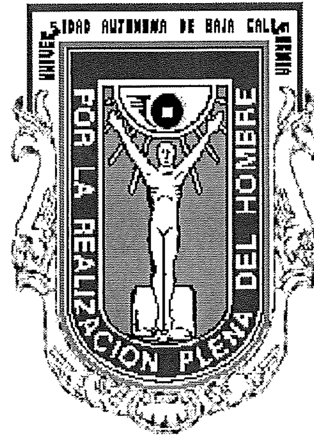
UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

FACULTAD DE CIENCIAS



DESARROLLO DE UNA HERRAMIENTA PARA LA REPRESENTACIÓN DEL CONOCIMIENTO USANDO REDES SEMÁNTICAS

TESIS

QUE PARA OBTENER EL TÍTULO DE

LICENCIADO EN CIENCIAS COMPUTACIONALES

PRESENTA:

ULISES ANTONIO FLETES GONZÁLEZ

ENSENADA, BAJA CALIFORNIA. MÉXICO. FEBRERO DE 2002

UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

FACULTAD DE CIENCIAS

DESARROLLO DE UNA HERRAMIENTA PARA LA REPRESENTACIÓN DEL CONOCIMIENTO USANDO REDES SEMÁNTICAS

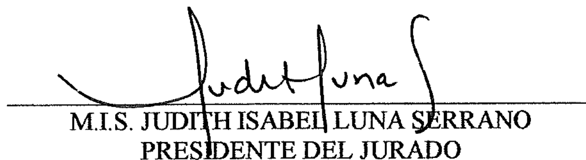
TESIS QUE PARA OBTENER EL TÍTULO DE

LICENCIADO EN CIENCIAS COMPUTACIONALES

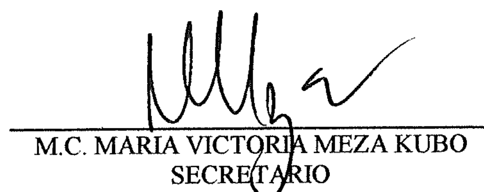
PRESENTA:

ULISES ANTONIO FLETES GONZÁLEZ

APROBADO POR:



M.I.S. JUDITH ISABEL LUNA SERRANO
PRESIDENTE DEL JURADO



M.C. MARIA VICTORIA MEZA KUBO
SECRETARIO

M.C. MIGUEL ANGEL IBARRA RIVERA
1ER VOCAL

Dedicatoria

Quiero dedicar esta tesis a mis padres

y

hermanos

que tanto añoraron este momento.

También a toda mi familia donde quiera que se encuentren.

Agradecimientos

Quiero agradecer a Dios y a mi Virgen por darme la oportunidad de vivir y permitirme concluir con esta etapa de mis estudios profesionales.

A mis padres que con gran sacrificio me brindaron el apoyo durante mis estudios y en especial a mi madre que me alentó durante los momentos difíciles.

A mis hermanos Miny, Dalia, Noe por el gran apoyo que me brindaron.

A todos mis compañeros y amigos y en especial a EMZR que me brindo su amistad, compañerismo y apoyo, que juntos compartimos momentos agradables.

A mis asesores de tesis Leopoldo Morán y Solares, Judith Luna Serrano.

A la UABC por el apoyo económico para la realización de esta tesis.

RESUMEN de la tesis de ULISES ANTONIO FLETES GONZÁLEZ presentada como requisito parcial para la obtención del título de Licenciado en Ciencias Computacionales. Ensenada, Baja California, México, Febrero de 2002.

DESARROLLO DE UNA HERRAMIENTA PARA LA REPRESENTACIÓN DEL CONOCIMIENTO USANDO REDES SEMÁNTICAS

Resumen aprobado por:

M.I.S. JUDITH ISABEL LUNA SERRANO
Presidente

En esta tesis se presenta el análisis, diseño e implementación de una herramienta para la representación del conocimiento a través de redes semánticas utilizando la técnica de modelado por objetos (OMT). Los conceptos representados pueden incluir información asociada como imágenes, textos, gráficas, sinónimos y sonidos. Tiene la particularidad de permitir al usuario, extender su red semántica al agregar más información al esquema, llegando a crear redes extensas. Una vez representado el concepto, el usuario puede crear otras redes, debido a que la herramienta permite navegar a través de la información. Además puede ser utilizada como una herramienta de referencia rápida, donde la información resultante de una consulta es representada de manera gráfica. Por estas características es una herramienta que facilita el aprendizaje en los cursos a distancia. La instrumentación del sistema fue utilizando el lenguaje Java, éste permite la portabilidad en múltiples plataformas. La presente versión fue probada en las plataformas Windows NT, Windows 9x y LINUX.

CONTENIDO

1. INTRODUCCIÓN.. 1
2.ANTