figura 71.Pantalla entidad.
Fuente: Elaborado por los autores.

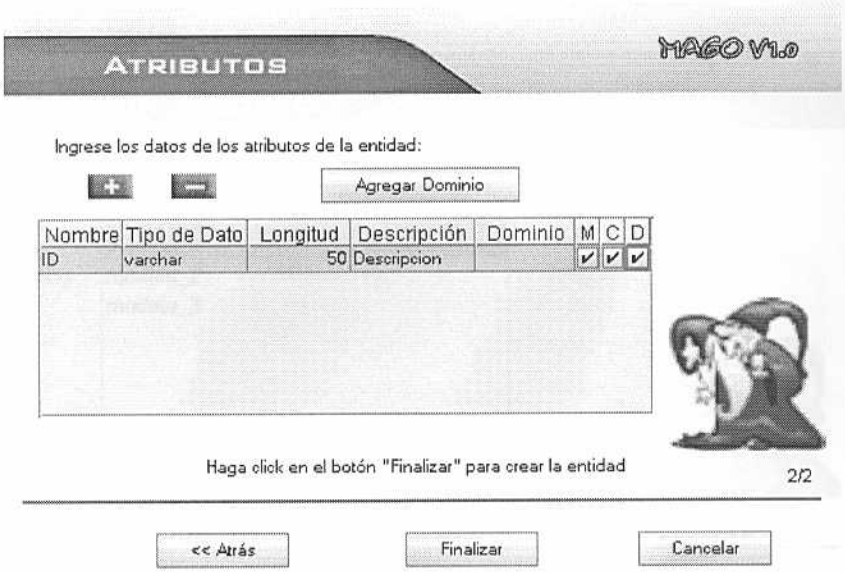


Figura 72.Pantalla atributos.
Fuente: Elaborado por los autores.

M.3. Crear Dominio.

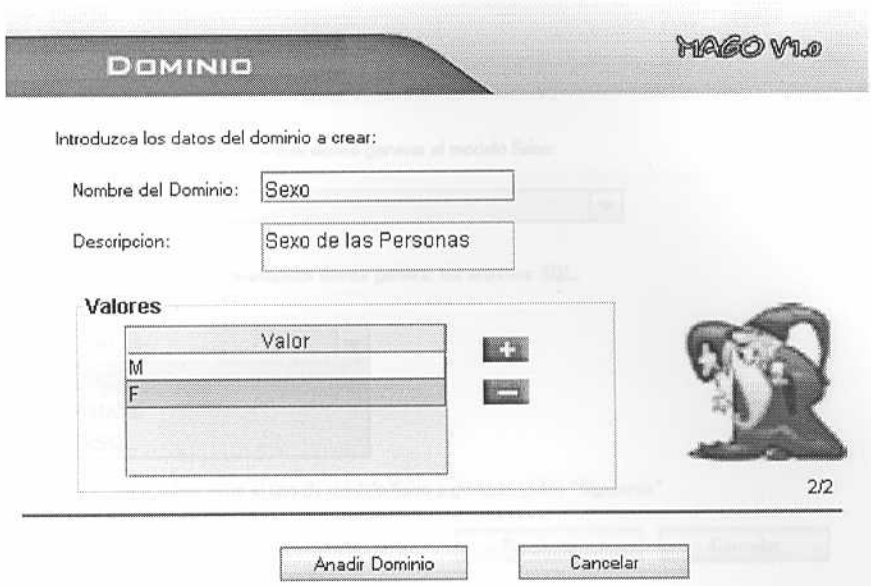


Figura 73.Pantalla dominio.
Fuente: Elaborado por los autores.

M.4. Abrir Modelo.

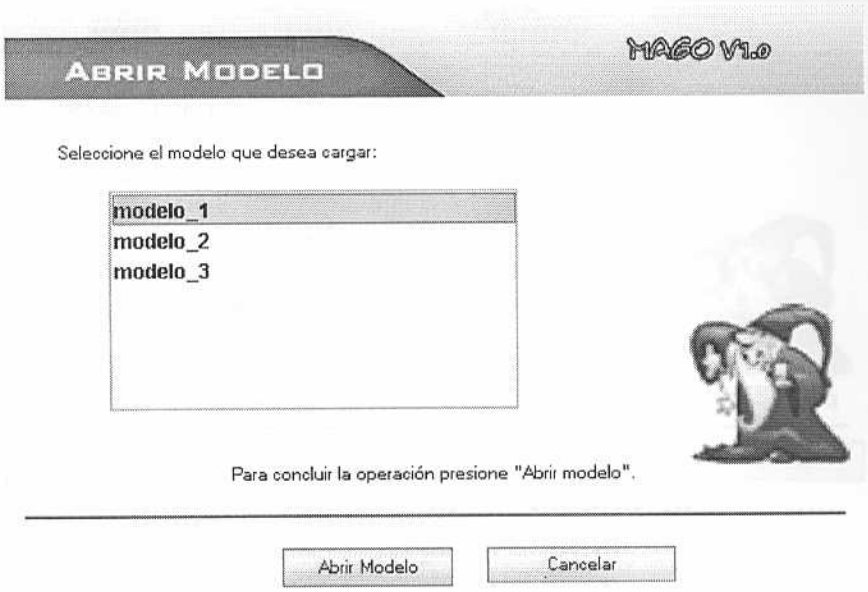


Figura 74.Pantalla abrir modelo.
Fuente: Elaborado por los autores.

M.5. Generar Modelo Físico.

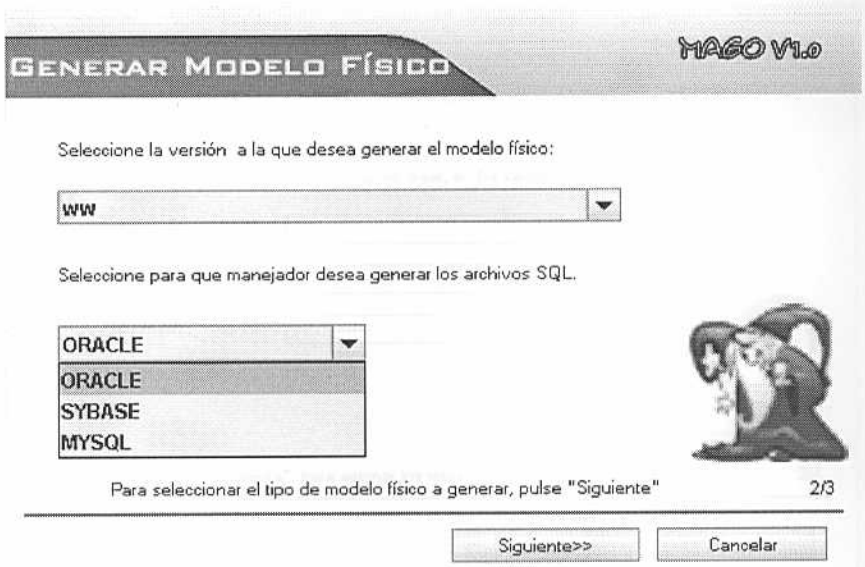


Figura 75.Pantalla generar modelo físico.
Fuente: Elaborado por los autores.

Figura 76.Pantalla generar modelo físico.
Fuente: Elaborado por los autores.

M.6. Crear Usuario.

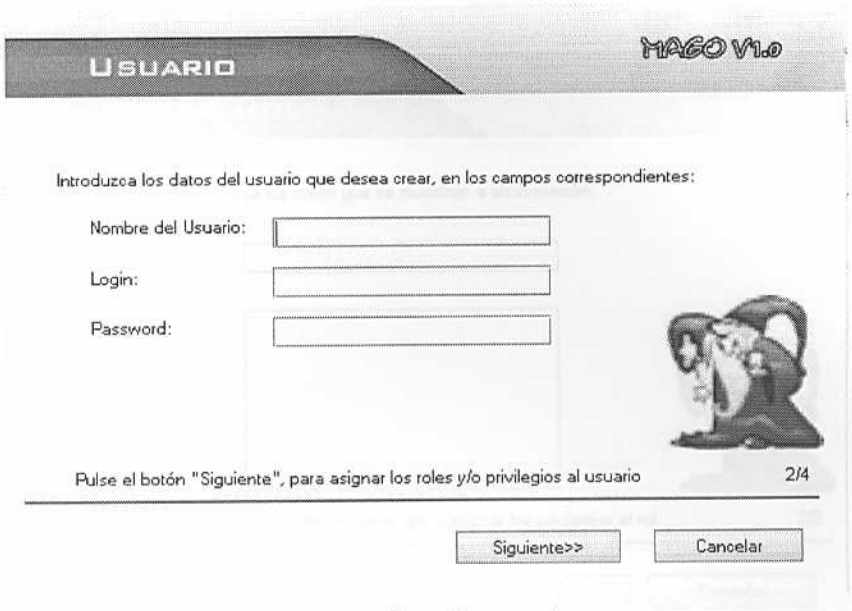


Figura 77.Pantalla usuario.
Fuente: Elaborado por los autores.

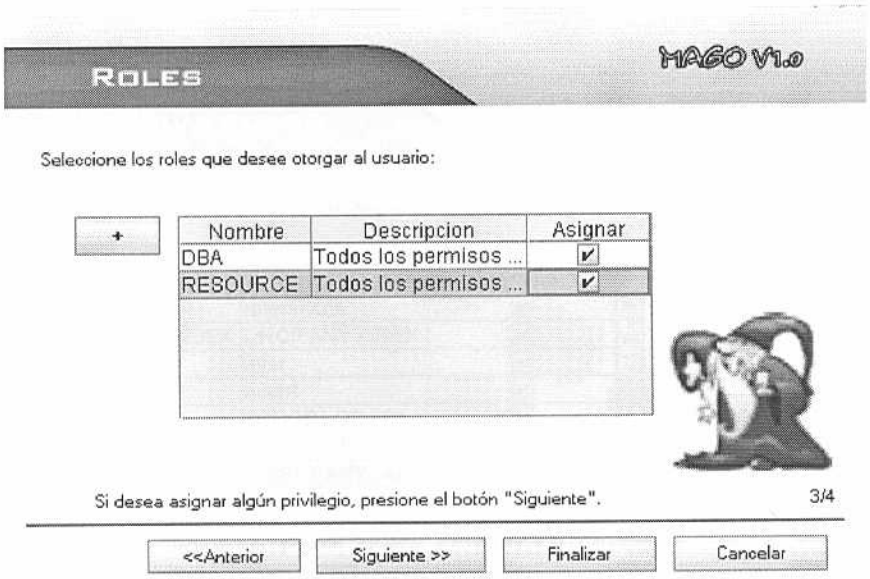


Figura 78.Pantalla roles.
Fuente: Elaborado por los autores.

M.7. Crear Rol.

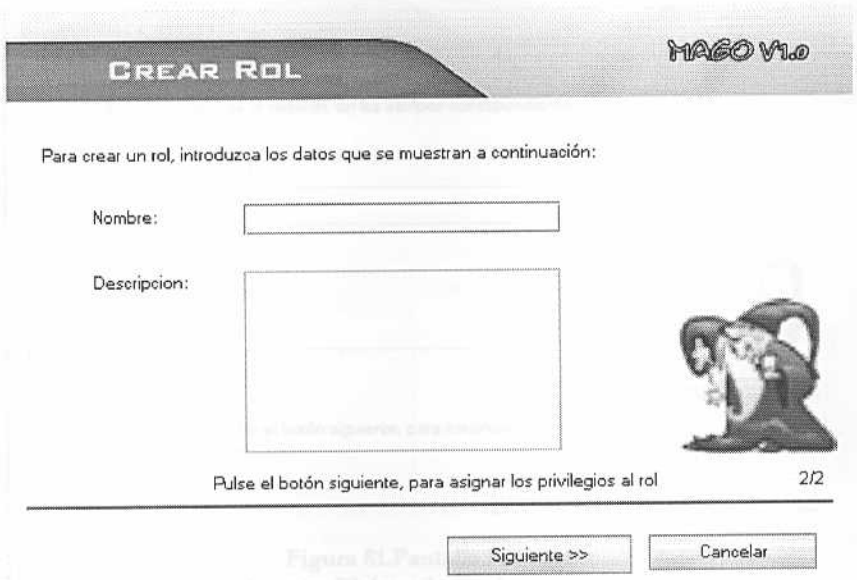


Figura 79.Pantalla crear rol.
Fuente: Elaborado por los autores.

Figura 80.Pantalla privilegios.
Fuente: Elaborado por los autores.

M.8. Crear Relación.

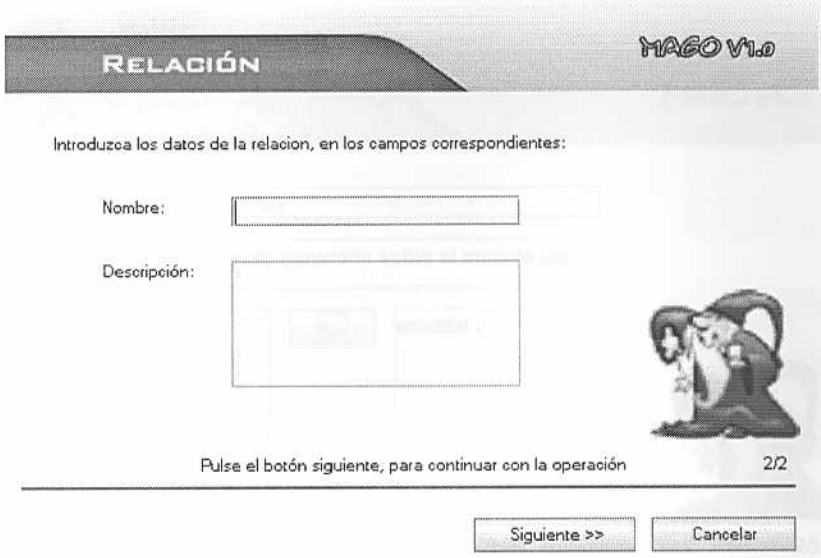


Figura 81.Pantalla relación.
Fuente: Elaborado por los autores.

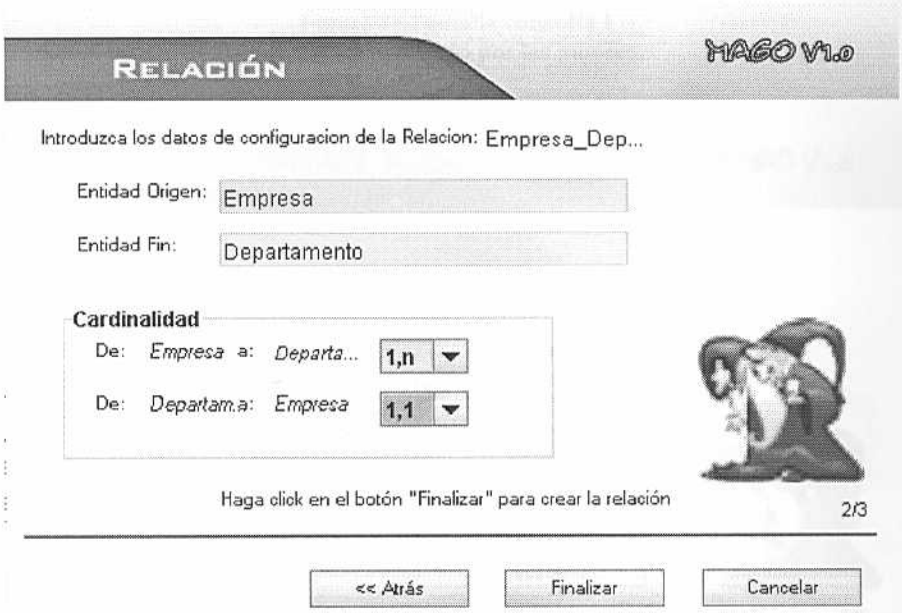


Figura 82.Pantalla relación 2.
Fuente: Elaborado por los autores.

M.9. Generar Consulta.

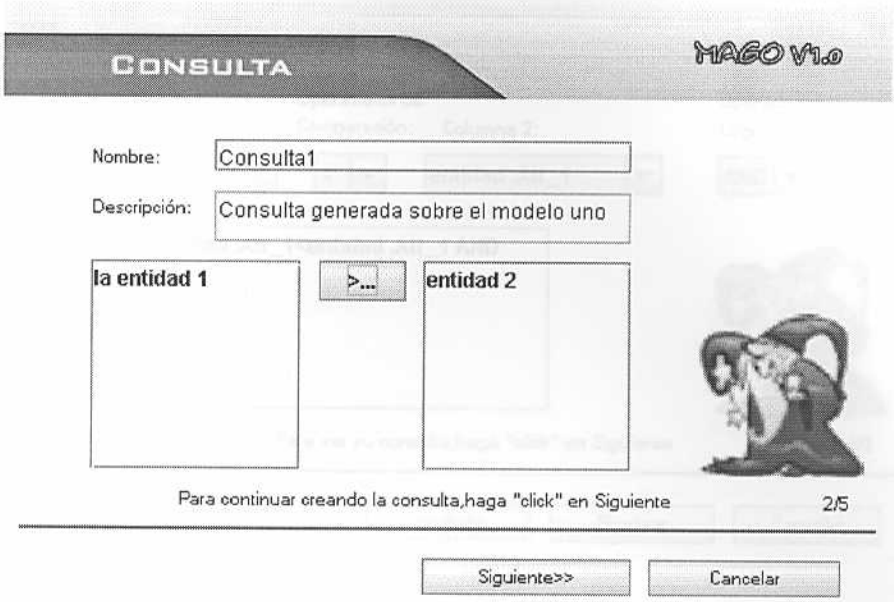


Figura 83.Pantalla consulta 1
Fuente: Elaborado por los autores.

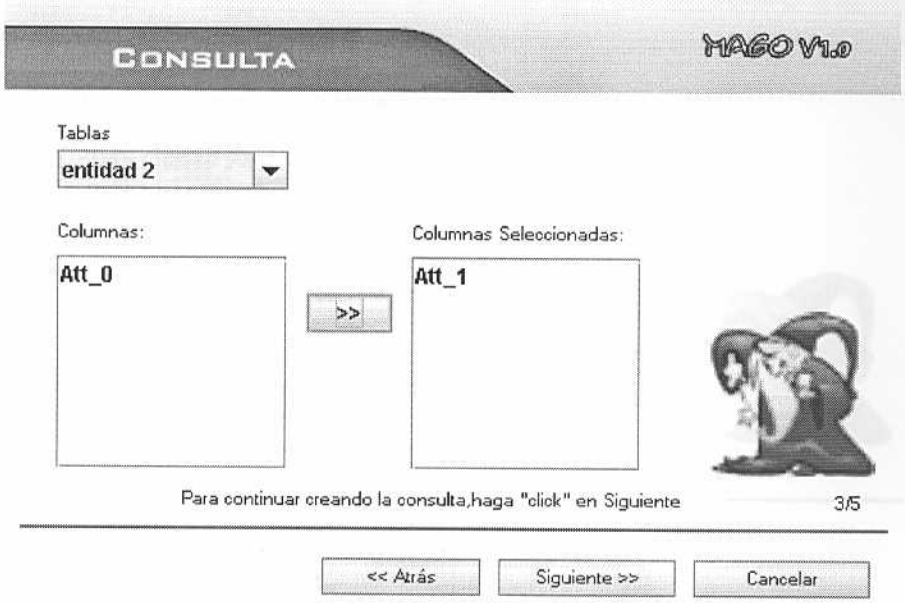


Figura 84.Pantalla consulta 2.
Fuente: Elaborado por los autores.

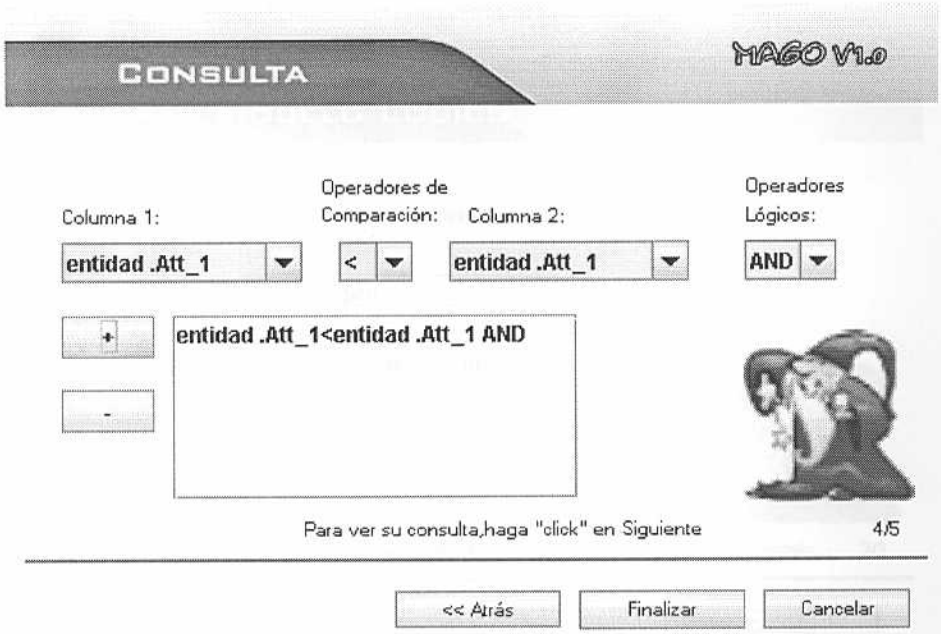


Figura 85.Pantalla consulta 3.
Fuente: Elaborado por los autores.

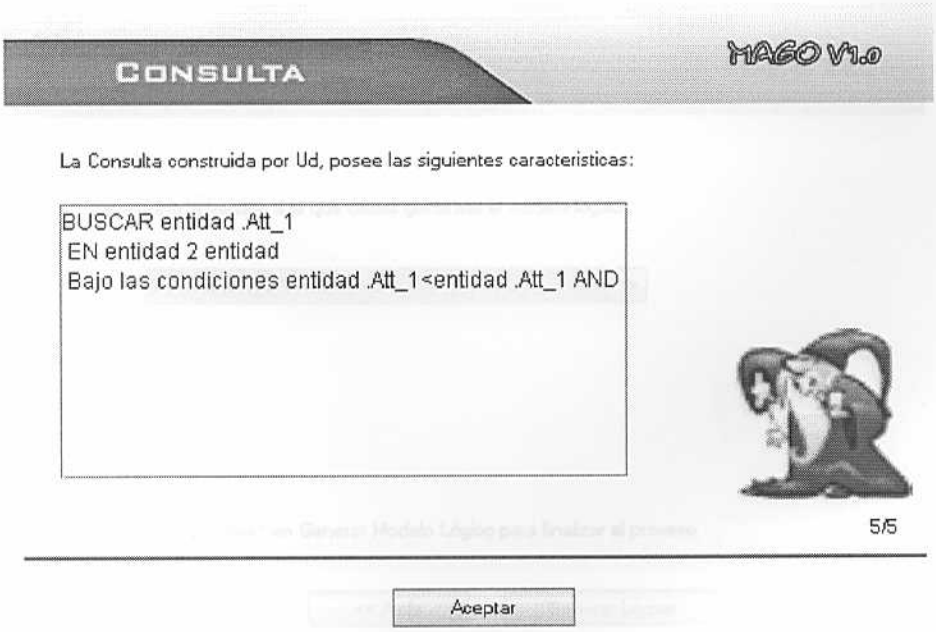


Figura 86.Pantalla consulta 4.
Fuente: Elaborado por los autores.

M.10. Generar Modelo Lógico.

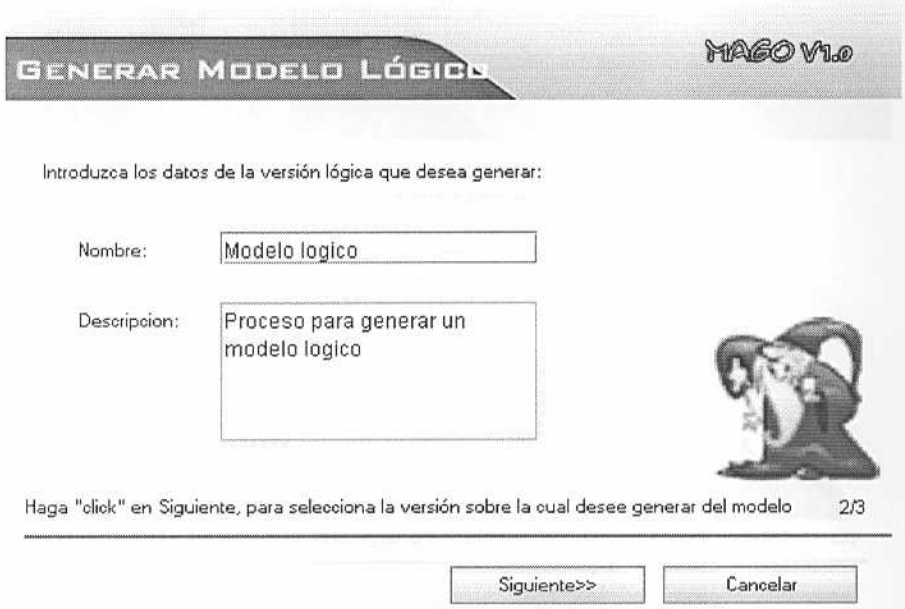


Figura 87.Pantalla generar modelo lógico 1.
Fuente: Elaborado por los autores.

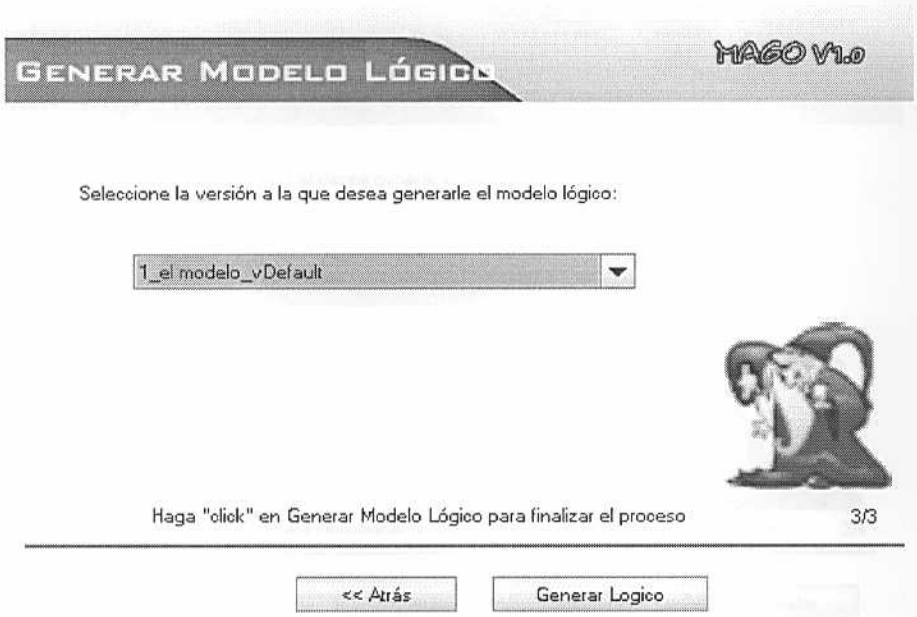


Figura 88.Pantalla generar modelo lógico 2.
Fuente: Elaborado por los autores.

M.11. Generar Reporte.

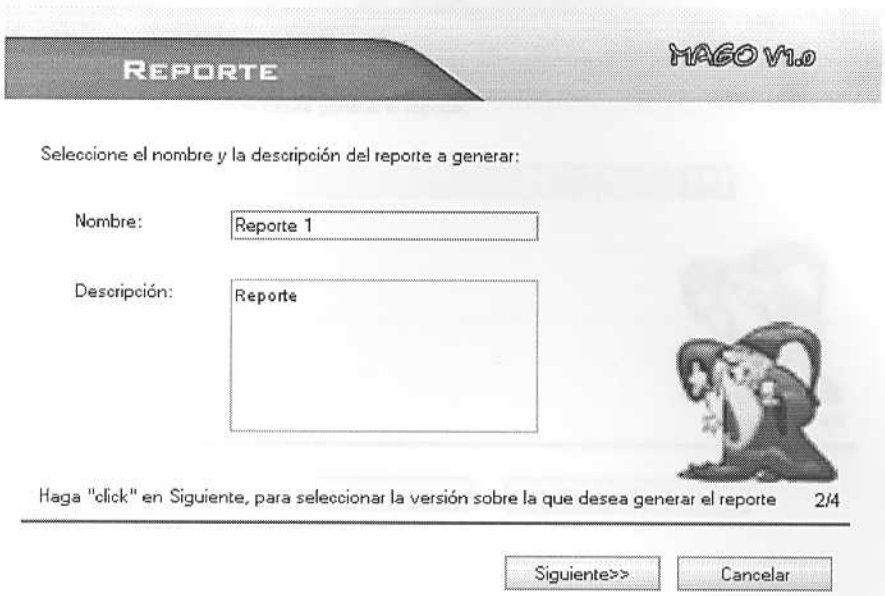


Figura 89.Pantalla reporte 1.
Fuente: Elaborado por los autores.

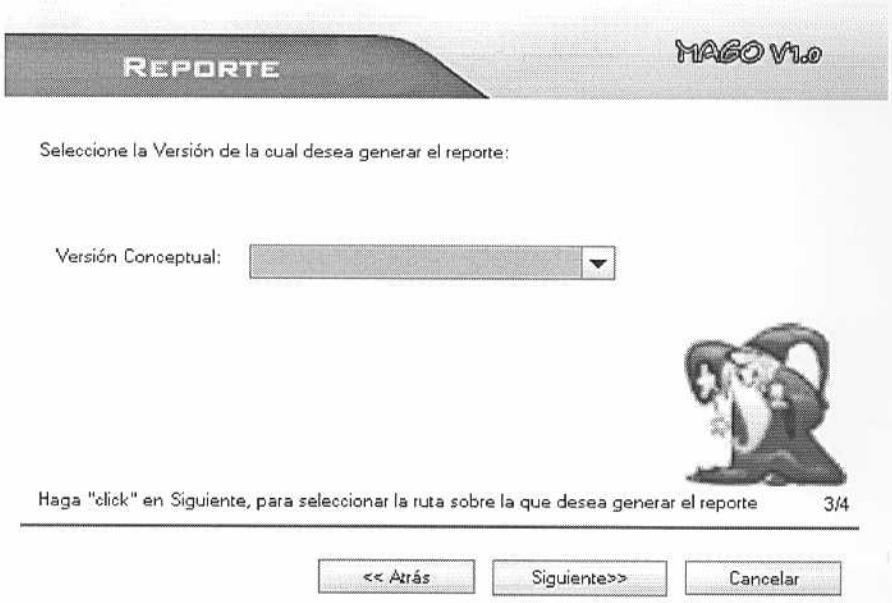


Figura 90.Pantalla reporte 2.
Fuente: Elaborado por los autores.

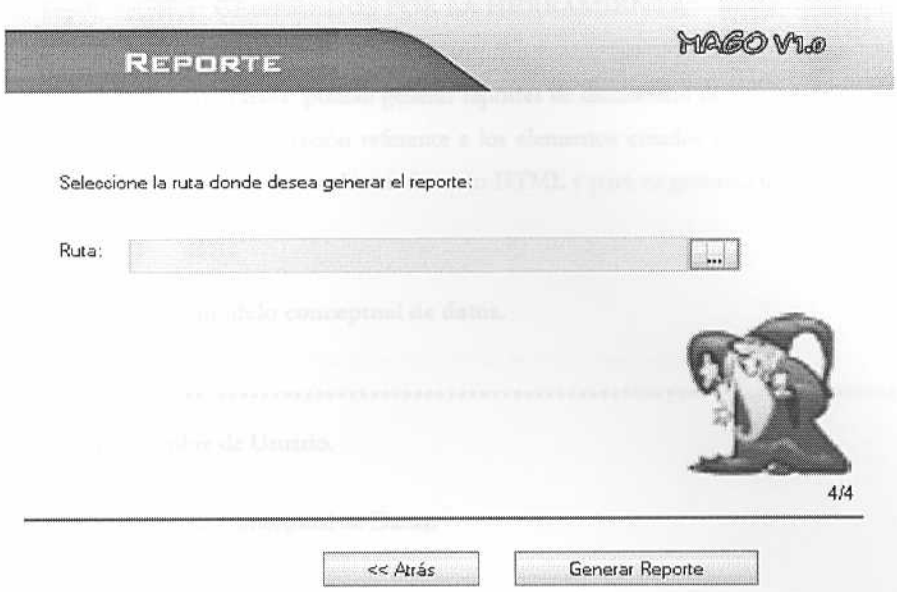


Figura 91.Pantalla reporte 3.
Fuente: Elaborado por los autores.

APÉNDICE N. ESTRUCTURA DE REPORTES DE DICCIONARIO DE DATOS GENERADOS POR LA HERRAMIENTA

Los usuarios de la herramienta podrán generar reportes de diccionario de datos mediante los cuales podrán recopilar toda aquella información referente a los elementos creados dentro de sus modelos de datos. Dichos reportes pueden ser generados en formato HTML y para su generación se utilizo la plantilla presentada a continuación.

Plantilla de Reporte para modelo conceptual de datos.

****

Usuario Propietario: Nombre de Usuario.

Nombre del Modelo: Modelo Conceptual de Datos.

Tipo de Modelo: Modelo Conceptual.

Fecha de Creación del Modelo: DD/MM/AAAA.

Nombre de la Versión: Versión 1.0 del Modelo Conceptual de Datos.

Descripción de la Versión: Descripción de prueba para la versión 1.0.

****

LISTA DE ENTIDADES.

Entidad1

Nombre de la Entidad: Entidad 1.

Descripción de la entidad: Descripción de Entidad 1.

Lista de Atributos de la Entidad 1.

Nombre	id	Descripción	Clave	Único	Mandatorio	Tipo Dato	Dominio
Atributo	idAtributo	Descripción del Atributo	X	X	X	Tipo de Dato	Nombre del Dominio

****

LISTA DE RELACIONES.

Nombre	id	Descripción	Entidad Origen	Entidad Fin	Cardinalidad
Relación	idRelación	Descripción de la relación	Entidad Origen	Entidad Fin	Cardinalidad Ej. 0,n_1,n

****

LISTA DE DOMINIOS.

Dominio 1

Nombre del Dominio: Dominio 1

Descripción del Dominio: Descripción para el dominio "Dominio 1"

Lista de Valores: {valor_0, valor_1, valor_2}

****

LISTA DE ROLES DEL MODELO.

Rol 1.

Nombre del Rol: Rol 1.

Descripción del Rol: Descripción para el Rol 1.

Privilegios Asignados al Rol: {privilegio 1, privilegio 2…, privilegio n}

****

LISTA DE USUARIOS PARA EL MODELO: "Modelo Conceptual de Datos".

Usuario 1.

Nombre del Usuario: Usuario 1.

Descripción del Usuario: Usuario 1.

Lista de Roles asignados al Usuario 1.

Nombre Rol	idRol	Descripción	Privilegios del Rol
Rol 1	idRol1	Descripción Rol 1	{Lista de Privilegios}

Lista de Privilegios asignados al Usuario 1.

Nombre Privilegio	idPrivilegio	Descripción
Privilegio 1	idPrivilegio1	Descripción Privilegio 1

APÉNDICE Ñ. API DE LA HERRAMIENTA.

Packages	
com.app	Contiene todas aquellas clases de configuración de la herramienta, autentificación de usuarios, control del sistema de archivos, controles de acceso.
com.conceptual	Contiene las clases de configuración de los modelos conceptuales de datos así como todas aquellas operaciones para manipular los objetos que los conforman.
com.interpretes	Contiene todas las clases para realizar los procesos de interpretación y traducción entre las etapas de diseño de bases de datos.
com.logico	Contiene las clases de configuración de los modelos Lógicos de datos así como todas aquellas operaciones para manipular los objetos que los conforman.

Package Details.

- Package com.app.

Class Summary	
ApplicationUser	Clase que implementa los objetos relacionados a usuarios dentro de la herramienta.
magoApplication	Clase que implementa la gestión de configuración interna de la herramienta.
paserXmlAppConfig	Clase que contiene los métodos de interpretación de los documentos XML de configuración de la herramienta.

---

com.app

Class ApplicationUser

java.lang.Object
└ **com.app.ApplicationUser**

---

public class **ApplicationUser**
extends java.lang.Object

Clase que implementa los objetos relacionados a usuarios dentro de la herramienta.

Field Summary	
`java.lang.String`	**conceptualPath** Ruta que contiene los modelos conceptuales de un usuario dentro su espacio de trabajo

java.lang.String	**idUsuario** Identificador de referencia del usuario
java.lang.String	**logicalPath** Ruta que contiene los modelos Lógicos de un usuario dentro su espacio de trabajo
java.lang.String	**loginUsuario** Nombre de acceso a la herramienta
java.lang.String	**nombreUsuario** Nombre del usuario a registrar
java.lang.String	**passwordUsuario** Contraseña de acceso a la herramienta
java.lang.String	**physicalPath** Ruta que contiene los modelos Físicos de un usuario dentro su espacio de trabajo

Constructor Summary
ApplicationUser(java.lang.String idUsuario, java.lang.String nombreUsuario, java.lang.String loginUsuario,java.lang.String passwordUsuario) Constructor de la clase ApplicationUser

Method Summary	
void	**constructConceptualPath**(java.lang.String rootPath) Método que construye la ruta y directorios de los archivos conceptuales de un usuario.
void	**constructLogicalPath**(java.lang.String rootPath) Método que construye la ruta y directorios de los archivos Lógicos de un usuario.
void	**constructPhysicalPath**(java.lang.String rootPath) Método que construye la ruta y directorios de los archivos Físicos de un usuario.
java.lang.String	**getUserConfigFile**() Método que construye la configuración Xml de los usuarios.

---

com.app

Class magoApplication

java.lang.Object

└ com.app.magoApplication

public class **magoApplication**
extends java.lang.Object

Clase que implementa la gestión de configuración interna de la herramienta.

Field Summary	
java.lang.String	**rootPath** Ruta raíz del sistema de archivos generados por la herramienta.
java.util.Vector	**usuariosAplicacion** Estructura que contiene los usuarios registrados en la aplicación.

Constructor Summary
magoApplication (java.lang.String rootPath) Constructor de la clase magoApplication

Method Summary	
void	**addApplicationUser** (ApplicationUser usr) Método que agrega un nuevo usuario a la herramienta.
java.lang.String	**getApplicationConfig** ( ) Método que obtiene la configuración XML de los objetos necesarios para la gestion de la heramienta
boolean	**validateSamePassword** (java.lang.String p1, java.lang.String p2) Método de validación
ApplicationUser	**validateUser** (java.lang.String login, java.lang.String password) Método que valida el acceso de usuarios a la herramienta.
boolean	**validateUserExistence** (java.lang.String login) Método que valida que garantiza la unicidad del nombre de usuario dentro de la herramienta.

com.app

Class paserXmlAppConfig

java.lang.Object
└ **com.app.paserXmlAppConfig**

public class **paserXmlAppConfig**
extends java.lang.Object

Constructor Summary
paserXmlAppConfig ( ) Constructor vacio de la clase paserXmlAppConfig.

Method Summary	
`java.lang.Object`	**ParserXmlAppConfig**(`java.lang.String ConfigXml`) Método que implementa el parser del documento XML que contiene la configuración de gestión de usuarios de la herramienta así como también la información de la ubicación física de sus espacios de trabajo.

- Package com.conceptual.

Class Summary	
Atributo	La clase Atributo crea los Objetos que representan a los Atributos contenidos en Entidades de un modelo de datos.
Consulta	La clase Consulta crea los Objetos que representan a una consulta dentro de un modelo conceptual de datos.
Dominio	La Clase Dominio implementa los objetos que corresponden a los dominios entre los cuales puede estar un atributo de una entidad en un Modelo Conceptual de Datos.
Entidad	La clase Entidad crea los Objetos que representan a Entidades dentro de un modelo de datos.
Modelo	La clase Modelo crea los Objetos que representan a modelos específicos de datos.
Modelos	La Clase Modelos implementa los objetos que corresponden al conjunto de modelos de un usuario, dentro de un modelo conceptual de Datos.

ParserXmlConceptual	Clase que implementa los métodos de parseo del archivo del modelo conceptual XML, para el llenado de objetos
ParserXmlUsers	La Clase ParserXmlUsers implementa los objetos que corresponden al parser del archivo XML con la configuración de los usuarios de una base de datos.
Privilegio	La clase Privilegio crea los Objetos que representa a los privilegios que puede tener un usuario dentro de los modelos.
Relación	La clase Relación crea los Objetos que representan a las relaciones entre entidades de un modelo conceptual de datos.
Rol	La clase Rol crea los Objetos que representan a un Rol que será asignado luego a un usuario.
Usuario	La clase Usuario crea los Objetos que representan a un usuario específico dentro de la herramienta.
Valor	La clase Valor crea los Objetos que representan a los valores que pertenecen a un dominio de un modelo conceptual de datos.
Versión	La clase Versión crea los Objetos que representan a la versión de un modelo conceptual de datos.

com.conceptual

Class Atributo

java.lang.Object
└ **com.conceptual.Atributo**

public class **Atributo**
extends java.lang.Object

La clase Atributo crea los Objetos que representan a los Atributos contenidos en Entidades de un modelo de datos.

Field Summary	
`java.lang.String`	**descripcionAtributo** Descripción del Atributo
`java.lang.String`	**dominio** Dominio al que hace referencia si tiene alguna restricción de Dominio.
`java.lang.String`	**idAtributoS** Identificador de referencia del atributo
`int`	**longitud** Tamaño del tipo de dato seleccionado para el atributo.
`java.lang.String`	**nombre** Nombre del atributo

java.lang.String	**opcionalidadAtributo** Opcionalidad del Atributo.
int	**primario** Indicador de si el atributo es identificador clave de la entidad.
java.lang.String	**tipoDeDatoReferencia** Tipo de Dato del Atributo.
int	**único** Indicador de si el atributo es único o no.

Constructor Summary

Atributo(java.lang.String nombre,java.lang.String idAtributoS, java.lang.String descripcionAtributo,java.lang.String opcionalidadAtributo, java.lang.String tipoDeDatoReferencia,int longitud,int primario, int unico, java.lang.String dominio)
Constructor de la clase Atributos

Method Summary

Atributo	**getAtributo**() Método get de la clase atributo.
java.lang.String	**getMetaModel**() Método de la clase Atributo que construye la porción del archivo XML generado para la configuración del modelo conceptual de datos.
void	**setAtributo**() Método set de la clase atributo

---

com.conceptual

Class Consulta

java.lang.Object
└ **com.conceptual.Consulta**

---

public class **Consulta**
extends java.lang.Object

La clase Consulta crea los Objetos que representan a una consulta dentro de un modelo conceptual de datos.

Field Summary	
`java.util.Hashtable`	**camposConsulta** Estructura que contiene los campos asociadas a una consulta.
`java.util.Vector`	**condicionConsulta** Estructura que contiene las condiciones asociadas a una consulta.
`java.lang.String`	**descripcionConsulta** Descripción de la versión de un modelo conceptual de datos.
`java.lang.String`	**idConsulta** Identificador de Referencia de la Consulta perteneciente a un modelo conceptual de datos.
`java.lang.String`	**nombreConsulta** Nombre de la Consulta de un modelo especifico.
`java.util.Hashtable`	**tablasConsulta** Estructura que contiene las tablas asociadas a una consulta.

Constructor Summary
Consulta(`java.lang.String idConsulta,java.lang.String nombreConsulta, java.lang.String descripcionConsulta`) Constructor de la Clase Consulta

Method Summary	
Consulta	**getConsulta**() Método get de la clase Consulta
`java.lang.String`	**getMetaModel**() Método de la clase Consulta que construye la porción del archivo XML generado para la configuración del modelo conceptual de datos en cuanto a Consultas.

com.conceptual

Class Dominio

java.lang.Object
└ **com.conceptual.Dominio**

public class **Dominio**
extends java.lang.Object

La Clase Dominio implementa los objetos que corresponden a los dominios entre los cuales puede estar un atributo de una entidad en un Modelo Conceptual de Datos.

Field Summary	
`int`	**idDominio** Identificador del dominio.
`java.lang.String`	**idDominioS** Identificador del dominio.
`java.lang.String`	**nombreDominio** Nombre del dominio a crear
`java.util.Vector`	**vectorValores** Estructura que almacena los valores posibles que puede tomar un atributo.

Constructor Summary
Dominio(`java.lang.String nombreDominio,java.lang.String idDominioS`) Constructor de la clase Dominio.

Method Summary	
`void`	**addValues**(Valor `val`) Método que añade objetos de la clase Valor a un Dominio.
`java.lang.String`	**concatAllValues**() Método que recibe un vector de valores y retorna dichos valores concatenados y separados por coma.
Dominio	**getDominio**() Método get de la clase Dominio
`java.lang.String`	**getMetaModel**() Método de la clase Dominio que construye la porción del archivo XML generado para la configuración del modelo conceptual de datos.
`void`	**setValues**()

	Método que modifica los valores dentro de un Dominio especifico.

com.conceptual

Class Entidad

java.lang.Object
└ **com.conceptual.Entidad**

public class **Entidad**
extends java.lang.Object

La clase Entidad crea los Objetos que representan a Entidades dentro de un modelo de datos.

Field Summary

`java.util.Vector`	**atributo** Estructura que almacena los atributos de una Entidad dentro de un Modelo.
`java.lang.String`	**descripcionEntidad** Descripción de la entidad creada.
`int`	**idEntidad** Identificador de referencia la entidad
`java.lang.String`	**idEntidadS** Descripcion de la entidad creada.
`java.lang.String`	**nombreEntidad** Nombre de la entidad dentro del modelo conceptual de datos

Constructor Summary

Entidad`(java.lang.String nombreEntidad,java.lang.String idEntidadS, java.lang.String descripcionEntidad)`
Constructor de la Clase Entidad

Method Summary

`void`	**addAtribute**(Atributo `att`) Método que Añade una atributo a una entidad especifica.

java.util.Vector	**getAllPrimaryKey**( ) Método que obtiene una lista de las claves primarias de un atributo
Atributo	**getAttributeByName**(java.lang.String nombreAtributo) Método que recupera un objeto de la clase Atributo dado el nombre del mismo
Entidad	**getEntidad**( ) Método get de la clase entidad.
java.lang.String	**getMetaModel**( ) Método de la clase Entidad que construye la porción del archivo XML generado para la configuración del modelo conceptual de datos.
void	**removeAtributo**(java.lang.String id) Método que elimina un atributo de una entidad.

---

com.conceptual

Class Modelo

java.lang.Object
└ **com.conceptual.Modelo**

---

public class **Modelo**
extends java.lang.Object

La clase Modelo crea los Objetos que representan a modelos específicos de datos.

Field Summary	
java.lang.String	**descripcionModelo** Descripción del modelo de datos.
int	**idModelo** Identificador del Modelo.
java.lang.String	**idModeloS** Identificador del Modelo.
java.lang.String	**nombreModelo** Nombre del Modelo conceptual de datos.
java.lang.String	**tipoModelo** Tipo del Modelo: Conceptual, Lógico o Físico.
java.util.Vector	**Versiónes** Estructura que almacena los atributos de la clase Versión de un Modelo Conceptual de Datos.

Constructor Summary
Modelo(java.lang.String nombreModelo,java.lang.String idModeloS, java.lang.String descripcionModelo,java.lang.String tipoModelo) Constructor de la clase Modelo.

Method Summary	
void	**addVersión**(Versión ver1) Método que agrega una versión a un modelo especifico de datos.
java.lang.String	**getMetaModel**() Método de la clase Modelo que construye la porción del archivo XML generado para la configuración del modelo conceptual de datos.
java.lang.String	**getMetaModelUsr**() Método de la clase Modelo que construye la porción del archivo XML generado para la configuración del modelo conceptual de datos.
java.lang.String	**getTodayDate**() Método que calcula la fecha del día en el cual se va a crear la Versión de un Modelo.
Versión	**getVersión**(java.lang.String id, boolean b) Método que recupera un objeto de la clase Versión a partir de un Id dado.
Versión	**getVersiónByIdName**(java.lang.String id, java.lang.String nombreVersión) Método que recupera un objeto de la clase Versión a partir de un Id y nombre dado.
void	**removeVersión**(Versión ver) Método de la clase Modelo Que elimina una Versión de un modelo conceptual dentro del espacio de trabajo de un usuario.
void	**setVersión**(Versión ver) Método de la clase Modelo Que modifica una Versión de un modelo conceptual dentro del espacio de trabajo de un usuario.

com.conceptual

Class Modelos

java.lang.Object
└ **com.conceptual.Modelos**

public class **Modelos**
extends java.lang.Object

La Clase Modelos implementa los objetos que corresponden al conjunto de modelos de un usuario, dentro de un modelo conceptual de Datos.

Field Summary	
`java.lang.String`	**idUsuario** Identificador del usuario propietario del modelo.
`int`	**IdUsuarioModelos** Identificador del usuario propietario del modelo.
`java.util.Vector`	**vectorModelos** Contiene la lista de modelos de un usuario.

Constructor Summary
Modelos ( ) Constructor de la clase Modelos
Modelos (`java.lang.String idUsuario`) Constructor de la clase Modelos

Method Summary	
`void`	**addModelo** (Modelo `mod`) Método que añade un modelo de datos a la lista de modelos de un usuario.
`java.lang.String`	**getMetaModel** ( ) Método de la clase Modelos que construye la porción del archivo XML generado para la configuración del modelo conceptual de datos.
`java.lang.String`	**getMetaModelUsr** ( ) Método de la clase Modelos que construye la porción del archivo XML generado para la configuración del modelo conceptual de datos.
Modelo	**getModelById** (`int id`) Método que recupera un objeto de la clase Modelo a partir de un IdModelo Dado
Modelo	**getModelByIdName** (`java.lang.String id, java.lang.String nombre`) Método que recupera un objeto de la clase Modelo a partir de un Id y Nombre del Modelo dado

Modelo	**getModelByStringId** (`java.lang.String id`) Método que recupera un objeto de la clase Modelo a partir de un IdModelo Dado
`void`	**removeModelo** (Modelo `mod`) Método para remover un modelo dentro del espacio de trabajos de un usuario.
`int`	**searchModel** (Modelo `mod`) Método que indica si un modelo existe o no dentro del espacio de trabajo de un usuario.
`void`	**setModelo** (Modelo `mod`) Método set de los objetos Modelo para realizar modificaciones o actualizar la información.
`void`	**writteFile** (`java.io.FileOutputStream out`) Método de escritura en archivo de la configuración Xml de los Modelos de la herramienta.
`void`	**writteFileU** (`java.io.FileOutputStream out`) Método de escritura en archivo de la configuración Xml de los Modelos de la herramienta.

---

com.conceptual

Class ParserXmlConceptual

java.lang.Object
└ **com.conceptual.ParserXmlConceptual**

---

public class **ParserXmlConceptual**
extends java.lang.Object

Clase que implementa los métodos de parseo del archivo del modelo conceptual XML, para el llenado de objetos

---

Constructor Summary
ParserXmlConceptual ( ) Constructor de la clase ParserXmlConceptual

Method Summary	
`java.lang.Object`	**ParserXmlCdm** (`java.lang.String CdmXml`)

	Método que implementa el parser del documento XML que contiene la configuración los modelos conceptuales creados por el usuario.

com.conceptual

Class ParserXmlUsers

java.lang.Object
└ **com.conceptual.ParserXmlUsers**

public class **ParserXmlUsers**
extends java.lang.Object

La Clase ParserXmlUsers implementa los objetos que corresponden al parser del archivo XML con la configuración de los usuarios de una base de datos.

Constructor Summary
ParserXmlUsers ( )
Constructor de la clase ParserXmlUsers.

Method Summary	
`java.lang.Object`	**ParserXmlUsr**(`java.lang.String UsrXml`, Modelos `modelosPar`) Método de la clase ParserXmlUsers que lee y carga en memoria el archivo xml con la configuración de los usuarios de la herramienta.

com.conceptual

Class Privilegio

java.lang.Object
└ **com.conceptual.Privilegio**

public class **Privilegio**
extends java.lang.Object

La clase Privilegio crea los Objetos que representa a los privilegios que puede tener un usuario dentro de los modelos.

Field Summary	
`java.lang.String`	**descripcionPrivilegio** Descripción del privilegio.
`java.lang.String`	**idPrivilegio** Identificador del privilegio.
`java.lang.String`	**nombrePrivilegio** Nombre del privilegio.

Constructor Summary
Privilegio(`java.lang.String idPrivilegio, java.lang.String nombrePrivilegio, java.lang.String descripcionPrivilegio`) Constructor de la clase privilegios.

Method Summary	
`java.lang.String`	**getMetaModel**() Método de la clase Privilegio que construye la porción del archivo XML generado para la configuración del modelo conceptual de datos.
Privilegio	**getPrivilegio**() Método get de la clase Privilegio

com.conceptual

Class Relación

java.lang.Object
└ **com.conceptual.Relacion**

public class **Relacion**
extends java.lang.Object

La clase Relación crea los Objetos que representan a las relaciones entre entidades de un modelo conceptual de datos.

Field Summary	
`java.lang.String`	**cardinalidad** Cardinalidad de la relación
`java.lang.String`	**descripcionRelacion** Descripción de la relación
`java.lang.String`	**entidadFin** Entidad a la que llega la relación.
`java.lang.String`	**entidadOrigen** Entidad de la que parte la relación.
`int`	**idRelacion** Identificador de la relación.
`java.lang.String`	**idRelacionS** Identificador de la relación.
`java.lang.String`	**nombreRelacion** Nombre de la relación entre las entidades en un modelo.
`java.lang.String`	**opcionalidadRelacion** Opcionalidad de la relación

Constructor Summary
Relacion(`java.lang.String nombreRelacion, java.lang.String idRelacionS, java.lang.String descripcionRelacion, java.lang.String entidadOrigen, java.lang.String entidadFin, java.lang.String cardinalidad, java.lang.String opcionalidadRelacion`) Constructor de la clase Relación.

Method Summary	
`java.lang.String`	**getMetaModel**() Método de la clase Relación que construye la porción del archivo XML generado para la configuración del modelo conceptual de datos.
Relacion	**getRelacion**() Método get de la clase Relación

com.conceptual
Class Rol

java.lang.Object
└ **com.conceptual.Rol**

public class **Rol**
extends java.lang.Object

La clase Rol crea los Objetos que representan a un Rol que sera asignado luego a un usuario.

Field Summary	
`java.lang.String`	**descripcionRol** Descripción del Rol que va a ser creado.
`java.lang.String`	**idRol** Identificador del Rol que va a ser creado.
`java.lang.String`	**nombreRol** Nombre del Rol que va a ser Creado.
`java.util.Vector`	**privilegios** Estructura que almacena los privilegios que pueden ser asignados a un usuario.

Constructor Summary
Rol(`java.lang.String idRol, java.lang.String nombreRol, java.lang.String descripcionRol`) Constructor de la clase Rol.

Method Summary	
`void`	**addPrivilegios**(Privilegio `pri`) Método que añade un Privilegio a la lista de privilegios de un usuario.
`java.lang.String`	**getMetaModel**() Método de la clase Rol que construye la porción del archivo XML generado

	para la configuración del modelo conceptual de datos.
Rol	**getRol**() Método get de la clase Rol
void	**removeRolePrivilege**(`java.lang.String id`) Método que elimina un privilegio dado de un rol especifico.
void	**setRolePrivileges**(`java.util.Vector privilegios`) Método que modifica los privilegios de un rol especifico.

---

com.conceptual

Class Usuario

java.lang.Object
└ **com.conceptual.Usuario**

---

public class **Usuario**
extends java.lang.Object

La clase Usuario crea los Objetos que representan a un usuario especifico dentro de la herramienta.

Field Summary	
`java.lang.String`	**idUsuario** Identificador del usuario dentro de la herramienta.
`java.lang.String`	**loginUsuario** Login del usuario de la Base de Datos creada por la herramienta.
`java.lang.String`	**nombreUsuario** Nombre del Usuario de la Base de Datos creada por la herramienta .
`java.lang.String`	**passwordUsuario** Clave de acceso de la Base de Datos creada por la herramienta.
`java.util.Vector`	**privilegios** Estructura que almacena los privilegios que pueden ser asignados a un usuario.
`java.util.Vector`	**roles** Estructura que almacena los roles que pueden ser asignados a un usuario.

Constructor Summary

Usuario(java.lang.String idUsuario,java.lang.String nombreUsuario, java.lang.String loginUsuario,java.lang.String passwordUsuario) Constructor de la clase Usuario

Method Summary	
void	**addPri**(Privilegio pri) Método que añade un Privilegio a la lista de privilegios de un usuario.
void	**addRol**(Rol rol) Método que añade un rol a la lista de roles de un usuario.
java.lang.String	**getMetaModel**() Método de la clase Usuario que construye la porción del archivo XML generado para la configuración del modelo conceptual de datos.
Usuario	**getUsuario**() Método get de la clase usuario.
void	**removeUserPrivilege**(java.lang.String id) Método que elimina un privilegio asignado a un usuario.
void	**removeUserRole**(java.lang.String id) Método que elimina un rol asignado a un usuario.
void	**setUserPrivilege**(java.lang.String id) Método que modifica un privilegio asignado a un usuario.
void	**setUserRole**(java.lang.String id) Método que modifica un rol asignado a un usuario.

---

com.conceptual

Class Valor

java.lang.Object
└ **com.conceptual.Valor**

---

public class **Valor**
extends java.lang.Object

La clase Valor crea los Objetos que representan a los valores que pertenecen a un dominio de un modelo conceptual de datos.

Field Summary	
`java.lang.String`	**valor** Es el valor que va a ser agregado a un dominio especifico dentro de un modelo.

Constructor Summary
Valor(`java.lang.String valor`) Constructor de la Clase Valor

Method Summary	
`java.lang.String`	**getMetaModel**() Método de la clase Valor que construye la porción del archivo XML generado para la configuración del modelo conceptual de datos.
Valor	**getValor**() Método get de la clase Valor

---

com.conceptual
Class Version

java.lang.Object
└ **com.conceptual.Version**

All Implemented Interfaces: java.lang.Cloneable

---

public class **Versión**
extends java.lang.Object
implements java.lang.Cloneable

La clase Versión crea los Objetos que representan a la version de un modelo conceptual de datos.

Field Summary	
`java.util.Vector`	**consultas**

	Estructura que almacena las Consultas de una Versión en un Modelo Conceptual de datos específico.
`java.lang.String`	**descripcionVersion** Descripción de la versión de un modelo conceptual de datos.
`java.util.Vector`	**dominios** Estructura que almacena los dominios de una Versión en un Modelo Conceptual de datos específico.
`java.util.Vector`	**entidad** Estructura que almacena las Entidades de una Versión en un Modelo Conceptual de datos Específico.
`java.lang.String`	**fechaCreacion** Fecha de Creación de la Versión del Modelo de Datos.
`java.lang.String`	**idVersionS** Identificador de Referencia de la versión perteneciente a un modelo conceptual de datos
`java.lang.String`	**nombreVersion** Nombre de la versión de un modelo especifico.
`java.util.Vector`	**relacion** Estructura que almacena las Relaciones entre entidades de una Versión en un Modelo Conceptual de datos especifico.
`java.util.Hashtable`	**tablaPrivilegios** Tabla que contiene los privilegios de la versión.
`java.util.Hashtable`	**tablaRoles** Tabla que contiene los roles de la versión.
`java.lang.String`	**ultimaModificacion** Fecha de la última modificación realizada sobre la versión del Modelo Conceptual de datos.
`java.util.Vector`	**usuarios** Estructura que almacena los Usuarios de una Versión en un Modelo Conceptual de datos específico.

Constructor Summary
Version(`java.lang.String idVersionS,java.lang.String nombreVersion, java.lang.String descripcionVersion,java.lang.String fechaCreacion, java.lang.String ultimaModificacion`) Constructor de la Clase Versión

Method Summary	
void	**addConsulta** (Consulta con) Método que Añade una consulta a una Versión especifica de un Modelo
void	**addDominio** (Dominio dom) Método que Añade un Dominio a una Versión especifica de un Modelo
void	**addEntidad** (Entidad ent) Método que Añade una Entidad a una Versión especifica de un Modelo
void	**addRelacion** (Relación rel) Método que Añade una Relación a una Versión especifica de un Modelo
void	**addUsuario** (Usuario usu) Método que Añade un Usuario a una Versión especifica de un Modelo
java.lang.Object	**clone** () Implementación del método clone para la interfaz Cloneable
java.lang.String	**generaReporteConceptual** () Método que construye el reporte de diccionario de datos conceptual de una versión especifica
Consulta	**getConsultaById** (java.lang.String id) Método que obtiene una Consulta a partir de su identificador de referencia.
Dominio	**getDominioById** (java.lang.String id) Método que obtiene un Dominio a partir de su identificador de referencia.
Entidad	**getEntidadByIdName** (java.lang.String id) Método que obtiene una entidad a partir de su identificador de referencia.
Entidad	**getEntidadByName** (java.lang.String NombreEntidad) Método que recupera un objeto de la clase entidad dado el nombre de la misma.
java.lang.String	**getMetaModel** () Método de la clase Versión que construye la porción del archivo XML generado para la configuración del modelo conceptual de datos.
java.lang.String	**getMetaModelUsr** () Método de la clase Versión que construye la porción del archivo XML generado para la configuración del modelo conceptual de datos en cuanto a usuarios.
Relacion	**getRelacionById** (java.lang.String id) Método que obtiene una Relación a partir de su identificador de referencia.
Usuario	**getUsuarioById** (java.lang.String id) Método que obtiene un Usuario a partir de su identificador de referencia.
java.util.Vector	**llenaTablaPrivilegios** () Método que construye la tabla que contiene todos los privilegios creados para una versión.
java.util.Vector	**llenaTablaRoles** () Método que construye la tabla que contiene todos los roles creados para una versión.
void	**removeDominio** (java.lang.String id) Método que elimina un dominio dentro de un modelo conceptual de datos.

void	**removeEntidad**(java.lang.String id) Método que elimina una entidad dentro de un modelo conceptual de datos.
void	**removeRelacion**(java.lang.String id) Método que elimina una relación dentro de un modelo conceptual de datos.
void	**removeUsuario**(java.lang.String id) Método que elimina un usuario dentro de un modelo conceptual de datos.
void	**setDominio**(java.lang.String id) Método que modifica un dominio especifica dentro de un modelo conceptual de datos.
void	**setEntidad**(java.lang.String id) Método que modifica una entidad especifica dentro de un modelo conceptual de datos.
void	**setRelacion**(java.lang.String id) Método que modifica una relación especifica dentro de un modelo conceptual de datos.
void	**setUsuario**(java.lang.String id) Método que modifica un Usuario especifica dentro de un modelo conceptual de datos.
Version	**setVersion**(java.lang.String nombre, java.lang.String descripción) Método de la clase Versión que modifica el valor de una versión en especifico.
void	**writteFile**(java.io.FileOutputStream out) Método de escritura en archivo del reporte que se esta generando.

- Package com.interpretes.

Class Summary	
interpreteFisico	Clase que implementa los métodos de traducción entre las fases lógica y física del diseño de bases de datos.
interpreteLogico	Clase que implementa los métodos de traducción entre las fases conceptual y lógica del diseño de bases de datos.
interpreteReversa	Clase que implementa los métodos de traducción entre las fases física y lógica del diseño de bases de datos.

com.interpretes

Class interpreteFisico

java.lang.Object
└ **com.interpretes.interpreteFisico**

public class **interpreteFisico**
extends java.lang.Object

Clase que implementa los métodos de traducción entre las fases lógica y física del diseño de bases de datos.

Constructor Summary
interpreteFisico ( ) Constructor de la clase interpreteFisico.

Method Summary	
static java.lang.String	**generaFisico** (java.util.Hashtable hash, VersionLogica verLog) Método que Genera el archivo de creación de base de datos asociado a una versión Lógica de un modelo de datos.
static void	**generaFisicoQuery** (java.util.Hashtable hash, VersionLogica verLog) Método que Genera el archivo de creación de consultas asociado a una versión Lógica de un modelo de datos.
static java.lang.String	**generaFisicoUsr** (java.util.Hashtable hash, VersionLogica verLog) Método que Genera el archivo de creación de Usuarios, roles y privilegios asociado a una versión Lógica de un modelo de datos.
static java.util.Hashtable	**parserXmlConfig** (java.lang.String ConfigXml) Método que interpreta el archivo XML con la configuración de la sintaxis del sistema manejador de base de datos.
static void	**writteFile** (java.io.OutputStream out, java.lang.String sen)

com.interpretes

Class interpreteLogico

java.lang.Object
└ **com.interpretes.interpreteLogico**

public class **interpreteLogico**
extends java.lang.Object

Constructor Summary
interpreteLogico ( ) Constructor de la clase interpreteLogico.

Method Summary	
static VersionLogica	**generaLogico** (Version version) Método que realiza la transformación de una versión conceptual de un modelo de datos, en una versión lógica.

com.interpretes

Class interpreteReversa

java.lang.Object
└ **com.interpretes.interpreteReversa**

All Implemented Interfaces:

interpreteReversaConstants

---

public class **interpreteReversa**
extends java.lang.Object
implements interpreteReversaConstants

Field Summary	
static Token	**jj_nt**
static boolean	**lookingAhead**
static Token	**token**
static interpreteReversaTokenManager	**token_source**

Constructor Summary
interpreteReversa ( ) Constructor de la clase interpreteReversa.

interpreteReversa(java.io.InputStream stream)
interpreteReversa(interpreteReversaTokenManager tm)
interpreteReversa(java.io.Reader stream)

Method Summary	
static void	**alterTable**(VersionLogica verLog)
static java.util.Vector	**Atributo**(java.util.Vector vAtt)
static void	**axioma**(Modelo mod, VersionLogica verLog)
static void	**ck**(java.lang.String nombreTabla, VersionLogica verLog, java.lang.String nombreConst)
static void	**constraint**(java.lang.String nombreTabla, VersionLogica verLog)
static void	**createIndex**(VersionLogica verLog)
static void	**createTable**(VersionLogica verLog)
static void	**disable_tracing**()
static void	**dropTable**()
static void	**enable_tracing**()
static void	**fk**(java.lang.String nombreTabla, VersionLogica verLog, java.lang.String nombreConst)
static ParseException	**generateParseException**()
static Token	**getNextToken**()
static Token	**getToken**(int index)
static VersionLogica	**inicio**(java.lang.String args)

static void	**main**(java.lang.String[] args)
static void	**pk**(java.lang.String nombreTabla, VersionLogica verLog, java.lang.String nombreConst)
static void	**ReInit**(java.io.InputStream stream)
void	**ReInit**(interpreteReversaTokenManager tm)
static void	**ReInit**(java.io.Reader stream)

- Package com.logico.

Class Summary	
Atributo	La Clase Atributo implementa los objetos que corresponden a los atributos que poseen las Tablas dentro de los esquemas Logicos de base de datos.
Check	La clase Check crea los Objetos que representan las restricciones de Dominio de las columnas en un modelo de datos, extiende de la clase restricción.
Consulta	La clase Consulta crea los Objetos que representan a una consulta dentro de un modelo logico de datos.
Dominio	La Clase Dominio implementa los objetos que corresponden a los dominios entre los cuales puede estar una columna de una Tabla en un Modelo Logico de Datos.
Foranea	La clase Foranea crea los Objetos que representan las restricciones de clave foranea de las columnas en un modelo de datos extiende de la clase restriccion.
Modelo	La clase Modelo crea los Objetos que representan a un modelo logico de datos especificos de datos.
Modelos	La Clase Modelos implementa los objetos que corresponden al conjunto de modelos logicos de un usuario.
ParserXmlLogico	Clase que implementa los metodos de parseo del archivo del modelo logico XML, para el llenado de objetos.
ParserXmlUsers	La clase ParserXmlUsers crea los Objetos de inteprete para los archivos XML con la configuracion de usuarios.
Primary	La clase Primary crea los Objetos que representan las restricciones de clave primaria de las columnas en un modelo de datos extiende de la clase restriccion.
Privilegio	La clase Privilegio crea los Objetos que representa a los privilegios que puede tener un usuario dentro de los modelos.

Restriccion	La clase Restriccion crea los Objetos que representa a las restricciones posibles a las columnas de una Tabla.
Rol	La clase Rol crea los Objetos que representan a un Rol que sera asignado luego a un usuario.
Tabla	La clase Tabla crea los Objetos que representan a las Tablas de un modelo logico de datos.
Unique	La clase Unique crea los Objetos que representan las restricciones de unicidad de las columnas en un modelo de datos extiende de la clase restriccion.
Usuario	La clase Usuario crea los Objetos que representan a un usuario especifico dentro de la herramienta.
Valor	La clase Valor crea los Objetos que representan los Valores dentro de un Dominio.
VersionLogica	La clase Version crea los Objetos que representan a la version de un modelo logico de datos.

com.logico

Class Atributo

java.lang.Object
└ **com.logico.Atributo**

public class **Atributo**
extends java.lang.Object

La Clase Atributo implementa los objetos que corresponden a los atributos que poseen las entidades dentro de los esquemas conceptuales de base de datos.

Field Summary	
`java.lang.String`	**descripcionAtributo** Descripción de la columna
`java.lang.String`	**dominio** Dominio al que esta asociado
`java.lang.String`	**idAtributoS** Identificador de referencia de la columna
`int`	**longitud** Espacio en memoria que ocupara esa columna
`java.lang.String`	**nombre** Nombre de la columna
`java.lang.String`	**opcionalidadAtributo** Define si la columna es opcional o no
`java.util.Vector`	**restriccion**

	Estructura que almacena las restricciones de una columna
`java.lang.String`	**tipoDeDatoReferencia** Tipo de dato de la columna

Constructor Summary
Atributo(`java.lang.String nombre,java.lang.String idAtributoS, java.lang.String descripcionAtributo,java.lang.String opcionalidadAtributo, java.lang.String tipoDeDatoReferencia,int longitud,java.lang.String dominio`) Constructor de la clase Atributo.

Method Summary	
`void`	**addRestriccion**(Restriccion `res`) Método que añade una restricción a una columna especifica.
`java.lang.String`	**getMetaModel**() Método de la clase Atributo que construye la porción del archivo XML generado para la configuración del modelo Lógico de datos.

---

com.logico

Class Check

java.lang.Object
└ com.logico.Restriccion
  └ **com.logico.Check**

---

public class **Check**
extends Restriccion

La clase Check crea los Objetos que representan las restricciones de Dominio de las columnas en un modelo de datos, extiende de la clase restricción.

Field Summary	
`java.lang.String`	**dominio** nombre del dominio al que hace referencia la restricción Check

Constructor Summary
Check(java.lang.String idRestriccion, java.lang.String nombreRestriccion, java.lang.String dominio) Constructor de la Subclase Check

Method Summary	
java.lang.String	**getMetaModelLogic**() Método abstracto que será implementado por las subinterfaces de restricción

com.logico

Class Consulta

java.lang.Object
└ **com.logico.Consulta**

public class **Consulta**
extends java.lang.Object

La clase Consulta crea los Objetos que representan a una consulta dentro de un modelo lógico de datos.

Field Summary	
java.util.Hashtable	**camposConsulta** Estructura que contiene los campos asociadas a una consulta.
java.util.Vector	**condicionConsulta** Estructura que contiene las condiciones asociadas a una consulta.
java.lang.String	**descripcionConsulta** Descripción de la consulta de un modelo lógico de datos.
java.lang.String	**idConsulta** Identificador de Referencia de la Consulta perteneciente a un modelo lógico de datos.
java.lang.String	**nombreConsulta** Nombre de la Consulta de un modelo especifico.
java.util.Hashtable	**tablasConsulta** Estructura que contiene las tablas asociadas a una consulta.

Constructor Summary
Consulta(java.lang.String idConsulta,java.lang.String nombreConsulta, java.lang.String descripcionConsulta) Constructor de la Clase Consulta

Method Summary	
void	**addColumn2Query**(java.lang.String nombreColumna, java.lang.String nombreTablaOrigen) Metodo que Anade columnas a una consulta
void	**addCondition2Query**(java.lang.String condicion) Metodo que Anade Condiciones a una consulta
void	**addTable2Query**(java.lang.String nombreTabla) Metodo que Anade tablas a una consulta
java.lang.String	**getAllQueryColumns**() Metodo que concatena todas las columnas de una consulta para su impresion.
java.lang.String	**getAllQueryConditions**() Metodo que concatena todas las Condiciones de una consulta para su impresion.
java.lang.String	**getAllQueryTables**() Metodo que concatena todas las Tablas de una consulta para su impresion.
java.lang.String	**getMetaModel**() Metodo que construye el String XML de la configuracion de Consulta

com.logico

Class Dominio

java.lang.Object
└ **com.logico.Dominio**

public class **Dominio**
extends java.lang.Object

La Clase Dominio implementa los objetos que corresponden a los dominios entre los cuales puede estar una columna de una Tabla en un Modelo Logico de Datos.

Field Summary	
int	**idDominio** Identificador del dominio.
java.lang.String	**idDominioS** Identificador del dominio.
java.lang.String	**nombreDominio** Nombre del Dominio.
java.util.Vector	**vectorValores** Estructura que almacena los valores posibles que puede tomar un atributo.

Constructor Summary
Dominio(java.lang.String nombreDominio,java.lang.String idDominioS) Constructor de la clase Dominio Logico.

Method Summary	
void	**addValues**(Valor val) Metodo que anade valores a un dominio.
java.lang.String	**concatAllValues**(java.util.Vector values) Metodo que recibe un vector de valores y retorna dichos valores concatenados y separados por coma.
java.lang.String	**getDomainName**() Metodo que devuelve el nombre de un Dominio.
java.lang.String	**getMetaModel**() Metodo de la clase Dominio que construye la porcion del archivo XML generado para la configuracion del modelo logico de datos.

com.logico

Class Foranea

java.lang.Object
└ com.logico.Restriccion
  └ **com.logico.Foranea**

public class **Foranea**
extends Restriccion

La clase Foránea crea los Objetos que representan las restricciones de clave foránea de las columnas en un modelo de datos extiende de la clase restricción.

Field Summary	
`java.lang.String`	**atribRef** Columna a la que referencia.
`java.lang.String`	**tablaRef** Tabla a la que referencia.

Constructor Summary
Foranea(`java.lang.String idRestriccion, java.lang.String nombreRestriccion, java.lang.String tablaRef, java.lang.String atribRef`) Constructor de la clase Foranea

Method Summary	
`java.lang.String`	**getMetaModelLogic**() Método de la clase Foránea que construye la porción del archivo XML generado para la configuración del modelo lógico de datos.

com.logico
Class Modelo

java.lang.Object
└ **com.logico.Modelo**

public class **Modelo**
extends java.lang.Object

La clase Modelo crea los Objetos que representan a un modelo lógico de datos específicos de datos.

Field Summary	
java.lang.String	**descripcionModelo** Descripción del Modelo Lógico de datos.
int	**idModelo** Identificador de referencia del modelo lógico de datos.
java.lang.String	**idModeloS** Identificador de referencia del modelo lógico de datos.
java.lang.String	**nombreModelo** Nombre del Modelo Lógico de Datos.
java.lang.String	**tipoModelo** Tipo de Modelo: conceptual, lógico o físico.
java.util.Vector	**versiones** Estructura que almacena las versiones de un modelo lógico de datos.

Constructor Summary
Modelo(java.lang.String nombreModelo,java.lang.String idModeloS, java.lang.String descripcionModelo,java.lang.String tipoModelo) Constructor de la clase Modelo Lógico.

Method Summary	
void	**addVersion**(VersionLogica ver1) Método que asocia una versión a un modelo lógico de datos.
java.lang.String	**getMetaModelLogic**() Método de la clase Modelo que construye la porción del archivo XML generado para la configuración del modelo lógico de datos.
java.lang.String	**getMetaModelUsr**() Método de la clase Modelo que construye la porción del archivo XML generado para la configuración del modelo lógico de datos, en relación a los usuarios.
VersionLogica	**getVersion**(java.lang.String id) Método para recuperar una versión especifica de un modelo de datos, dado su Identificador de referencia.
VersionLogica	**getVersionByIdName**(java.lang.String id, java.lang.String nombreVersion) Método para recuperar una versión especifica de un modelo de datos, dado su

	Identificador de referencia y su nombre.

---

com.logico

Class Modelos

java.lang.Object
└ **com.logico.Modelos**

---

public class **Modelos**
extends java.lang.Object

La Clase Modelos implementa los objetos que corresponden al conjunto de modelos logicos de un usuario.

Field Summary	
`int`	**IdUsuarioModelos** Identificador de referencia del usuario propietario de los modelos.
`java.lang.String`	**IdUsuarioModelosS** Identificador de referencia del usuario propietario de los modelos.
`java.util.Vector`	**vectorModelos** Estructura que contiene todos los modelos logicos de un usuario.

Constructor Summary
Modelos(`java.lang.String IdUsuarioModelosS`) Constructor de la clase Modelos

Method Summary	
`void`	**addModelo**(Modelo `mod`) Metodo que anade un modelo de datos a la lista de modelos de un usuario.
`java.lang.String`	**getMetaModelLogic**() Metodo de la clase Modelos que construye la porcion del archivo XML generado para la configuracion de los modelos logicos de datos.
`java.lang.String`	**getMetaModelUsr**() Metodo de la clase Modelos que construye la porcion del archivo XML generado para la configuracion del modelo logico de datos en cuanto a usuarios.

Modelo	**getModelById**(`java.lang.String id`) Metodo que busca un modelo logico dado su Identificador de Referencia.
Modelo	**getModelByIdName**(`java.lang.String id, java.lang.String nombre`) Metodo que busca un modelo logico dado su Identificador de Referencia.
`void`	**writteFile**(`java.io.OutputStream out`) Metodo de escritura en archivo invocando a la funcion getMetamodelLogic().

---

com.logico

Class ParserXmlLogico

java.lang.Object
└ **com.logico.ParserXmlLogico**

---

public class **ParserXmlLogico**
extends java.lang.Object

Clase que implementa los métodos de parseo del archivo del modelo lógico XML, para el llenado de objetos.

Constructor Summary
ParserXmlLogico( ) Constructor de la clase ParserXmlLogico.

Method Summary	
`java.lang.Object`	**ParserXmlLdm**(`java.lang.String LdmXml`) Método que implementa el parser del documento XML que contiene la configuración los modelos conceptuales creados por el usuario

---

com.logico

Class ParserXmlUsers

java.lang.Object
└ **com.logico.ParserXmlUsers**

---

public class **ParserXmlUsers**
extends java.lang.Object

La clase ParserXmlUsers crea los Objetos de interprete para los archivos XML con la configuración de usuarios.

Constructor Summary
ParserXmlUsers ( ) Constructor de la clase ParserXmlUsers.

Method Summary	
`java.lang.Object`	**ParserXmlUsr**(`java.lang.String UsrXml`, Modelos `modelosPar`) Método de la clase ParserXmlUsers que lee y carga en memoria el archivo xml con la configuración de los usuarios de la herramienta.

com.logico

Class Primary

java.lang.Object
└ com.logico.Restriccion
  └ **com.logico.Primary**

public class **Primary**
extends Restriccion

La clase Primary crea los Objetos que representan las restricciones de clave primaria de las columnas en un modelo de datos extiende de la clase restricción.

Field Summary

Fields inherited from class com.logico.Restriccion
idRestriccion, nombreRestriccion

Constructor Summary
Primary(`java.lang.String idRestriccion, java.lang.String nombreRestriccion`) Constructor de la clase Primary

Method Summary	
`java.lang.String`	**getMetaModelLogic**() Método de la clase Primary que construye la porción del archivo XML generado para la configuración del modelo lógico de datos.

---

com.logico

Class Privilegio

java.lang.Object
└ **com.logico.Privilegio**

---

public class **Privilegio**
extends java.lang.Object

La clase Privilegio crea los Objetos que representa a los privilegios que puede tener un usuario dentro de los modelos.

Field Summary	
java.lang.String	**descripcionPrivilegio** Descripción del privilegio.
java.lang.String	**idPrivilegio** Identificador de referencia del privilegio.
java.lang.String	**nombrePrivilegio** Nombre del privilegio.

Constructor Summary
Privilegio(java.lang.String idPrivilegio, java.lang.String nombrePrivilegio,

java.lang.String descripcionPrivilegio) Constructor de la clase Privilegio

Method Summary	
java.lang.String	**getMetaModel** ( ) Método de la clase Privilegio que construye la porción del archivo XML generado para la configuración del modelo conceptual de datos.

---

com.logico

Class Restricción

java.lang.Object
└ **com.logico.Restriccion**

Direct Known Subclasses:

Check, Foranea, Primary, Unique

---

public abstract class **Restriccion**
extends java.lang.Object

La clase Restricción crea los Objetos que representa a las restricciones posibles a las columnas de una Tabla.

Field Summary	
`java.lang.String`	**idRestriccion** Identificador de Referencia de la Restricción.
`java.lang.String`	**nombreRestriccion** nombre de la Restricción

Constructor Summary

Restriccion(`java.lang.String idRestriccion, java.lang.String nombreRestriccion`) Constructor de la clase Restricción.

Method Summary	
abstract `java.lang.String`	**getMetaModelLogic**() Método abstracto que será implementado por las subinterfaces de restricción

---

com.logico

Class Rol

java.lang.Object
└ **com.logico.Rol**

---

public class **Rol**
extends java.lang.Object

La clase Rol crea los Objetos que representan a un Rol que será asignado luego a un usuario.

Field Summary	
`java.lang.String`	**descripcionRol** Descripción del Rol que va a ser creado.
`java.lang.String`	**idRol** Identificador del Rol que va a ser creado.
`java.lang.String`	**nombreRol** Nombre del Rol que va a ser Creado.
`java.util.Vector`	**privilegios** Estructura que almacenas los privilegios asignados a un rol.

Constructor Summary
Rol(`java.lang.String idRol,java.lang.String nombreRol,`

`java.lang.String descripcionRol)` Constructor de la clase Rol.

Method Summary	
`void`	**addPrivilegios** (Privilegio `pri`) Método que añade un Privilegio a la lista de privilegios de un usuario.
`java.lang.String`	**concatAllPrivilegeNames** () Método que concatena los nombres de los privilegios asignados a un rol, para su posterior escritura.
`java.lang.String`	**getMetaModel** () Método de la clase Rol que construye la porción del archivo XML generado para la configuración del modelo conceptual de datos.

---

com.logico

Class Tabla

java.lang.Object
└ **com.logico.Tabla**

---

public class **Tabla**
extends java.lang.Object

La clase Tabla crea los Objetos que representan a las Tablas de un modelo lógico de datos.

Field Summary	
`java.util.Vector`	**atributo** Estructura que almacena la lista de columnas dentro de una tabla.
`java.lang.String`	**descripcionTabla** Descripción de la Tabla creada.
`int`	**idTabla** Identificador de referencia la Tabla
`java.lang.String`	**idTablaS** Identificador de referencia la Tabla
`java.lang.String`	**nombreTabla** Nombre de la Tabla dentro del modelo Lógico de datos

Constructor Summary
Tabla(`java.lang.String nombreTabla, java.lang.String idTablaS, java.lang.String descripcionTabla`) Constructor de la Clase Tabla

Method Summary	
`void`	**addAtribute**(Atributo `att`) Método que Añade una atributo a una Tabla especifica.
`java.lang.String`	**concatAllpk**(`java.util.Vector vec`) Método que recibe como parámetro un vector de claves primarias y las concatena como strings separadas por coma.
Atributo	**getAttributeByName**(`java.lang.String nombreAtributo`) Método que obtiene un objeto Atributo asociado a una tabla, a partir de su nombre.
`java.lang.String`	**getMetaModel**() Método de la clase Tabla que construye la porción del archivo XML generado para la configuración del modelo conceptual de datos.
`java.util.Hashtable`	**searchFkConstraints**(Tabla `tab`) Método que a partir de una tabla dentro de un modelo lógico de datos, devuelve una estructura con sus restricciones de clave foránea.
`java.util.Hashtable`	**searchPkConstraints**(Tabla `tab`) Método que a partir de una tabla dentro de un modelo lógico de datos, devuelve una estructura con sus restricciones de clave primaria.

---

com.logico

Class Unique

java.lang.Object
└ com.logico.Restriccion
  └ **com.logico.Unique**

---

public class **Unique**
extends Restriccion

La clase Unique crea los Objetos que representan las restricciones de unicidad de las columnas en un modelo de datos extiende de la clase restricción.

Field Summary

Fields inherited from class com.logico.Restriccion
idRestriccion, nombreRestriccion

Constructor Summary

Unique(java.lang.String idRestriccion, java.lang.String nombreRestriccion) Constructor de la Clase Unique

Method Summary

java.lang.String	**getMetaModelLogic**() Método de la clase Unique que construye la porción del archivo XML generado para la configuración del modelo lógico de datos.

com.logico

Class Usuario

java.lang.Object
└ **com.logico.Usuario**

public class **Usuario**
extends java.lang.Object

La clase Usuario crea los Objetos que representan a un usuario específico dentro de la herramienta.

Field Summary

java.lang.String	**idUsuario** Identificador del usuario dentro de la herramienta.
java.lang.String	**loginUsuario** Login del usuario de la herramienta.
java.lang.String	**nombreUsuario** Nombre del Usuario de la herramienta.
java.lang.String	**passwordUsuario** Clave de acceso del usuario para la herramienta.
java.util.Vector	**privilegios** Estructura que almacena Lista de Privilegios que posee ese usuario en especifico.
java.util.Vector	**roles** Estructura que almacena Lista de Roles que posee ese usuario en especifico.

Constructor Summary
Usuario(java.lang.String idUsuario,java.lang.String nombreUsuario, java.lang.String loginUsuario,java.lang.String passwordUsuario) Constructor de la clase Usuario

Method Summary	
void	**addPri**(Privilegio pri) Método que añade un Privilegio a la lista de privilegios de un usuario.
void	**addRol**(Rol rol) Método que añade un rol a la lista de roles de un usuario.
java.lang.String	**concatAllPrivilegeNames**() Método que concatena los nombres de los privilegios asignados a un usuario, para su posterior escritura.
java.lang.String	**concatAllRoleNames**() Método que concatena los nombres de los roles asignados a un usuario, para su posterior escritura.
java.lang.String	**getMetaModel**() Método de la clase Usuario que construye la porción del archivo XML generado para la configuración del modelo conceptual de datos.

com.logico

Class Valor

java.lang.Object
└ **com.logico.Valor**

---

public class **Valor**
extends java.lang.Object

La clase Valor crea los Objetos que representan los Valores dentro de un Dominio.

Field Summary	
`java.lang.String`	**valor** Valor a ser chequeado dentro de una restricción de dominio sobre un atributo

Constructor Summary
Valor(`java.lang.String valor`) Constructor de la Clase Valor

Method Summary	
`java.lang.String`	**getMetaModel**() Método de la clase Valor que construye la porción del archivo XML generado para la configuración del modelo lógico de datos.

---

com.logico

Class VersionLogica

java.lang.Object
└ **com.logico.VersionLogica**

---

public class **VersionLogica**
extends java.lang.Object

La clase Versión crea los Objetos que representan a la versión de un modelo lógico de datos.

Field Summary	
`java.util.Vector`	**consultas** Estructura que almacena las Consultas de una Versión en un Modelo Lógico de datos especifico.
`java.lang.String`	**descripcionVersionLogica** Descripción de la versión de un modelo lógico de datos.
`java.util.Vector`	**dominios** Estructura que almacena los dominios de una Versión en un Modelo Lógico de datos especifico.
`java.lang.String`	**fechaCreacion** Fecha de Creación de la Versión del Modelo de Datos.
`java.lang.String`	**idVersionLogicaS** Identificador de Referencia de la versión perteneciente a un modelo lógico de datos
`java.lang.String`	**nombreVersionLogica** Nombre de la versión de un modelo especifico.
`java.util.Hashtable`	**privilegiosSistema** Estructura que almacena privilegios del sistema.
`java.util.Hashtable`	**rolesSistema** Estructura que almacena Roles del sistema.
`java.util.Vector`	**tablas** Estructura que almacena las Tablas de una Versión en un Modelo Lógico de datos Especifico.
`java.lang.String`	**ultimaModificacion** Fecha de la ultima modificación realizada sobre la versión del Modelo Lógico de datos.
`java.util.Vector`	**usuarios** Estructura que almacena los Usuarios de una Versión en un Modelo Lógico de datos especifico.

Constructor Summary
VersionLogica(`java.lang.String idVersionLogicaS, java.lang.String nombreVersionLogica, java.lang.String descripcionVersionLogica, java.lang.String fechaCreacion, java.lang.String ultimaModificacion`) Constructor de la Clase VersionLogica

Method Summary	
void	**addConsulta** (Consulta con) Método que Añade una Consulta a una Versión Lógica especifica de un Modelo
void	**addDominio** (Dominio dom) Método que Añade una un objeto del tipo Dominio a una Versión Lógica especifica de un Modelo
void	**addTabla** (Tabla tab) Método que Añade una Tabla a una VersionLogica especifica de un Modelo
void	**addUser** (Usuario usu) Método que Añade una un objeto del tipo Usuario a una Versión Lógica especifica de un Modelo
Dominio	**getDominioByName** (java.lang.String nomDom) Método que Recupera un objeto de la Clase Dominio a partir de su nombre.
java.lang.String	**getMetaModel** () Método de la clase VersionLogica que construye la porción del archivo XML generado para la configuración del modelo lógico de datos.
java.lang.String	**getMetaModelUsr** () Método de la clase VersionLogica que construye la porción del archivo XML generado para la configuración del modelo lógico de datos en cuanto a usuarios.
Tabla	**getTablaByName** (java.lang.String nomTab) Método que Recupera un objeto de la Clase Tabla a partir de su nombre.
Tabla	**getTableByName** (java.lang.String nombreTabla) Método que Recupera un objeto de la Clase Tabla a partir de su nombre.

APÉNDICE O. DICCIONARIO DE DATOS

Nombre	Descripción
ENTIDAD	Una entidad es un objeto concreto o abstracto que presenta interés para el sistema y sobre el que se recoge información la cual va a ser representada en un sistema de base de datos. La mayoría de las entidades modelan objetos o eventos del mundo real, por ejemplo, clientes, productos o llamadas de pedidos.
RELACION	Objeto del modelo conceptual, que representa la asociación entre entidades. Mediante estas se analiza n sus componentes para la posterior traducción al modelo relacional de datos.
ATRIBUTO	Es una unidad básica e indivisible de información acerca de una entidad o una relación y sirve para identificar y describir a las mismas.
USUARIO	La entidad USUARIO representa tanto a aquellos usuarios propietarios de modelos de datos dentro de la herramienta, o una colección de objetos y privilegios identificado con un nombre y password
MODELO	La entidad modelo al ser instanciada representa un modelo de datos, creado por el usuario, bien sea en conceptual o relacional.
VERSION	La entidad Versión representa aquellas versiones que los usuarios pueden realizar de sus modelos de datos, para su posterior manipulación.
RESTRICCION	Constituyen el mecanismo primordial para especificar ciertas reglas que deben cumplir los datos almacenados en la base de datos. Esta relacionada con aquellas restricciones de integridad derivadas del esquema relacional Puede ser utilizada para definir restricciones de integridad de identidad, restricciones de integridad referencial, restricciones de dominio en las columnas de una tabla o inclusive ciertas restricciones explícitas.
ROL	La entidad ROL representa un conjunto de privilegios, asignables a un usuario o un rol.

PRIVILEGIO	La entidad PRIVILEGIO representa el o los permisos para realizar una acción, asignable a un usuario o un rol
USUARIO_ROL	Entidad de Intersección entre las entidades USUARIO y ROL
ROL_PRIVILEGIO	Entidad de Intersección entre las entidades ROL y PRIVILEGIO
CONSULTA	La entidad CONSULTA representa las preguntas que se formularan a la a la base de datos con el fin de extraer y presentar la información resultante.
CLAVE_FORANEA	La Entidad CLAVE_FORANEA representa las restricciones de clave foránea que se aplica sobre columnas de una tabla.
USUARIO_PRIVILEGIO	Entidad de Intersección entre las entidades USUARIO y PRIVILEGIO
CLAVE_PRIMARIA	La Entidad CLAVE_PRIMARIA representa las restricciones de primaria que se aplica sobre columnas de una tabla.
TABLA	La entidad TABLA representa la unidad donde se crean el conjunto de datos de una base de datos. Estos datos estarán ordenados en columnas verticales. Estos definidos mediante campos y sus características.
UNIQUE	La Entidad UNIQUE representa las restricciones de Unicidad que se aplica sobre columnas de una tabla.
CHECK	La Entidad CHECK representa las restricciones de dominio que se aplica sobre columnas de una tabla.
COLUMNA	En ella se especifican todo lo referente a los datos que se almacenan dentro de las tablas del modelo lógico de datos.
DOMENIO	Representa conjunto de valores de un mismo tipo entre los que puede estar un atributo.
REPORTE	La entidad reporte representa a lo REPORTES de diccionario de datos que puede generar un usuario acerca de los modelos de datos que crea.

Lista de herencias en el Diagrama

Nombre	Descripción	Entidad Padre
HERENCIA_CONSTRAINTS	Hijo de la entidad padre RESTRICCION	RESTRICCION

Entidad ATRIBUTO

Atributo NOMBRE de la entidad ATRIBUTO

Descripción del atributo NOMBRE de la entidad ATRIBUTO

Nombre	NOMBRE
Descripción	Nombre que el usuario asocia al atributo, para su posterior manipulación.
Tipo de Dato	VA50
Mandatorio	Si

Atributo ID_ATRIBUTO de la entidad ATRIBUTO

Descripción del atributo ID_ATRIBUTO de la entidad ATRIBUTO

Nombre	ID_ATRIBUTO
Descripción	Identificador de Referencia del atributo, utilizado internamente por la herramienta, para su posterior recuperación.
Tipo de Dato	I
Mandatorio	Si

Atributo DESCRIPCION_ATRIBUTO de la entidad ATRIBUTO

Descripción del atributo DESCRIPCION_ATRIBUTO de la entidadATRIBUTO

Nombre	DESCRIPCION_ATRIBUTO
Descripción	Contenido opcional que puede agregar el usuario acerca del atributo de una entidad, para la posterior generación de reportes mediante el uso de la herramienta.
Tipo de Dato	VA300
Mandatorio	No

Atributo OPCIONALIDAD_ATRIBUTO de la entidad ATRIBUTO

Descripción del atributo OPCIONALIDAD_ATRIBUTO de la entidad ATRIBUTO

Nombre	OPCIONALIDAD_ATRIBUTO
Descripción	*Determina si el atributo debe ser obligatorio dentro de una tupla, los valores posibles de este atributo son: 0 si el* atributo es opcional y 1 si es mandatario.
Tipo de Dato	A1
Mandatorio	Si

Restricción de Entidad atributo OPCIONALIDAD_ATRIBUTO de Entidad ATRIBUTO

Lista de valores	0 opcional

	1 Mandatario

Atributo TIPO_DE_DATO_REFERENCIA de la entidad ATRIBUTO

Descripción del atributo TIPO_DE_DATO_REFERENCIA de la entidad ATRIBUTO

Nombre	TIPO_DE_DATO_REFERENCIA
Descripción	Representa el formato en el que van a ser almacenados los datos en memoria para la implementación de la base de datos.
Tipo de Dato	VA30
Mandatorio	Si

Atributo LONGITUD de la entidad ATRIBUTO

Descripción del atributo LONGITUD de la entidad ATRIBUTO

Nombre	LONGITUD
Descripción	Espacio en memoria que representara ese atributo al ser implementado dentro de la base de datos.
Tipo de Dato	I
Mandatorio	No

Atributo PRIMARIO de la entidad ATRIBUTO

Descripción del atributo PRIMARIO de la entidad ATRIBUTO

Nombre	PRIMARIO
Descripción	Representa una bandera de referencia que indica si el atributo es primario o no dentro de la entidad, para la posterior generación de restricciones de clave primaria. Puede tomar 2 valores posibles: 0 si no es primario, 1 si es primario.
Tipo de Dato	A1
Mandatario	Si

Restricción de Entidad atributo PRIMARIO de Entidad ATRIBUTO

Lista de valores	0 no primario 1 primario

Atributo UNICO de la entidad ATRIBUTO

Descripción del atributo UNICO de la entidad ATRIBUTO

Nombre	UNICO
Descripción	Representa una bandera de referencia que indica si el atributo es único o no dentro de la entidad, para la posterior generación de restricciones de unicidad e índices. Puede tomar 2 valores posibles: 0 si no es único, 1 si es único.
Tipo de Dato	A1
Mandatario	Si

Restricción de Entidad atributo UNICO de Entidad ATRIBUTO

Lista de valores	0 no 1 único

Lista de identificadores de la Entidad ATRIBUTO

Nombre	Entidad Padre
ID_ATRIBUTO_PI	Entidad 'ATRIBUTO'

Entidad CHECK

Descripción de Entidad CHECK

Nombre	CHECK
Descripción	La Entidad CHECK representa las restricciones de dominio que se aplica sobre columnas de una tabla.

Atributo VALOR_CHECK de la entidad CHECK

Descripción del atributo VALOR_CHECK de la entidad CHECK

Nombre	VALOR_CHECK
Descripción	Lista de valores posibles en los que se verifica la restricción de dominio
Tipo de Dato	VA20
Mandatorio	Si

Entidad CLAVE_FORANEA

Descripción de Entidad CLAVE_FORANEA

Nombre	CLAVE_FORANEA
Descripción	La Entidad CLAVE_FORANEA representa las restricciones de clave foránea que se aplica sobre columnas de una tabla.

Atributo ID_CLAVE_FORANEA de la entidad CLAVE_FORANEA

Descripción del atributo ID_CLAVE_FORANEA de la entidad CLAVE_FORANEA

Nombre	ID_CLAVE_FORANEA
Descripción	Identificador de Referencia de la restricción de clave foránea, utilizado internamente por la herramienta, para su posterior recuperación.
Tipo de Dato	I
Mandatorio	Si

Atributo NOMBRE_CLAVE_FORANEA de la entidad CLAVE_FORANEA

Descripción del atributo NOMBRE_CLAVE_FORANEA de la entidad CLAVE_FORANEA

Nombre	NOMBRE_CLAVE_FORANEA

Descripción	Nombre que se le asocia a las restricciones de clave foránea, para su posterior manipulación.
Tipo de Dato	VA30
Mandatario	Si

Atributo DESCRIPCION_CLAVE_FORANEA de la entidad CLAVE_FORANEA

Descripción del atributo DESCRIPCION_CLAVE_FORANEA de la entidad CLAVE_FORANEA

Nombre	DESCRIPCION_CLAVE_FORANEA
Descripción	Contenido opcional que puede agregar el usuario acerca de las restricciones de clave foránea, para la posterior generación de reportes mediante el uso de la herramienta.
Tipo de Dato	
Mandatario	No

Lista de identificadores de Entidad CLAVE_FORANEA

Nombre	Padre
ID_FORANEA_PI	Entidad 'CLAVE_FORANEA'

Entidad CLAVE_PRIMARIA

Descripción de Entidad CLAVE_PRIMARIA

Nombre	CLAVE_PRIMARIA
Descripción	La Entidad CLAVE_PRIMARIA representa las restricciones de primaria que se aplica sobre columnas de una tabla.

Atributo ID_CLAVE_PRIMARIA de la entidad CLAVE_PRIMARIA

Descripción del atributo ID_CLAVE_PRIMARIA de la entidad CLAVE_PRIMARIA

Nombre	ID_CLAVE_PRIMARIA
Descripción	Identificador de Referencia de las restricciones de clave primaria, utilizado internamente por la herramienta, para su posterior recuperación.
Tipo de Dato	I
Mandatorio	Si

Atributo DESCRIPCION_PK de la entidad CLAVE_PRIMARIA

Descripción del atributo DESCRIPCION_PK de la entidad CLAVE_PRIMARIA

Nombre	DESCRIPCION_PK
Descripción	Contenido opcional que puede agregar el usuario acerca de las restricciones de clave primaria, para la posterior generación de reportes mediante el uso de la herramienta.
Tipo de Dato	VA300
Mandatario	No

Atributo NOMBRE_CLAVE_PRIMARIA de la entidad CLAVE_PRIMARIA

Descripción del atributo NOMBRE_CLAVE_PRIMARIA de la entidad CLAVE_PRIMARIA

Nombre	NOMBRE_CLAVE_PRIMARIA
Descripción	Nombre que se asocia a las restricciones de clave primaria, para su posterior manipulación.
Tipo de Dato	VA30
Mandatorio	Si

Lista de identificadores de la Entidad CLAVE_PRIMARIA

Nombre	Padre
ID_PK_PI	Entidad 'CLAVE_PRIMARIA'

Entidad COLUMNA

Descripción de Entidad COLUMNA

Nombre	COLUMNA
Descripción	La entidad COLUMNA unidad básica de una base de datos. En ella se especifican todo lo referente a los datos que se almacenan dentro de las tablas del modelo lógico de datos.

Atributo NOMBRE_ATRIBUTO_REFERENCIA de la entidad COLUMNA

Descripción del atributo NOMBRE_ATRIBUTO_REFERENCIA de la entidad COLUMNA

Nombre	NOMBRE_ATRIBUTO_REFERENCIA
Descripción	Nombre que el usuario asocia a la columna, para su posterior manipulación.
Tipo de Dato	VA30
Mandatorio	Si

Atributo ID_ATRIBUTO_REFERENCIA de la entidad COLUMNA

Descripción del atributo ID_ATRIBUTO_REFERENCIA de la entidadCOLUMNA

Nombre	ID_ATRIBUTO_REFERENCIA
Descripción	Identificador de Referencia de las columnas, utilizado internamente por la herramienta, para su posterior recuperación.
Tipo de Dato	I
Mandatorio	Si

Atributo OPCIONALIDAD_ATRIBUTO_REFERENCIA de la entidad COLUMNA

Descripción del atributo OPCIONALIDAD_ATRIBUTO_REFERENCIA de la entidad COLUMNA

Nombre	OPCIONALIDAD_ATRIBUTO_REFERENCIA
Descripción	Determina si la columna debe ser obligatorio dentro de una tupla, los valores posibles de este atributo son: 0 si la columna es opcional y 1 si es mandatoria.
Tipo de Dato	A1
Mandatorio	Si

Atributo TIPO_DE_DATO_REFERENCIA de la entidad COLUMNA

Descripción del atributo TIPO_DE_DATO_REFERENCIA de la entidad COLUMNA

Nombre	TIPO_DE_DATO_REFERENCIA
Descripción	Representa el formato en el que van a ser almacenados los datos en memoria para la implementación de la base de datos.
Tipo de Dato	VA30
Mandatorio	Si

Lista de identificadores de la Entidad COLUMNA

Nombre	Padre
ID_ATRIBUTO_REFERENCIA_PI	Entidad 'COLUMNA'

Entidad CONDICIONES

Descripción de Entidad CONDICIONES

Nombre	CONDICIONES
Descripción	Entidad que almacena las condiciones pertenecientes a una consulta realizada por el ususario.

Atributo StringCondiciones de la entidad CONDICIONES

Descripción del atributo StringCondiciones de la entidad CONDICIONES

Nombre	StringCondiciones
Descripción	Cadena de Caracteres que almacena las condiciones dentro de una consulta.
Tipo de Dato	VA30
Mandatorio	Si

Entidad CONSULTA

Descripción de Entidad CONSULTA

Nombre	CONSULTA
Descripción	La entidad CONSULTA representa las preguntas que se formularan a la a la base de datos con el fin de extraer y presentar la información resultante.

Atributo ID_CONSULTA de la entidad CONSULTA

Descripción del atributo ID_CONSULTA de la entidad CONSULTA

Nombre	ID_CONSULTA
Descripción	Identificador de Referencia de la consulta, utilizado internamente por la herramienta, para su posterior recuperación.
Tipo de Dato	I
Mandatorio	Si

Atributo NOMBRE_CONSULTA de la entidad CONSULTA

Descripción del atributo NOMBRE_CONSULTA de la entidad CONSULTA

Nombre	NOMBRE_CONSULTA
Descripción	Nombre de referencia que el usuario otorga a la consuta generada.
Tipo de Dato	VA30
Mandatorio	Si

Lista de identificadores de la Entidad CONSULTA

Nombre	Padre
ID_CONSULTA_PI	Entidad 'CONSULTA'

Entidad DATOS_USUARIOS

Descripción de Entidad DATOS_USUARIOS

Nombre	DATOS_USUARIOS
Descripción	

Entidad DOMINIO

Descripción de Entidad DOMENIO

Nombre	DOMINIO
Descripción	Representa conjunto de valores de un mismo tipo entre los que puede estar un atributo.

Atributo NOMBRE_DOMINIO de la entidad DOMENIO

Descripción del atributo NOMBRE_DOMINIO de la entidad DOMENIO.

Nombre	NOMBRE_DOMENIO
Descripción	Nombre que el usuario asocia a al dominio, para su posterior manipulación.
Tipo de Dato	VA50
Mandatorio	Si

Atributo ID_DOMENIO de la entidad DOMENIO

Descripción del atributo ID_DOMENIO de la entidad DOMENIO.

Nombre	ID_DOMENIO
Descripción	Identificador de Referencia del dominio, utilizado internamente por la herramienta, para su posterior recuperación.
Tipo de Dato	I
Mandatorio	Si

Atributo LISTA_VALORES de la entidad DOMINIO

Descripción del atributo LISTA_VALORES de la entidad DOMINIO

Nombre	LISTA_VALORES
Descripción	La Lista de valores posibles entre los que se van a aplicar validaciones a atributos que referencia a dicho dominio.
Tipo de Dato	VA100
Mandatorio	Si

Lista de identificadores de la Entidad DOMINIO

Nombre	Padre
ID_DOMENIO_PI	Entidad 'DOMENIO'

Entidad ENTIDAD

Descripción de Entidad ENTIDAD

Nombre	ENTIDAD
Descripción	Una entidad es un objeto concreto o abstracto que presenta interés para el sistema y sobre el que se recoge información la cual va a ser representada en un sistema de base de datos. La mayoría de las entidades modelan objetos o eventos del mundo real, por ejemplo, clientes, productos o llamadas de pedidos.

Atributo NOMBRE_ENTIDAD de la entidad ENTIDAD

Descripción del atributo NOMBRE_ENTIDAD de la entidad ENTIDAD

Nombre	NOMBRE_ENTIDAD
Descripción	Nombre que el usuario asocia a la entidad dentro del modelo conceptual, para su posterior manipulación.
Tipo de Dato	VA50
Mandatorio	Si

Atributo ID_ENTIDAD de la entidad ENTIDAD

Descripción del atributo ID_ENTIDAD de la entidad ENTIDAD

Nombre	ID_ENTIDAD
Descripción	Identificador de Referencia de la Entidad del Modelo Conceptual, utilizado internamente por la herramienta, para su posterior recuperación.
Tipo de Dato	I
Mandatorio	Si

Atributo *DESCRIPCION_ENTIDAD* de la entidad *ENTIDAD*

Descripción del atributo DESCRIPCION_ENTIDAD de la entidad ENTIDAD

Nombre	DESCRIPCION_ENTIDAD
Descripción	Contenido opcional que puede agregar el usuario acerca de una entidad especifica dentro de un modelo conceptual, para la posterior generación de reportes mediante el uso de la herramienta.

Tipo de Dato	VA300
Mandatorio	No

Lista de identificadores de la Entidad ENTIDAD

Nombre	Padre
ID_ENTIDAD_PI	Entidad 'ENTIDAD'

Entidad ESPACIO_TRABAJO

Descripción de Entidad ESPACIO_TRABAJO

Nombre	ESPACIO_TRABAJO
Descripción	Entidad que almacena los datos correspondientes al espacio de trabajo de un usuario.

Atributo ID_WS de la entidad ESPACIO_TRABAJO

Descripción del atributo ID_WS de la entidad ESPACIO_TRABAJO

Nombre	ID_WS
Descripción	Identificador de referencia del espacio de trabajo del usuario.
Tipo de Dato	VA30
Mandatorio	Si

Atributo NOMBRE_WS de la entidad ESPACIO_TRABAJO

Descripción del atributo NOMBRE_WS de la entidad ESPACIO_TRABAJO

Nombre	NOMBRE_WS
Descripción	Nombre asignado al espacio de trabajo de un Usuario especifico.
Tipo de Dato	VA30
Mandatorio	Si

Lista de identificadores de la Entidad ESPACIO_TRABAJO

Nombre	Padre
Identifier_1	Entidad 'ESPACIO_TRABAJO'

Entidad FICHERO

Descripción de Entidad FICHERO

Nombre	FICHERO
Descripción	Entidad que almacena los datos acerca de la ruta fisica de los modelos y archivos de configuracion pertenecientes a un espacio de trabajo en especifico

Atributo ID_FICHERO de la entidad FICHERO

Descripción del atributo ID_FICHERO de la entidad FICHERO

Nombre	ID_FICHERO
Descripción	Identificador de referencia de la carpeta física sobre la cual se almacenan los archivos
Tipo de Dato	VA30
Mandatorio	Si

Lista de identificadores de la Entidad FICHERO

Nombre	Padre
Identifier_1	Entidad 'FICHERO'

Entidad MODELO

Descripción de Entidad MODELO

Nombre	MODELO
Descripción	La entidad modelo al ser instanciada representa un modelo de datos, creado por el usuario, bien sea en conceptual o relacional.

Atributo NOMBRE_MODELO de la entidad MODELO

Descripción del atributo NOMBRE_MODELO de la entidad MODELO

Nombre	NOMBRE_MODELO
Descripción	Nombre que el usuario asocia a los modelos, para su posterior manipulación.
Tipo de Dato	VA50
Mandatorio	Si

Atributo ID_MODELO de la entidad MODELO

Descripción del atributo ID_MODELO de la entidad MODELO

Nombre	ID_MODELO
Descripción	Identificador de Referencia de los modelos de datos, utilizado internamente por la herramienta, para su posterior recuperación.
Tipo de Dato	I
Mandatorio	Si

Atributo DESCRIPCION_MODELO de la entidad MODELO

Descripción del atributo DESCRIPCION_MODELO de la entidad MODELO

Nombre	DESCRIPCION_MODELO

Descripción	Contenido opcional que puede agregar el usuario acerca de los modelos de datos, para la posterior generación de reportes mediante el uso de la herramienta.
Tipo de Dato	VA300
Mandatorio	Si

Atributo TIPO_MODELO de la entidad MODELO

Descripción del atributo TIPO_MODELO de la entidad MODELO

Nombre	TIPO_MODELO
Descripción	Representa al tipo de modelo que el usuario desee almacenar, bien sea conceptual o lógico. Esta regido por un dominio mediante el cual chequea los valores de este atributo entre los valores 'con" (modelo conceptual) y "log" (lógico)
Tipo de Dato	A3
Mandatorio	Si

Restricción de Entidad atributo TIPO_MODELO de la Entidad MODELO

Lista de valores	Con Conceptual Log Lógico

Lista de identificadores de la Entidad MODELO

Nombre	Padre

ID_MODELO_PI	Entidad 'MODELO'

Entidad PRIVILEGIO

Descripción de Entidad PRIVILEGIO

Nombre	PRIVILEGIO
Descripción	La entidad PRIVILEGIO representa el o los permisos para realizar una acción, asignable a un usuario o un rol

Atributo ID_PRIVILEGIO de la entidad PRIVILEGIO

Descripción del atributo ID_PRIVILEGIO de la entidad PRIVILEGIO

Nombre	ID_PRIVILEGIO
Descripción	Identificador de Referencia de los Privilegios, utilizado internamente por la herramienta, para su posterior recuperación.
Tipo de Dato	I
Mandatorio	Si

Atributo NOMBRE_PRIVILEGIO de la entidad PRIVILEGIO

Descripción del atributo NOMBRE_PRIVILEGIO de la entidad PRIVILEGIO

Nombre	NOMBRE_PRIVILEGIO
Descripción	Nombre Predefinido asociado a los privilegios, para su posterior manipulación.

Tipo de Dato	VA50
Mandatorio	Si

Atributo DESCRIPCION_PRIVILEGIO de la entidad PRIVILEGIO

Descripción del atributo DESCRIPCION_PRIVILEGIO de la entidad PRIVILEGIO

Nombre	DESCRIPCION_PRIVILEGIO
Descripción	Contenido opcional que puede agregar el usuario acerca de los privilegios, para la posterior generación de reportes mediante el uso de la herramienta.
Tipo de Dato	VA300
Mandatorio	No

Lista de identificadores de la Entidad PRIVILEGIO

Nombre	Padre
ID_PRIVILEGIO_PI	Entidad 'PRIVILEGIO'

Entidad RELACION

Descripción de Entidad RELACION

Nombre	RELACION

Descripción	Objeto del modelo conceptual, que representa la asociación entre entidades. Mediante estas se analiza n sus componentes para la posterior traducción al modelo relacional de datos.

Atributo *NOMBRE_RELACION* de la entidad *RELACION*

Descripción del atributo NOMBRE_RELACION de la entidad RELACION

Nombre	NOMBRE_RELACION
Descripción	Nombre que el usuario asocia a la Relación dentro del modelo conceptual, para su posterior manipulación.
Tipo de Dato	VA50
Mandatorio	*Si*

Atributo ID_RELACION de la entidad RELACION

Descripción del atributo ID_RELACION de la entidadRELACION

Nombre	ID_RELACION
Descripción	Identificador de Referencia de la Relación, utilizado internamente por la herramienta, para su posterior recuperación.
Tipo de Dato	I
Mandatorio	Si

Atributo DESCRIPCION_RELACION de la entidad RELACION

Descripción del atributo *DESCRIPCION_RELACION de la entidad RELACION*

Nombre	DESCRIPCION_RELACION
Descripción	Contenido opcional que puede agregar el usuario acerca de la Relación entre 2 entidades de un modelo conceptual de datos, para la posterior generación de reportes mediante el uso de la herramienta.
Tipo de Dato	VA300
Mandatorio	No

Atributo CARDENALIDAD de la entidad RELACION

Descripción del atributo CARDENALIDAD de la entidad RELACION

Nombre	CARDENALIDAD
Descripción	Representa Es el número de tuplas que posee una relacionen un momento determinado.
Tipo de Dato	VA5
Mandatorio	Si

Restricción de Entidad atributo CARDENALIDAD de la Entidad RELACION

Lista de valores	0,n cero-muchos 0,1 cero-uno 1,1 uno-uno 1,n uno-muchos n,n muchos-muchos

Atributo OPCIONALIDAD_RELACION de la entidad RELACION

Descripción del atributo OPCIONALIDAD_RELACION de la entidad RELACION

Nombre	OPCIONALIDAD_RELACION
Descripción	Permiten definir con mayor claridad aquellos casos en los que una relación puede no establecerse. Las especificaciones de estas situaciones nos permitirá definir estructuras más precisas, consistentes y de baja redundancia.
Tipo de Dato	VA3
Mandatorio	Si

Restricción de la Entidad atributo OPCIONALIDAD_RELACION de la Entidad RELACION

Lista de valores	*0 mandatoria* 1 opcional

Lista de identificadores de la Entidad RELACION

Nombre	Padre
ID_RELACION_PI	*Entidad 'RELACION'*

Entidad REPORTE

Descripción de Entidad *REPORTE*

Nombre	REPORTE
Descripción	La entidad reporte representa a lo REPORTES de diccionario de datos que puede generar un usuario acerca de los modelos de datos que crea.

Atributo ID_REPORTE de la entidad REPORTE

Descripción del atributo ID_REPORTE de la entidadREPORTE

Nombre	ID_REPORTE
Descripción	Identificador de Referencia de los Reportes, utilizado internamente por la herramienta, para su posterior recuperación.
Tipo de Dato	VA30
Mandatorio	Si

Atributo NOMBRE_REPORTE de la entidad REPORTE

Descripción del atributo NOMBRE_REPORTE de la entidad REPORTE

Nombre	NOMBRE_REPORTE
Descripción	Nombre que el usuario asocia a los reportes, para su posterior manipulación.
Tipo de Dato	VA30
Mandatorio	Si

Atributo DESCRIPCION_REPORTE de la entidad REPORTE

Descripción del atributo DESCRIPCION_REPORTE de la entidad REPORTE

Nombre	DESCRIPCION_REPORTE
Descripción	Contenido opcional que puede agregar el usuario acerca de los reportes, para la posterior generación de reportes mediante el uso de la herramienta.
Tipo de Dato	VA300
Mandatorio	No

Lista de identificadores de la Entidad REPORTE

Nombre	Padre
PI_REPORTE	Entidad 'REPORTE'

Entidad RESTRICCION

Descripción de Entidad RESTRICCION

Nombre	RESTRICCION
Descripción	Constituyen el mecanismo primordial para especificar ciertas reglas que deben cumplir los datos almacenados en la base de datos. Esta relacionada con aquellas restricciones de integridad derivadas del esquema relacional Puede ser utilizada para definir restricciones de integridad de identidad, restricciones de integridad referencial, restricciones de dominio en las columnas de una tabla o inclusive ciertas restricciones explícitas.

Atributo ID_RESTRICCION de la entidad RESTRICCION

Descripción del atributo ID_RESTRICCION de la entidad RESTRICCION

Nombre	ID_RESTRICCION
Descripción	Identificador de Referencia de las restricciones, utilizado internamente por la herramienta, para su posterior recuperación.
Tipo de Dato	I
Mandatorio	Si

Atributo NOMBRE_RESTRICCION de la entidad RESTRICCION

Descripción del atributo NOMBRE_RESTRICCION de la entidad RESTRICCION

Nombre	NOMBRE_RESTRICCION
Descripción	Nombre que el usuario asocia a las restricciones, para su posterior manipulación.
Tipo de Dato	VA50
Mandatorio	Si

Lista de identificadores de la Entidad RESTRICCION

Nombre	Padre
ID_RESTRICCION_PI	Entidad 'RESTRICCION'

Entidad ROL

Descripción de Entidad ROL

Nombre	ROL
Descripción	La entidad ROL representa un conjunto de privilegios, asignables a un usuario o un rol.

Atributo ID_ROL de la entidad ROL

Descripción del atributo ID_ROL de la entidad ROL

Nombre	ID_ROL
Descripción	Identificador de Referencia de los Roles, utilizado internamente por la herramienta, para su posterior recuperación.
Tipo de Dato	I
Mandatorio	Si

Atributo NOMBRE_ROL de la entidad ROL

Descripción del atributo NOMBRE_ROL de la entidad ROL

Nombre	NOMBRE_ROL
Descripción	Nombre que el usuario asocia a los Roles, para su posterior manipulación.
Tipo de Dato	VA50
Mandatorio	Si

Atributo DESCRIPCION_ROL de la entidad ROL

Descripción del atributo DESCRIPCION_ROL de la entidad ROL

Nombre	DESCRIPCION_ROL
Descripción	Contenido opcional que puede agregar el usuario acerca de los Roles, para la posterior generación de reportes mediante el uso de la herramienta.
Tipo de Dato	VA300
Mandatorio	No

Lista de identificadores de la Entidad ROL

Nombre	Padre
ID_ROL_PI	Entidad 'ROL'

Entidad ROL_PRIVILEGIO

Descripción de Entidad ROL_PRIVILEGIO

Nombre	ROL_PRIVILEGIO
Descripción	Entidad de Intersección entre las entidades ROL y PRIVILEGIO

Entidad TABLA

Descripción de Entidad TABLA

Nombre	TABLA
Descripción	La entidad TABLA representa la unidad donde se crean el conjunto de datos de una base de datos. Estos datos estarán ordenados en columnas verticales. Estos definidos mediante campos y sus características.

Atributo NOMBRE_TABLA de la entidad TABLA

Descripción del atributo NOMBRE_TABLA de la entidad TABLA

Nombre	NOMBRE_TABLA
Descripción	Nombre que el usuario asocia a las tablas, para su posterior manipulación.
Tipo de Dato	VA30
Mandatorio	Si

Atributo ID_TABLA de la entidad TABLA

Descripción del atributo ID_TABLA de la entidad TABLA

Nombre	ID_TABLA
Descripción	Identificador de Referencia de las tablas, utilizado internamente por la herramienta, para su posterior recuperación.
Tipo de Dato	I
Mandatorio	Si

Atributo DESCRIPCION_TABLA de la entidad TABLA

Descripción del atributo DESCRIPCION_TABLA de la entidad TABLA

Nombre	TABLA
Descripción	La entidad TABLA representa la unidad donde se crean el conjunto de datos de una base de datos. Estos datos estarán ordenados en columnas verticales. Estos definidos mediante campos y sus características.

Atributo NOMBRE_TABLA de la entidad TABLA

Descripción del atributo NOMBRE_TABLA de la entidad TABLA

Nombre	NOMBRE_TABLA
Descripción	Nombre que el usuario asocia a las tablas, para su posterior manipulación.
Tipo de Dato	VA30
Mandatorio	Si

Atributo ID_TABLA de la entidad TABLA

Descripción del atributo ID_TABLA de la entidad TABLA

Nombre	ID_TABLA
Descripción	Identificador de Referencia de las tablas, utilizado internamente por la herramienta, para su posterior recuperación.
Tipo de Dato	I
Mandatorio	Si

Atributo DESCRIPCION_TABLA de la entidad TABLA

Descripción del atributo DESCRIPCION_TABLA de la entidad TABLA

Nombre	DESCRIPCION_TABLA
Descripción	Contenido opcional que puede agregar el usuario acerca de las tablas, para la posterior generación de reportes mediante el uso de la herramienta.
Tipo de Dato	VA300
Mandatorio	No

Lista de identificadores de la Entidad TABLA

Nombre	Padre
ID_TABLA_PI	Entidad 'TABLA'

Entidad UNIDAD_ALMACENAMIENTO

Descripción de Entidad UNIDAD_ALMACENAMIENTO

Nombre	UNIDAD_ALMACENAMIENTO
Descripción	La entidad UNIDAD_ALMACENAMIENTO representa el espacio físico en el que va a ser instalado la herramienta.

Atributo ID_UNIDAD_ALMACENAMIENTO de la entidad UNIDAD_ALMACENAMIENTO

Descripción del atributo ID_UNIDAD_ALMACENAMIENTO de la entidad UNIDAD_ALMACENAMIENTO

Nombre	ID_UNIDAD_ALMACENAMIENTO

Descripción	Identificador de Referencia de la unidad de almacenamiento.
Tipo de Dato	VA30
Mandatorio	Si

Atributo NOMBRE_UNIDAD_ALMACENAMIENTO de la entidad UNIDAD_ALMACENAMIENTO

Descripción del atributo NOMBRE_UNIDAD_ALMACENAMIENTO de la entidad UNIDAD_ALMACENAMIENTO

Nombre	NOMBRE_UNIDAD_ALMACENAMIENTO
Descripción	Nombre que el usuario asocia a la unidad de almacenamiento.
Tipo de Dato	VA30
Mandatorio	Si

Atributo DESCRIPCION_UNIDAD_ALMACENAMIENTO de la entidad UNIDAD_ALMACENAMIENTO

Descripción del atributo DESCRIPCION_UNIDAD_ALMACENAMIENTO de la entidad UNIDAD_ALMACENAMIENTO

Nombre	DESCRIPCION_UNIDAD_ALMACENAMIENTO
Descripción	Contenido opcional que se agrega acerca de la unidad de almacenamiento en el que va a ser instalada la herramienta.
Tipo de Dato	VA300
Mandatorio	No

Lista de identificadores de la Entidad UNIDAD_ALMACENAMIENTO

Nombre	Padre
Identifier_1	Entidad 'UNIDAD_ALMACENAMIENTO'

Entidad UNIQUE

Descripción de Entidad UNIQUE

Nombre	UNIQUE
Descripción	La Entidad UNIQUE representa las restricciones de Unicidad que se aplica sobre columnas de una tabla.

Entidad USUARIO

Descripción de Entidad USUARIO

Nombre	USUARIO
Descripción	La entidad USUARIO representa tanto a aquellos usuarios propietarios de modelos de datos dentro de la herramienta, o una colección de objetos y privilegios identificado con un nombre y password

Atributo LOGIN_USUARIO de la entidad USUARIO

Descripción del atributo LOGIN_USUARIO de la entidad USUARIO

Nombre	LOGIN_USUARIO
Descripción	Nombre de usuario según el cual será identificado el usuario dentro de la herramienta, o dentro de la base de datos que

	se implantara posteriormente mediante el modelo.
Tipo de Dato	VA50
Mandatorio	Si

Atributo PASSWORD_USUARIO de la entidad USUARIO

Descripción del atributo PASSWORD_USUARIO de la entidad USUARIO

Nombre	PASSWORD_USUARIO
Descripción	contraseña tanto de acceso a la herramienta, como la que será asignada a los usuarios creados para la base de datos a crear
Tipo de Dato	VA50
Mandatorio	Si

Atributo ID_USUARIO de la entidad USUARIO

Descripción del atributo ID_USUARIO de la entidad USUARIO

Nombre	ID_USUARIO
Descripción	Identificador de Referencia de los usuarios, utilizado internamente por la herramienta, para su posterior recuperación.
Tipo de Dato	VA50
Mandatorio	Si

Lista de identificadores de la Entidad USUARIO

Nombre	Padre
ID_USUARIO_PI	Entidad 'USUARIO'

Entidad USUARIO_APLICACION

Descripción de Entidad USUARIO_APLICACION

Nombre	USUARIO_APLICACION
Descripción	Entidad que almacena datos acerca de los usuarios registrados en la aplicacion asi como tambien su informacion personal.

Atributo ID_USUARIO de la entidad USUARIO_APLICACION

Descripción del atributo ID_USUARIO de la entidad USUARIO_APLICACION

Nombre	ID_USUARIO
Descripción	Identificadorde referencia del usuario que hace login en la aplicacion para crear su espacio de trabajo y crear modelos de datos.
Tipo de Dato	VA30
Mandatorio	Si

Atributo LOGIN_USUARIO de la entidad USUARIO_APLICACION

Descripción del atributo LOGEN_USUARIO de la entidad USUARIO_APLICACION

Nombre	LOGIN_USUARIO
Descripción	Nombre utilizado por el usuario para acceder a su espacio de trabajo.
Tipo de Dato	VA30
Mandatorio	Si

Atributo PASS_USUARIO de la entidad USUARIO_APLICACION

Descripción del atributo PASS_USUARIO de la entidad USUARIO_APLICACION

Nombre	PASS_USUARIO
Descripción	Contrasena utilizada por el usuario para acceder a su espacio de trabajo.
Tipo de Dato	VA30
Mandatorio	Si

Lista de identificadores de la Entidad USUARIO_APLICACION

Nombre	Padre
Identifier_1	Entidad 'USUARIO_APLICACION'

Entidad USUARIO_PRIVILEGIO

Descripción de Entidad USUARIO_PRIVILEGIO

Nombre	USUARIO_PRIVILEGIO
Descripción	Entidad de Intersección entre las entidades USUARIO y PRIVILEGIO

Entidad USUARIO_ROL

Descripción de Entidad USUARIO_ROL

Nombre	USUARIO_ROL
Descripción	Entidad de Intersección entre las entidades USUARIO y ROL

Entidad VERSION

Descripción de Entidad VERSION

Nombre	VERSION
Descripción	La entidad Versión representa aquellas versiones que los usuarios pueden realizar de sus modelos de datos, para su posterior manipulación.

Atributo ID_VERSION de la entidad VERSION

Descripción del atributo ID_VERSION de la entidad VERSION

Nombre	ID_VERSION
Descripción	Identificador de Referencia de la Versión, utilizado internamente por la herramienta, para su posterior recuperación.
Tipo de Dato	I
Mandatorio	Si

Atributo NOMBRE_VERSION de la entidad VERSION

Descripción del atributo NOMBRE_VERSION de la entidad VERSION

Nombre	NOMBRE_VERSION
Descripción	Nombre que el usuario asocia a la versión, para su posterior manipulación.
Tipo de Dato	VA50
Mandatorio	Si

Atributo DESCRIPCION_VERSION de la entidad VERSION

Descripción del atributo DESCRIPCION_VERSION de la entidad VERSION

Nombre	DESCRIPCION_VERSION

Descripción	Contenido opcional que puede agregar el usuario acerca de la versión de un modelo, para la posterior generación de reportes mediante el uso de la herramienta.
Tipo de Dato	VA300
Mandatorio	No

Atributo FECHA_CREACION de la entidad VERSION

Descripción del atributo FECHA_CREACION de la entidad VERSION

Nombre	FECHA_CREACION
Descripción	Fecha en la que fue creada la versión de un modelo de datos, por parte del usuario de la herramienta.
Tipo de Dato	TS
Mandatorio	Si

Atributo ULTIMA_MODIFICACION de la entidad VERSION

Descripción del atributo ULTIMA_MODIFICACION de la entidad VERSION

Nombre	ULTIMA_MODIFICACION
Descripción	Atributo de control que utiliza la herramienta para poder recuperar cuando el usuario lo requiera, la versión de un modelo que haya abierto o creado recientemente.
Tipo de Dato	TS
Mandatorio	Si

Lista de identificadores de la Entidad VERSION

Nombre	Padre
ID_VERSION_PI	Entidad 'VERSION'

Relación ATT_REF_CONSTRAENT

Descripción de la relación ATT_REF_CONSTRAENT

Nombre	ATT_REF_CONSTRAENT
Descripción	Relación que se establece entre las entidades COLUMNA y RESTRICCION, a una columna se le pueden aplicar mas de una restricción.
Entidad 1	COLUMNA
Entidad 2	RESTRICCION
Entidad 1 -> Entidad 2 cardinalidad	1,n
Entidad 2 -> Entidad 1 cardinalidad	1,1

Relacion ATT_REF_FK

Descripción de la relación ATT_REF_FK

Nombre	ATT_REF_FK
Descripción	Relación que se establece entre las entidades COLUMNA y CLAVE_FORANEA, a que una columna puede estar en

	más de una clave foránea.
Entidad 1	COLUMNA
Entidad 2	CLAVE_FORANEA
Entidad 1 -> Entidad 2 cardinalidad	0,n
Entidad 2 -> Entidad 1 cardinalidad	0,1

Relacion A_ENTIDAD

Descripción de la relación A_ENTIDAD

Nombre	A_ENTIDAD
Descripción	Relación que se establece entre las entidades Entidad y Relación y representa que una entidad se asocia con otra mediante una RELACION.
Entidad 1	ENTIDAD
Entidad 2	RELACION
Entidad 1 -> Entidad 2 cardinalidad	0,n
Entidad 2 -> Entidad 1 cardinalidad	1,1

Relación CONTIENE

Descripción de la relación CONTIENE

Nombre	CONTIENE

Descripción	
Entidad 1	FICHERO
Entidad 2	FICHERO
Entidad 1 -> Entidad 2 cardinalidad	0,n
Entidad 2 -> Entidad 1 cardinalidad	0,1

Relación CONTIENE

Descripción de la relación CONTIENE

Nombre	CONTIENE
Descripción	Relación que se establece entre las entidades Rol y Usuario_Rol, representa una relación hacia la intersección entre las tablas USUARIO y ROL. Un usuario puede tener muchos roles asignados y un ROL puede tener muchos usuarios asociados
Entidad 1	ROL
Entidad 2	USUARIO_ROL
Entidad 1 -> Entidad 2 cardinalidad	1,n
Entidad 2 -> Entidad 1 cardinalidad	1,1

Relación CONTIENE

Descripción de la relación CONTIENE

Nombre	CONTIENE
Descripción	Relación que se establece entre las entidades CONSULTA y TABLA, representa que una consulta puede asociar a muchas tablas de la base de datos.
Entidad 1	CONSULTA
Entidad 2	TABLA
Entidad 1 -> Entidad 2 cardinalidad	0,1
Entidad 2 -> Entidad 1 cardinalidad	0,n

Relación CONTIENE

Descripción de la relación CONTIENE

Nombre	CONTIENE
Descripción	Relación que se establece entre las entidades Modelo y Dominio y representa que en un modelo pueden declararse uno o muchos dominios de verificación de valores.
Entidad 1	MODELO
Entidad 2	DOMENIO
Entidad 1 -> Entidad 2 cardinalidad	0,n
Entidad 2 -> Entidad 1 cardinalidad	0,1

Relación CONTIENE

Descripción de la relación CONTIENE

Nombre	CONTIENE
Descripción	
Entidad 1	FICHERO
Entidad 2	ARCHIVOS
Entidad 1 -> Entidad 2 cardinalidad	0,n
Entidad 2 -> Entidad 1 cardinalidad	0,1

Relación DESARROLLA

Descripción de la relación DESARROLLA

Nombre	DESARROLLA
Descripción	
Entidad 1	USUARIO_APLICACION
Entidad 2	ESPACIO_TRABAJO
Entidad 1 -> Entidad 2 cardinalidad	0,n
Entidad 2 -> Entidad 1 cardinalidad	0,1

Relación DESDE_ENTIDAD

Descripción de la relación DESDE_ENTIDAD

Nombre	DESDE_ENTIDAD

Descripción	Relación que se establece entre las entidades Entidad y Relación y representa que una entidad se asocia con otra mediante una RELACION.
Entidad 1	ENTIDAD
Entidad 2	RELACION
Entidad 1 -> Entidad 2 cardinalidad	0,n
Entidad 2 -> Entidad 1 cardinalidads	1,1

Relación ESTA_COMPUESTO

Descripción de la relación ESTA_COMPUESTO

Nombre	ESTA_COMPUESTO
Descripción	Relación que se establece entre las entidades Usuario y Usuario_Privilegio, representa una relación hacia la intersección entre las tablas PRIVILEGIO y ROL_PRIVILEGIO. Un ROL puede tener muchos privilegios asignados y un PRIVILEGIO puede tener muchos roles asociados
Entidad 1	PRIVILEGIO
Entidad 2	ROL_PRIVILEGIO
Entidad 1 -> Entidad 2 cardinalidad	1,n
Entidad 2 -> Entidad 1 cardinalidad	1,1

Relacion ES_ASIGNADO

Descripción de la relación ES_ASIGNADO

Nombre	ES_ASIGNADO
Descripción	Relación que se establece entre las entidades Privilegio y Usuario_Privilegio, representa una relación hacia la intersección entre las tablas USUARIO y PRIVILEGIO. Un usuario puede tener muchos privilegios asignados y un PRIVILEGIO puede tener muchos usuarios asociados
Entidad 1	PRIVILEGIO
Entidad 2	USUARIO_PRIVILEGIO
Entidad 1 -> Entidad 2 cardinalidad	1,n
Entidad 2 -> Entidad 1 cardinalidad	1,1

Relación GENERA

Descripción de la relación GENERA

Nombre	GENERA
Descripción	Relación que se establece entre las entidades Usuario y Reporte, representa que un usuario puede generar muchos reportes.
Entidad 1	USUARIO
Entidad 2	REPORTE
Entidad 1 -> Entidad 2 cardinalidad	0,n
Entidad 2 -> Entidad 1 cardinalidad	0,1

Relación POSEE

Descripción de la relación POSEE

Nombre	POSEE
Descripción	Relación entre las entidades Modelo y versión que establece que un modelo puede estar compuesto por una o mas versiones
Entidad 1	VERSION
Entidad 2	MODELO
Entidad 1 -> Entidad 2 cardinalidad	1,1
Entidad 2 -> Entidad 1 cardinalidad	1,n

Relación REF_MODELO_TABLA

Descripción de la relación REF_MODELO_TABLA

Nombre	REF_MODELO_TABLA
Descripción	Relación que se establece entre las entidades Modelo y Tabla , representa que en un Modelo lógico de datos tienen que existir una o mas tablas.
Entidad 1	MODELO
Entidad 2	TABLA
Entidad 1 -> Entidad 2 cardinalidad	1,n
Entidad 2 -> Entidad 1 cardinalidad	1,1

Relación SE COMPONE

Descripción de la relación SE COMPONE

Nombre	SE COMPONE
Descripción	Relación que se establece entre las entidades Usuario y Usuario_Privilegio, representa una relación hacia la intersección entre las tablas ROL y ROL_PRIVILEGIO. Un ROL puede tener muchos privilegios asignados y un PRIVILEGIO puede tener muchos roles asociados
Entidad 1	ROL
Entidad 2	ROL_PRIVILEGIO
Entidad 1 -> Entidad 2 cardinalidad	1,n
Entidad 2 -> Entidad 1 cardinalidad	1,1

Relación SE COMPONE

Descripción de la relación SE COMPONE

Nombre	SE COMPONE
Descripción	Relación que se establece entre las entidades Modelo y Entidad, representa que en un modelo conceptual de datos tienen que existir una o más entidades.
Entidad 1	MODELO
Entidad 2	ENTIDAD
Entidad 1 -> Entidad 2 cardinalidad	1,n

Entidad 2 -> Entidad 1 cardinalidad	1,1

Relación SE_ASIGNA

Descripción de la relación SE_ASIGNA

Nombre	SE_ASIGNA
Descripción	Relación que se establece entre las entidades Usuario y Usuario_Privilegio, representa una relación hacia la intersección entre las tablas USUARIO y PRIVILEGIO. Un usuario puede tener muchos privilegios asignados y un PRIVILEGIO puede tener muchos usuarios asociados.
Entidad 1	USUARIO
Entidad 2	USUARIO_PRIVILEGIO
Entidad 1 -> Entidad 2 cardinalidad	1,n
Entidad 2 -> Entidad 1 cardinalidad	1,1

Relación SE_COMPONE

Descripción de la relación SE_COMPONE

Nombre	SE_COMPONE
Descripción	
Entidad 1	ESPACIO_TRABAJO
Entidad 2	ARCHIVOS

Entidad 1 -> Entidad 2 cardinalidad	0,n
Entidad 2 -> Entidad 1 cardinalidad	0,1

Relación SE_COMPONE_

Descripción de la relación SE_COMPONE_

Nombre	SE_COMPONE_
Descripción	Relación que se establece entre las entidades TABLA y COLUMNA, representa que una TABLA tiene que estar conformada por una o más columnas.
Entidad 1	TABLA
Entidad 2	COLUMNA
Entidad 1 -> Entidad 2 cardinalidad	1,n
Entidad 2 -> Entidad 1 cardinalidad	1,1

Relación SE_CREAN

Descripción de la relación SE_CREAN

Nombre	SE_CREAN
Descripción	Relación que se establece entre las entidades versión y consulta, representa que en una versión pueden existir o no una o mas consultas.
Entidad 1	VERSION

Entidad 2	CONSULTA
Entidad 1 -> Entidad 2 cardinalidad	0,n
Entidad 2 -> Entidad 1 cardinalidad	0,1

Relación SE_ESTRUCTURA

Descripción de la relación SE_ESTRUCTURA

Nombre	SE_ESTRUCTURA
Descripción	Relación que se establece entre las entidades UNIDAD_ALMACENAMIENTO y FICHERO, representa que el sistema de archivos puede estar compuesto por carpetas en las que se almacenaran los archivos.
Entidad 1	UNIDAD_ALMACENAMIENTO
Entidad 2	FICHERO
Entidad 1 -> Entidad 2 cardinalidad	0,n
Entidad 2 -> Entidad 1 cardinalidad	0,1

Relación SE_FORMA

Descripción de la relación SE_FORMA

Nombre	SE_FORMA
Descripción	Relación que se establece entre las entidades CONSULTA y CONDICIONES, representa que una consulta puede estar limitada por muchas condiciones
Entidad 1	CONSULTA

Entidad 2	CONDICIONES
Entidad 1 -> Entidad 2 cardinalidad	0,n
Entidad 2 -> Entidad 1 cardinalidad	0,1

Relación Supertipo

Descripción de la relación Supertipo

Nombre	Supertipo
Descripción	Relación que se establece recursivamente en la entidad "ENTIDAD" con el fin de modelar los supertipos.
Entidad 1	ENTIDAD
Entidad 2	ENTIDAD
Entidad 1 -> Entidad 2 cardinalidad	0,n
Entidad 2 -> Entidad 1 cardinalidad	0,1

Relación TABLA_REF_CONSTRAENT

Descripción de la relación TABLA_REF_CONSTRAENT

Nombre	TABLA_REF_CONSTRAENT
Descripción	Relación que se establece entre las entidades TABLA y RESTRICCION, representa que sobre una tabla están aplicadas una o más restricciones de clave primaria, foránea, unicidad o dominio.
Entidad 1	TABLA

Entidad 2	RESTRICCION
Entidad 1 -> Entidad 2 cardinalidad	1,n
Entidad 2 -> Entidad 1 cardinalidad	1,1

Relación TIENE

Descripción de la relación TIENE

Nombre	TIENE
Descripción	Relación que se establece entre las entidades Entidad y Usuario_Rol, representa una relación hacia la intersección entre las tablas USUARIO y ROL. Un usuario puede tener muchos roles asignados y un Rol puede tener muchos usuarios asociados.
Entidad 1	USUARIO
Entidad 2	USUARIO_ROL
Entidad 1 -> Entidad 2 cardinalidad	1,n
Entidad 2 -> Entidad 1 cardinalidad	1,1

Relación TIENE

Descripción de la relación TIENE

Nombre	TIENE
Descripcion	Relación que se establece entre las entidades Entidad y Atributo, representa que una entidad tiene que estar compuesta

	por uno o mas atributos.
Entidad 1	ENTIDAD
Entidad 2	ATRIBUTO
Entidad 1 -> Entidad 2 cardinalidad	1,n
Entidad 2 -> Entidad 1 cardinalidad	1,1

Relación es propietario de

Descripción de la relación es propietario de

Nombre	es propietario de
Descripción	Relación que se establece entre las entidades usuario y modelo, y representa que un usuario puede ser propietario de uno o mas modelos de datos
Entidad 1	USUARIO
Entidad 2	MODELO
Entidad 1 -> Entidad 2 cardinalidad	1,n
Entidad 2 -> Entidad 1 cardinalidad	1,1

Relación se aplica a

Descripción de la relación se aplica a

Nombre	se aplica a
Descripción	Relación que se establece entre las entidades dominio y Atributo y representa que un dominio puede ser aplicado a uno o muchos atributos.
Entidad 1	DOMENIO
Entidad 2	ATRIBUTO
Entidad 1 -> Entidad 2 cardinalidad	0,n
Entidad 2 -> Entidad 1 cardinalidad	0,1

APÉNDICE P.PRUEBAS

P.1. Pruebas caja negra

Caso de Prueba #1: Crear entidad.

Prueba 1.1 Curso normal:

Descripción: Verifica que el usuario pueda crear una entidad dentro la versión activa de un modelo conceptual de datos específico.

Entrada: El usuario escoge la opción entidad en la barra de herramientas e introduce cada uno de los datos solicitados para luego presionar el botón "continuar". Luego de haber agregado los atributos que sean necesarios y después de validados los datos, presiona el botón "Finalizar".

Salida: El sistema muestra el mensaje "La entidad nombreEntidad ha sido agregada a la versión nombreVersion", y se muestra dicha entidad gráficamente.

Prueba 1.2 Curso alterno:

Descripción: Verifica que el sistema no cree una entidad, cuando el usuario, presione el botón "Cancelar" en la ventana "Entidad".

Entrada: El usuario escoge la opción entidad en la barra de herramientas e introduce cada uno de los datos solicitados, posteriormente presiona el botón "Cancelar".

Salida: El sistema cierra la ventana "Entidad" y retorna al espacio de trabajo del usuario.

Prueba 1.3 Curso alterno:

Descripción: Verifica que el sistema muestre un mensaje de error en el caso que el usuario llene erróneamente los datos solicitados en la ventana entidad o deje algún campo vacío.

Entrada: El usuario escoge la opción entidad en la barra de herramientas e introduce incorrectamente los datos solicitados o deja de ingresar alguno.

Salida: El sistema muestra un mensaje de error.

Prueba 1.4 Curso alterno:

Descripción: Verifica que cuando el usuario se encuentra en la ventana de agregar atributos a la entidad y se presiona el botón "Cancelar" el sistema muestre un mensaje de advertencia con la confirmación de su acción.

Entrada: El usuario presiona el botón "Cancelar" en la ventana de atributos.

Salida: El sistema muestra un mensaje de advertencia donde le indica al usuario, que al cancelar la operación se perderán los datos de la entidad. De confirmar la operación, el sistema vuelve al espacio de trabajo, sino vuelve a la ventana de "Atributos".

Prueba 1.5 Curso alterno:

Descripción: Verifica que si el usuario no agrega algún atributo, al presionar el botón "Finalizar" el sistema muestre un mensaje de error.

Entrada: El usuario no agrega ningún atributo a la entidad y presiona el botón "Finalizar" en la ventana Atributo.

Salida: El sistema muestra un mensaje en el que se le indica al usuario que debe existir al menos un atributo clave dentro de la entidad.

Prueba 1.6 Curso alterno:

Descripción: Verifica que el sistema muestre un mensaje de error cuando el usuario no seleccione algún atributo como clave de la entidad.

Entrada: El usuario no selecciona algún atributo como clave.

Salida: El sistema muestra un mensaje en el que se le indica al usuario que debe existir al menos un atributo clave dentro de la entidad.

Caso de Prueba #2: Eliminar entidad.

Prueba 2.1 Curso normal:

Descripción: Prueba que el usuario pueda eliminar una entidad dentro la versión activa de un modelo conceptual de datos específico.

Entrada: El usuario selecciona la entidad que desea eliminar en el árbol de navegación y presiona el icono correspondiente a la opción "Eliminar" en la paleta.

Salida: El sistema muestra un mensaje de confirmación con el siguiente formato: ¿Desea eliminar la entidad nombreEntidad de la versión nombreVersion?, y elimina la entidad gráficamente.

Prueba 2.2 Curso alterno:

Descripción: Verifica que no ocurra alguna acción en el caso de que el usuario unicamente seleccione una entidad en la árbol de navegación.

Entrada: El usuario selecciona una entidad en el árbol de navegación.

Salida: El sistema no realiza operación alguna.

Prueba 2.3 Curso alterno:

Descripción: Verifica que no se elimine la entidad en caso de que el usuario cancele la operación.

Entrada: El usuario cancela la operación de eliminación desde el mensaje de confirmación que se muestra cuando se presiona el icono eliminar en la paleta.

Salida: El sistema vuelve al espacio de trabajo y no elimina la entidad.

Caso de Prueba #3: Modificar entidad.

Prueba 3.1 Curso normal:

Descripción: Prueba que el usuario pueda modificar los datos de una entidad dentro la versión activa de un modelo conceptual de datos específico.

Entrada: El usuario hace "doble click" en la entidad que desea modificar en el árbol de navegación.

Salida: El sistema muestra un mensaje de confirmación de los cambios realizados y modifica la entidad gráficamente.

Prueba 3.2 Curso alterno:

Descripción: Verifica que no ocurra alguna acción en el caso de que el usuario no mofique los campos de la entidad y presione el botón "Finalizar".

Entrada: El usuario presiona el botón "Finalizar" sin modificar los datos.

Salida: El sistema no modifica la entidad y regresa al espacio de trabajo.

Prueba 3.3 Curso alterno:

Descripción: Verifica que el sistema muestre un mensaje de error en el caso que el usuario llene erróneamente los datos o deje algún campo vacío.

Entrada: El usuario al modificar los datos introduce incorrectamente los datos solicitados o deja de ingresar alguno.

Salida: El sistema muestra un mensaje de error.

Prueba 3.4 Curso alterno:

Descripción: Verifica que el sistema no modifique la entidad al presionar el botón "Cancelar".

Entrada: El usuario hace "doble click" en una entidad e introduce cada uno de los datos que desea modificar, posteriormente presiona el botón "Cancelar".

Salida: El sistema cierra la ventana "Entidad" y retorna al espacio de trabajo del usuario.

Prueba 3.5 Curso alterno:

Descripción: Verifica que el sistema no modifique la entidad, si el usuario elimina todos los atributos que esta posee.

Entrada: El usuario elimina todos los atributos de la entidad y presiona el botón "Modificar"

Salida: El sistema muestra un mensaje de error donde le indica al usuario que debe existir al menos un atributo clave a la entidad.

Prueba 3.6 Curso alterno:

Descripción: Verifica que el sistema no modifique la entidad, si el usuario no selecciona algún atributo como clave.

Entrada: El usuario modifica los atributos de la entidad y no selecciona alguno de estos como clave, posterioremente presiona el botón "Finalizar".

Salida: El sistema muestra un mensaje de error donde le indica al usuario que debe existir al menos un atributo clave a la entidad.

Caso de Prueba #5: Eliminar Modelo.

Prueba 5.1 Curso normal:

Descripción: Prueba que el usuario pueda eliminar una modelo de su espacio de trabajo.

Entrada: El usuario selecciona el modelo que desea eliminar en el árbol de navegación y presiona el icono correspondiente a la opción "Eliminar" en la paleta.

Salida: El sistema muestra un mensaje de confirmación con el siguiente formato: ¿Desea eliminar el modelo nombreModelo?, y lo elimina.

Prueba 5.2 Curso alterno:

Descripción: Verifica que no ocurra alguna acción en el caso de que el usuario unicamente seleccione el modelo en la árbol de navegación.

Entrada: El usuario selecciona el modelo en el árbol de navegación.

Salida: El sistema no realiza operación alguna.

Prueba 5.3 Curso alterno:

Descripción: Verifica que no se elimine el modelo en caso de que el usuario cancele la operación.

Entrada: El usuario cancela la operación de eliminación desde el mensaje de confirmación que se muestra cuando se presiona el icono modelo en la paleta.

Salida: El sistema vuelve al espacio de trabajo y no elimina la modelo.

Caso de Prueba #6: Modificar modelo.

Prueba 6.1 Curso normal:

Descripción: Prueba que el usuario pueda modificar los datos de un modelo dentro de su espacio de trabajo.

Entrada: El usuario hace "doble click" en el modelo que desea modificar en el árbol de navegación.

Salida: El sistema muestra un mensaje de confirmación de los cambios realizados y modifica el modelo.

Prueba 6.2 Curso alterno:

Descripción: Verifica que el sistema no modifique el modelo al presionar el botón "Cancelar".

Entrada: El usuario hace "doble click" en el modelo e introduce cada uno de los datos que desea modificar, posteriormente presiona el botón "Cancelar".

Salida: El sistema cierra la ventana "Modelo" y retorna al espacio de trabajo del usuario.

Prueba 6.3 Curso alterno:

Descripción: Verifica que el sistema muestre un mensaje de error en el caso que el usuario llene erróneamente los datos o deje algún campo vacío.

Entrada: El usuario al modificar los datos introduce incorrectamente los datos solicitados o deja de ingresar alguno.

Salida: El sistema muestra un mensaje de error.

Prueba 6.4 Curso alterno:

Descripción: Verifica que el sistema no mofique el modelo en el caso que el usuario aborte la operación.

Entrada: El usuario presiona el botón "Cancelar" cuando el sistema le muestra el mensaje de confirmación de los cambios.

Salida: El sistema regresa al espacio de trabajo.

Caso de Prueba #7: Crear Dominio.

Prueba 7.1 Curso normal:

Descripción: Prueba que el usuario pueda agregar un dominio a la versión activa de su modelo.

Entrada: El usuario presiona el icono correspondiente a la opción "Dominio" en la paleta e introduce los datos.

Salida: El sistema agrega el dominio a la versión, para que pueda ser asignado a cualquier atributo de la versión.

Prueba 7.2 Curso alterno:

Descripción: Verifica que el sistema no cree un dominio, cuando el usuario, presione el botón "Cancelar" en la ventana "Dominio".

Entrada: El usuario presiona el botón "Cancelar" en la ventana "Dominio".

Salida: El sistema cierra la ventana "Dominio" y retorna al espacio de trabajo del usuario.

Prueba 7.3 Curso alterno:

Descripción: Verifica que el sistema muestre un mensaje de error en el caso que el usuario llene erróneamente los datos o deje algún campo vacío.

Entrada: El usuario introduce incorrectamente los datos solicitados o deja de ingresar alguno.

Salida: El sistema muestra un mensaje de error.

Caso de Prueba #8: Asignar dominio a atributo.

Prueba 8.1 Curso normal:

Descripción: Prueba que el usuario pueda asignar un dominio a los atributos de una entidad perteneciente a la versión activa.

Entrada: El usuario selecciona en la ventana "Atributo" uno de los dominios disponibles para algún atributo.

Salida: El sistema asigna el dominio al atributo.

Prueba 8.2 Curso alterno:

Descripción: Verifica que el sistema no se asigne algún dominio al atributo, cuando el usuario, presione el botón "Cancelar" en la ventana "Atributo".

Entrada: El usuario selecciona el dominio y presiona el botón "Cancelar" en la ventana "Atributo".

Salida: El sistema cierra la ventana "Atributo" y retorna al espacio de trabajo del usuario.

Caso de Prueba #9: Autentificarse en el sistema.

Prueba 9.1 Curso normal:

Descripción: Prueba que el usuario ingresar al sistema a través de su autentificación como un usuario registrado.

Entrada: El usuario introduce sus datos en la ventana de inicio de la aplicación.

Salida: El sistema da acceso a la herramienta.

Prueba 9.2 Curso alterno:

Descripción: Verifica que el sistema muestre un mensaje de error en el caso que el usuario llene erróneamente los datos o deje algún campo vacío.

Entrada: El usuario introduce incorrectamente los datos solicitados o deja de ingresar alguno.

Salida: El sistema muestra un mensaje de error.

Prueba 9.3 Curso alterno:

Descripción: Verifica que el sistema no le de acceso a la herramienta en el caso que este no este registrado.

Entrada: El usuario introduce sus datos y presiona el botón "Iniciar sesión".

Salida: El sistema muestra un mensaje de error y da la opción al usuario para que se registre.

Caso de Prueba #10: Registrar usuarios.

Prueba 10.1 Curso normal:

Descripción: Prueba que el usuario pueda ser registrado en el sistema.

Entrada: El usuario introduce sus datos en la ventana de registro.

Salida: El sistema registra los datos y adopta el usuario para el sistema.

Prueba 10.2 Curso alterno:

Descripción: Verifica que el sistema muestre un mensaje de error en el caso que el usuario llene erróneamente los datos o deje algún campo vacío.

Entrada: El usuario introduce incorrectamente los datos solicitados o deja de ingresar alguno.

Salida: El sistema muestra un mensaje de error.

Prueba 10.3 Curso alterno:

Descripción: Verifica que el sistema no registre el usuario al presionar el botón "Cancelar".

Entrada: El usuario introduce cada uno de los datos para su registro y presiona el botón "Cancelar".

Salida: El sistema cierra la ventana "Registro".

Prueba 10.4 Curso alterno:

Descripción: Verifica que el sistema muestre un mensaje de error en el caso que existan usuarios con los mismos datos en el sistema.

Entrada: El usuario introduce los datos solicitados.

Salida: El sistema muestra un mensaje de error.

Caso de Prueba #11: Crear Relación.

Prueba 11.1 Curso normal:

Descripción: Prueba que el usuario pueda agregar una relación a la versión activa.

Entrada: El usuario presiona el icono correspondiente a la opción "Relación" en la paleta e introduce los datos.

Salida: El sistema muestra un mensaje y muestra la relación gráficamente.

Prueba 11.2 Curso alterno:

Descripción: Verifica que el sistema no agregue la relación al presionar el botón "Cancelar".

Entrada: El usuario presiona el botón "Cancelar" en la ventana "Relación".

Salida: El sistema cierra la ventana "Relación" y retorna al espacio de trabajo del usuario.

Prueba 11.3 Curso alterno:

Descripción: Verifica que el sistema muestre un mensaje de error en el caso que el usuario llene erróneamente los datos o deje algún campo vacío.

Entrada: El usuario al crear el modelo,introduce incorrectamente los datos solicitados o deja de ingresar alguno.

Salida: El sistema muestra un mensaje de error.

Prueba 11.4 Curso alterno:

Descripción: Verifica que el sistema muestre un mensaje donde se le informe al usuario la forma correcta de modelar una relación muchos a muchos.

Entrada: El usuario escoge como cardinalidad de la relación "muchos a muchos".

Salida: El sistema muestra un mensaje informativo donde se le enseña al usuario la manera correcta de crear una relación de este tipo.

Caso de Prueba #12: Eliminar Relación.

Prueba 12.1 Curso normal:

Descripción: Prueba que el usuario pueda eliminar una relación de su versión.

Entrada: El usuario selecciona la relación que desea eliminar en el árbol de navegación y presiona el icono correspondiente a la opción "Eliminar" en la paleta.

Salida: El sistema muestra un mensaje de confirmación con el siguiente formato: ¿Desea eliminar el modelo nombreRelación?, y la elimina graficamente.

Prueba 12.2 Curso alterno:

Descripción: Verifica que no ocurra alguna acción en el caso de que el usuario unicamente seleccione la relación en el árbol de navegación.

Entrada: El usuario selecciona la relación en el árbol de navegación.

Salida: El sistema no realiza operación alguna.

Prueba 12.3 Curso alterno:

Descripción: Verifica que no se elimine la relación en caso de que el usuario cancele la operación.

Entrada: El usuario cancela la operación de eliminación desde el mensaje de confirmación que se muestra cuando se presiona el icono modelo en la paleta.

Salida: El sistema vuelve al espacio de trabajo y no elimina la relación.

Caso de Prueba #13: Modificar modelo.

Prueba 13.1 Curso normal:

Descripción: Prueba que el usuario pueda modificar los datos de una relación dentro de su espacio de trabajo.

Entrada: El usuario hace "doble click" en la relación que desea modificar en el árbol de navegación.

Salida: El sistema muestra un mensaje de confirmación de los cambios realizados y modifica la relación.

Prueba 13.2 Curso alterno:

Descripción: Verifica que no ocurra alguna acción en el caso de que el usuario no mofique los campos de la relació y presione el botón "Finalizar".

Entrada: El usuario presiona el botón "Finalizar" sin modificar los datos.

Salida: El sistema no modifica la relación y regresa al espacio de trabajo.

Prueba 13.3 Curso alterno:

Descripción: Verifica que el sistema no modifique la relación al presionar el botón "Cancelar".

Entrada: El usuario hace "doble click" en la relación e introduce cada uno de los datos que desea modificar, posteriormente presiona el botón "Cancelar".

Salida: El sistema cierra la ventana "Relación" y retorna al espacio de trabajo del usuario.

Prueba 13.4 Curso alterno:

Descripción: Verifica que el sistema muestre un mensaje de error en el caso que el usuario llene erróneamente los datos o deje algún campo vacío.

Entrada: El usuario al modificar los datos introduce incorrectamente los datos solicitados o deja de ingresar alguno.

Salida: El sistema muestra un mensaje de error.

Caso de Prueba #14: Modificar Dominio.

Prueba 14.1 Curso normal:

Descripción: Prueba que el usuario pueda modificar los datos de un dominio dentro de su espacio de trabajo.

Entrada: El usuario hace "doble click" en el dominio que desea modificar en el árbol de navegación.

Salida: El sistema muestra un mensaje de confirmación de los cambios realizados y modifica el dominio.

Prueba 14.2 Curso alterno:

Descripción: Verifica que no ocurra alguna acción en el caso de que el usuario no mofique los campos del dominio y presione el botón "Finalizar".

Entrada: El usuario presiona el botón "Finalizar" sin modificar los datos.

Salida: El sistema no modifica del dominio y regresa al espacio de trabajo.

Prueba 13.3 Curso alterno:

Descripción: Verifica que el sistema no modifique el dominio al presionar el botón "Cancelar".

Entrada: El usuario hace "doble click" en el dominio e introduce cada uno de los datos que desea modificar, posteriormente presiona el botón "Cancelar".

Salida: El sistema cierra la ventana "Dominio" y retorna al espacio de trabajo del usuario.

Prueba 13.4 Curso alterno:

Descripción: Verifica que el sistema muestre un mensaje de error en el caso que el usuario llene erróneamente los datos o deje algún campo vacío.

Entrada: El usuario al modificar los datos introduce incorrectamente los datos solicitados o deja de ingresar alguno.

Salida: El sistema muestra un mensaje de error.

Caso de Prueba #15: Crear Versión.

Prueba 15.1 Curso normal:

Descripción: Prueba que el usuario pueda crear una versión a sus modelos.

Entrada: El usuario presiona el icono correspondiente a la opción "Nueva versión" en la bara de herramienta e introduce los datos.

Salida: El sistema muestra un mensaje y activa la versión.

Prueba 15.2 Curso alterno.

Descripción: Verifica que el sistema no cree la versión al presionar el botón "Cancelar".

Entrada: El usuario presiona el botón "Cancelar" en la ventana "Versión".

Salida: El sistema cierra la ventana "Versión" y retorna al espacio de trabajo del usuario.

Prueba 15.3 Curso alterno:

Descripción: Verifica que el sistema muestre un mensaje de error en el caso que el usuario llene erróneamente los datos o deje algún campo vacío.

Entrada: El usuario al crear el modelo, introduce incorrectamente los datos solicitados o deja de ingresar alguno.

Salida: El sistema muestra un mensaje de error.

Caso de Prueba #16: Crear Usuario.

Prueba 16.1 Curso normal:

Descripción: Prueba que el usuario pueda crear un usuario para la base de datos.

Entrada: El usuario presiona el icono correspondiente a la opción "Usuario de base de datos" en la paleta e introduce los datos.

Salida: El sistema muestra un mensaje y crea el usuario.

Prueba 16.2 Curso alterno.

Descripción: Verifica que el sistema no cree el usuario al presionar el botón "Cancelar".

Entrada: El usuario presiona el botón "Cancelar" en la ventana "Usuario".

Salida: El sistema cierra la ventana "Usuario" y retorna al espacio de trabajo del usuario.

Prueba 16.3 Curso alterno:

Descripción: Verifica que el sistema muestre un mensaje de error en el caso que el usuario llene erróneamente los datos o deje algún campo vacío.

Entrada: El usuario al crear el modelo, introduce incorrectamente los datos solicitados o deja de ingresar alguno.

Salida: El sistema muestra un mensaje de error.

Prueba 16.4 Curso alterno.

Descripción: Verifica que el sistema no cree el usuario al presionar el botón "Cancelar" en la ventana de "Roles" o "Privilegios".

Entrada: El usuario presiona el botón "Cancelar" en la ventana "Rol" o "Privilegio".

Salida: El sistema cierra la ventana y retorna al espacio de trabajo del usuario.

Caso de Prueba #17: Crear Rol.

Prueba 17.1 Curso normal:

Descripción: Prueba que el usuario pueda crear un rol para los usuarios de una versión.

Entrada: El usuario presiona selecciona la opción agregar en la ventana "Rol".

Salida: El sistema muestra un mensaje y agrega el rol a la lista de roles disponible en la versión.

Prueba 17.2 Curso alterno.

Descripción: Verifica que el sistema no cree el rol al presionar el botón "Cancelar".

Entrada: El usuario presiona el botón "Cancelar" en la ventana "Rol".

Salida: El sistema cierra la ventana "Rol" y retorna a la ventana de "Usuario".

Prueba 17.3 Curso alterno:

Descripción: Verifica que el sistema muestre un mensaje de error en el caso que el usuario llene erróneamente los datos o deje algún campo vacío.

Entrada: El usuario al crear el modelo, introduce incorrectamente los datos solicitados o deja de ingresar alguno.

Salida: El sistema muestra un mensaje de error.

Caso de Prueba #18: Modificar Rol.

Prueba 18.1 Curso normal:

Descripción: Prueba que el usuario pueda modificar los datos de un rol.

Entrada: El usuario hace "doble click" en el rol que desea modificar en el árbol de navegación.

Salida: El sistema muestra un mensaje de confirmación de los cambios realizados y modifica al rol.

Prueba 18.2 Curso alterno:

Descripción: Verifica que no ocurra alguna acción en el caso de que el usuario no mofique los campos del rol y presione el botón "Finalizar".

Entrada: El usuario presiona el botón "Finalizar" sin modificar los datos.

Salida: El sistema no modifica del rol y regresa al espacio de trabajo.

Prueba 18.3 Curso alterno:

Descripción: Verifica que el sistema no modifique el rol al presionar el botón "Cancelar".

Entrada: El usuario hace "doble click" en el rol e introduce cada uno de los datos que desea modificar, posteriormente presiona el botón "Cancelar".

Salida: El sistema cierra la ventana "Rol" y retorna al espacio de trabajo del usuario.

Prueba 18.4 Curso alterno:

Descripción: Verifica que el sistema muestre un mensaje de error en el caso que el usuario llene erróneamente los datos o deje algún campo vacío.

Entrada: El usuario al modificar los datos introduce incorrectamente los datos solicitados o deja de ingresar alguno.

Salida: El sistema muestra un mensaje de error.

Caso de Prueba #19: Eliminar Rol.

Prueba 19.1 Curso normal:

Descripción: Prueba que el usuario pueda eliminar una rol de su versión.

Entrada: El usuario selecciona el rol que desea eliminar en el árbol de navegación y presiona el icono correspondiente a la opción "Eliminar" en la paleta.

Salida: El sistema muestra un mensaje de confirmación con el siguiente formato: ¿Desea eliminar el rol nombreRol?, y lo elimina.

Prueba 19.2 Curso alterno:

Descripción: Verifica que no ocurra alguna acción en el caso de que el usuario unicamente seleccione el rol en el árbol de navegación.

Entrada: El usuario selecciona el rol en el árbol de navegación.

Salida: El sistema no realiza operación alguna.

Prueba 19.3 Curso alterno:

Descripción: Verifica que no se elimine la rol en caso de que el usuario cancele la operación.

Entrada: El usuario cancela la operación de eliminación desde el mensaje de confirmación que se muestra cuando se presiona el icono modelo en la paleta.

Salida: El sistema vuelve al espacio de trabajo y no elimina el rol.

Caso de Prueba #20: Generar modelo lógico.

Prueba 20.1 Curso normal:

Descripción: Prueba que el usuario pueda generar un modelo lógico a partir de una versión conceptual activa.

Entrada: El usuario selecciona la opción "Generar modelo lógico" de la barra de herramientas.

Salida: El sistema muestra muestra gráficamente la versión lógica del modelo.

Prueba 20.2 Curso alterno:

Descripción: Verifica que no se realice el modelo lógico en caso de que el usuario cancele la operación.

Entrada: El usuario cancela la operación presionando el botón "Cancelar" en la ventana de generación.

Salida: El sistema vuelve al espacio de trabajo y no realiza la operación.

Caso de Prueba #21: Generar modelo físico.

Prueba 21.1 Curso normal:

Descripción: Prueba que el usuario pueda generar un modelo físico a partir de una versión lógica de su modelo.

Entrada: El usuario selecciona la opción "Generar modelo físico" de la barra de herramientas.

Salida: El sistema genera los scripts sql seleccionados.

Prueba 21.2 Curso alterno:

Descripción: Verifica que no se realice el modelo físico en caso de que el usuario cancele la operación.

Entrada: El usuario cancela la operación presionando el botón "Cancelar" en la ventana de generación.

Salida: El sistema vuelve al espacio de trabajo y no realiza la operación.

Caso de Prueba #22: Generar reportes.

Prueba 22.1 Curso normal:

Descripción: Prueba que el usuario pueda generar un reporte de diccionario de datos opara sus modelos.

Entrada: El usuario selecciona la opción "Generar modelo reporte" de la barra de herramientas.

Salida: El sistema genera un reporte de diccionario, en formato HTML.

Prueba 22.2 Curso alterno:

Descripción: Verifica que no se realice el reporte en caso de que el usuario cancele la operación.

Entrada: El usuario cancela la operación presionando el botón "Cancelar" en la ventana de generación.

Salida: El sistema vuelve al espacio de trabajo y no realiza la operación.

Caso de Prueba #23: Realizar Ingeniería Reversa.

Prueba 23.1 Curso normal:

Descripción: Prueba que el usuario pueda realizar ingeniería reversa a partir de un archivo sql.

Entrada: El usuario selecciona la opción "Realizar Ingeniería Reversa" de la barra de herramientas y le proporciona la ruta donde se encuentra el archivo sql.

Salida: El sistema genera un modelo lógico y lo muestra graficamente.

Prueba 23.2 Curso alterno:

Descripción: Verifica que no se realice el proceso de ingeniería reversa en caso de que el usuario cancele la operación.

Entrada: El usuario cancela la operación presionando el botón "Cancelar" en la ventana de generación.

Salida: El sistema vuelve al espacio de trabajo y no realiza la operación.

Caso de Prueba #24: Recuperar versión.

Prueba 24.1 Curso normal:

Descripción: Prueba que el usuario pueda recuperar una versión de cualquiera de sus modelos.

Entrada: El usuario selecciona la opción "Abrir" de la barra de herramientas y le selecciona la versión que desea recuperar.

Salida: El sistema activa la versión seleccionada y la muestra en el espacio de trabajo.

Prueba 24.2 Curso alterno:

Descripción: Verifica que no se realice el proceso recuperación en caso de que el usuario cancele la operación.

Entrada: El usuario cancela la operación presionando el botón "Cancelar" en la ventana.

Salida: El sistema vuelve al espacio de trabajo y no realiza la operación.

Caso de Prueba #25: Realizar consulta.

Prueba 24.1 Curso normal:

Descripción: Prueba que el usuario pueda realizar consultas sobre sus modelos.

Entrada: El usuario selecciona la opción "Consulta" de la paleta e introduce los datos.

Salida: El sistema muestra la consulta en el lenguaje de la herramienta.

Prueba 24.2 Curso alterno:

Descripción: Verifica que no se realice la consulta en caso de que el usuario cancele la operación.

Entrada: El usuario cancela la operación presionando el botón "Cancelar" en la ventana.

Salida: El sistema vuelve al espacio de trabajo y no realiza la operación.

P.2.Pruebas con usuarios.

P.2.1. Encuesta.

¿Cual es tu nivel de Conocimientos en el área de Base de datos?

a) Avanzado.
b) Medio.
c) Bajo.

¿Como considera el espacio de trabajo, muestra los elementos de los modelos de manera clara?

a) Si.
b) Medianamente.
c) No.

¿La herramienta resulta fácil de operar?

a) Si.
b) Medianamente.
c) No.

¿Como considera los tiempos de respuesta de las operaciones?

a) Excelente
b) Muy bueno
c) Bueno
d) Regular
e) Malo

¿Al probarlo ocurrieron perdidas en los datos?.

a) Excelente
b) Muy bueno
c) Bueno
d) Regular
e) Malo

¿Como es la asistencia para los errores? La navegación de la herramienta te ayuda a no cometer errores?

a) Excelente
b) Muy bueno
c) Bueno
d) Regular
e) Malo

¿Las operaciones realizadas mediante el uso de la herramienta, están siempre guiadas?

a) Excelente
b) Muy bueno
c) Bueno
d) Regular
e) Malo

La presentación de los modelos físicos generados por la herramienta los consideras:

a) Excelente
b) Muy bueno
c) Bueno
d) Regular
e) Malo

¿Necesitas capacitación para usar la herramienta?

a) Si
b) Medianamente.
c) No.

En lineas generales el nivel de satisfacción del sistema.

a)Insatisfecho.

b)Medianamente Satisfecho.

c) Satisfecho.

d)Totalmente Satisfecho.

¿Que no te gusta de la herramienta?

¿Que te gusta de la herramienta?

¿Que le agregarías?

Respuestas.

Muestra: 10 personas

1. ¿Cual es tu nivel de Conocimientos en el área de Base de datos?

Opción	Descripción	% de Respuesta
A	Avanzado.	20
B	Medio.	50
C	Bajo	10

Tabla 21. Resultados pregunta #1 (Prueba).
Fuente: Elaborado por los autores.

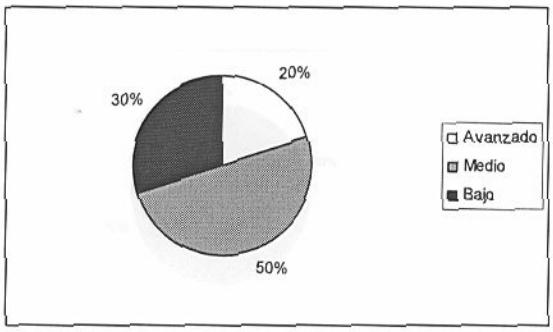


Figura 92.Gráfico pregunta #1 (Prueba).
Fuente: Elaborado por los autores.

2. ¿Como considera el espacio de trabajo, muestra los elementos de los modelos de manera clara?

Opción	Descripción	% de Respuesta
A	Si	30
B	Medianamente.	60
C	No.	10

Tabla 22. Resultados pregunta #2 (Prueba).
Fuente: Elaborado por los autores.

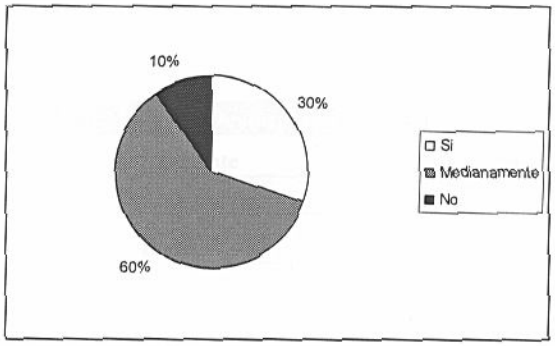


Figura 93.Gráfico pregunta #2 (Prueba).
Fuente: Elaborado por los autores.

3. ¿La herramienta resulta fácil de operar?

Opción	Descripción	% de Respuesta
A	Si	40
B	Medianamente.	60
C	No.	0

Tabla 23. Resultados pregunta #3 (Prueba).
Fuente: Elaborado por los autores.

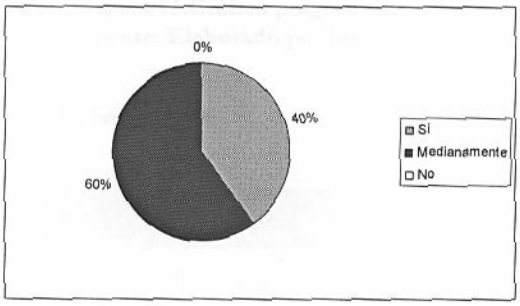


Figura 94.Gráfico pregunta #3 (Prueba).
Fuente: Elaborado por los autores.

4. ¿Como considera los tiempos de respuesta de las operaciones?

Opción	Descripción	% de Respuesta
A	Excelente	40
B	Muy Bueno	60
C	Bueno	0
D	Regular	0
E	Malo	0

Tabla 24. Resultados pregunta #4 (Prueba).
Fuente: Elaborado por los autores.

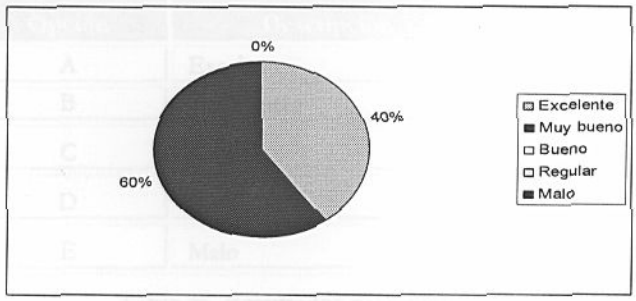


Figura 95.Gráfico pregunta #4 (Prueba).
Fuente: Elaborado por los autores.

5. ¿Al probarlo ocurrieron perdidas en los datos?

Opción	Descripción	% de Respuesta
A	Excelente	10
B	Muy Bueno	50
C	Bueno	30
D	Regular	10
E	Malo	0

Tabla 25. Resultados pregunta #5 (Prueba).
Fuente: Elaborado por los autores.

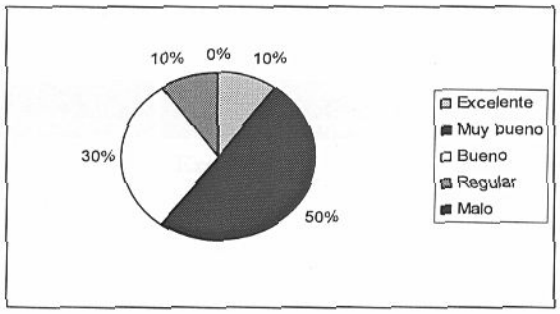


Figura 96.Gráfico pregunta #5 (Prueba).
Fuente: Elaborado por los autores.

6. ¿Las operaciones realizadas mediante el uso de la herramienta, están siempre guiadas?

Opción	Descripción	% de Respuesta
A	Excelente	20
B	Muy Bueno	40
C	Bueno	30
D	Regular	10
E	Malo	0

Tabla 26. Resultados pregunta #6 (Prueba).
Fuente: Elaborado por los autores.

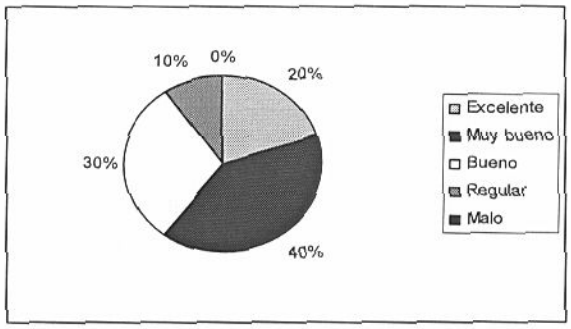


Figura 97.Gráfico pregunta #6 (Prueba).
Fuente: Elaborado por los autores.

7. La presentación de los modelos físicos generados por la herramienta los consideras:

Opción	Descripción	% de Respuesta
A	Excelente	40
B	Muy Bueno	30
C	Bueno	30
D	Regular	0
E	Malo	0

Tabla 27. Resultados pregunta #7 (Prueba).
Fuente: Elaborado por los autores.

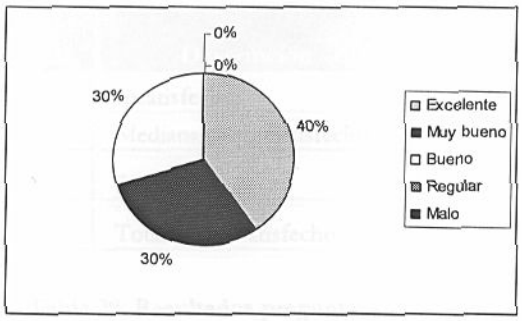


Figura 98.Gráfico pregunta #7 (Prueba).
Fuente: Elaborado por los autores.

8. ¿Necesitas capacitación para usar la herramienta?

Opción	Descripción	% de Respuesta
A	Si	0
B	Medianamente.	30
C	No.	70

Tabla 28. Resultados pregunta #8 (Prueba).
Fuente: Elaborado por los autores.

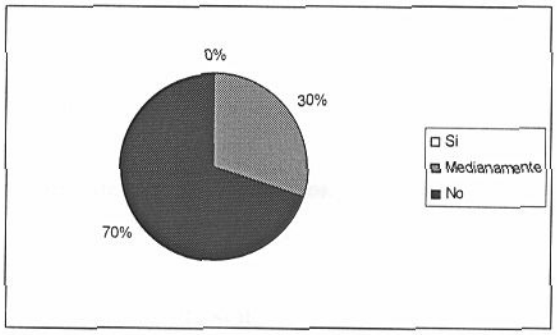


Figura 99.Gráfico pregunta #8 (Prueba).
Fuente: Elaborado por los autores.

9. En lineas generales el nivel de satisfacción del sistema.

Opción	Descripción	% de Respuesta
A	Insatisfecho.	0
B	Medianamente Satisfecho.	10
C	Satisfecho.	50
D	Totalmente Satisfecho.	40

Tabla 29. Resultados pregunta #9 (Prueba).
Fuente: Elaborado por los autores.

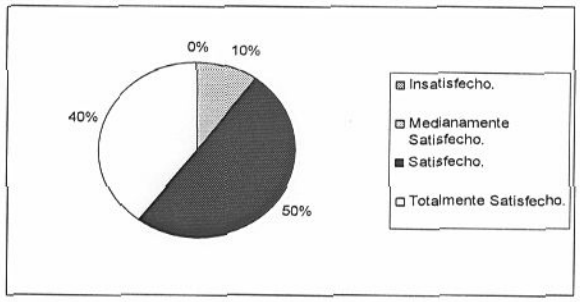


Figura 100.Gráfico pregunta #9 (Prueba).
Fuente: Elaborado por los autores.

10.¿Que no te gusta de la herramienta?

- Hay que familarizarce con la interfaz para navegarla mas cómodamente.
- Permitir Modificar el tamaño de las entidades y Tablas Gráficamente.
- Para crear las relaciones entre tablas y/o entidades, hay que especificar más gráficamente.

11.¿Que te gusta de la herramienta?

Interfaz Agradable.

Operaciones Lógicas y Fáciles.

Colores Agradables.

Iconos de botones y barras de tareas muy intuitivos.

12. ¿Que le agregarías?

Modulo para generación de código PL/SQL.

Modulo para creación de Triggers.

Diagramas de Clases, Casos de Uso, Etc.

APÉNDICE Q. TECNOLOGÍAS UTILIZADAS PARA LA REALIZACIÓN DE LA INTERFAZ.

Q.1.AWT y Swing.

AWT es el acrónimo del X Window Toolkit para Java, se trata de una biblioteca de clases Java para el desarrollo de interfaces de usuario gráficas, la cual ofrece un entorno demasiado simple, ya que no se han tenido en cuenta las ideas de entornos gráficos novedosos.

Javasoft, en vista de la precariedad de que hace gala el AWT, y para asegurarse que los elementos que desarrolla para generar interfaces gráficas sean fácilmente transportables entre plataformas, se ha unido con Netscape, IBM y Lighthouse Design para crear un conjunto de clases que proporcionen una sensación visual agradable y sean más fáciles de utilizar por el programador. Esta colección de clases son las *Java Foundation Classes* (JFC), que están constituidas por cinco grupos de clases, al menos en este momento: AWT, Java 2D, Accesibilidad, Arrastrar y Soltar y Swing.

AWT, engloba a todos los componentes del AWT que existían en la versión 1.1.2 del JDK y en los que se han incorporado en versiones posteriores:

- **Java 2D** es un conjunto de clases gráficas bajo licencia de IBM/Taligent, que todavía está en construcción
- **Accesibilidad**, proporciona clases para facilitar el uso de ordenadores y tecnología informática a disminuidos, como lupas de pantalla, y cosas así.
- **Arrastrar y Soltar** (*Drag and Drop*), son clases en las que se soporta Glasgow, que es la nueva generación de los JavaBeans
- **Swing**, es la parte más importante y la que más desarrollada se encuentra. Ha sido creada en conjunción con Netscape y proporciona una serie de componentes muy bien descritos y especificados de forma que su presentación visual es independiente de la plataforma en que se ejecute el applet o la aplicación que utilice estas clases. **Swing** simplemente extiende el AWT añadiendo un conjunto de componentes, *JComponents*, y sus clases de soporte. Hay un conjunto de componentes de Swing que son análogos a los de AWT, y algunos de ellos participan de la arquitectura MVC (*Modelo-Vista-Controlador*), aunque **Swing** también proporciona otros widgets nuevos como árboles, pestañas, etc.

La estructura básica del AWT se basa en *Componentes y Contenedores*. Estos últimos contienen Componentes posicionados a su respecto y son Componentes a su vez, de forma que los eventos pueden tratarse tanto en Contenedores como en Componentes, corriendo por cuenta del programador (todavía no hay herramientas de composición visual) el encaje de todas las piezas, así como la seguridad de tratamiento de los eventos adecuados. Con Swing se va un paso más allá, ya que todos los *JComponentes* son subclases de **Container**, lo que hace posible que widgets Swing puedan contener otros componentes, tanto de AWT como de Swing, lo que hace prever interesantes posibilidades.

Q.1.1.Estructura del AWT

La estructura de la versión actual del AWT se puede resumir en los puntos que se exponen a continuación:

- Los Contenedores contienen Componentes, que son los controles básicos.
- No se usan posiciones fijas de los Componentes, sino que están situados a través de una disposición controlada (layouts).
- El común denominador de más bajo nivel se acerca al teclado, ratón y manejo de eventos
- Alto nivel de abstracción respecto al entorno de ventanas en que se ejecute la aplicación (no hay áreas cliente, ni llamadas a X, ni hWnds, etc.).
- La arquitectura de la aplicación es dependiente del entorno de ventanas, en vez de tener un tamaño fijo.
- Es bastante dependiente de la máquina en que se ejecuta la aplicación (no puede asumir que un diálogo tendrá el mismo tamaño en cada máquina).
- Carece de un formato de recursos. No se puede separar el código de lo que es propiamente interface. No hay ningún diseñador de interfaces (todavía).

Q.1.2.Componentes y Contenedores

Un interfaz gráfico está construida en base a elementos gráficos básicos, los *Componentes*. Típicos ejemplos de estos Componentes son los botones, barras de desplazamiento, etiquetas, listas, cajas de selección o campos de texto. Los Componentes permiten al usuario interactuar con la aplicación y proporcionar información desde el programa al usuario sobre el estado del programa. En el AWT, todos los Componentes de la interface de usuario son instancias de la clase **Component** o uno de sus subtipos.

Los Componentes no se encuentran aislados, sino agrupados dentro de *Contenedores*. Los Contenedores contienen y organizan la situación de los Componentes; además, los Contenedores son en sí mismos Componentes y como tales pueden ser situados dentro de otros Contenedores. También contienen el código necesario para el control de eventos, cambiar la forma del cursor o modificar el icono

de la aplicación. En el AWT, todos los Contenedores son instancias de la clase **Container** o uno de sus subtipos.

Q.2. GEF.

GEF es un framework de desarrollo realizado en Java y basado en MVC, el cual fue desarrollado por personas de la Universidad de California y cuyo objetivo es permitir la construccion de aplicaciones que necesiten la creacion de diferentes graficos y formas.

Provee herramientas para crear y manipular estas formas de manera tal que se puedan visualizar gráficos complejos conectados de diversas maneras.

Existen dos niveles principales en los gráficos que pueden ser creados a través de GEF: el nivel de red y el nivel de diagrama. El nivel de red es aquel que posee todos los objetos lógicos, que pueden tener datos y comportamientos específicos de uso.Mientras que el nivel de diagrama permite estructurar una representacion visual de una red de varias anotaciones.

Q.2.1. Clases principales.

- Editor: es la clase central del sistema. Existe una instancia del redactor para cada diagrama que se exhiba en la pantalla. Su función es enviar mensajes a otras clases para que se ejecuten las acciones correspondientes.
- Cmds: en estas clases se define el método doIt() para que realice cierta acción en el Editor. Por ejemplo la clase CmdReorder posee los menus para ordenar y agrupar los diagramas.
- Guides: ayuda al usuario a realizar diagramas que se vean limpias de acuerdo a ciertas coordenadas.
- Figs: son los objetos que se pueden mostrar y manipular en el Editor. Posee los elementos de dibujo clasicos tales como lineas y triangulos.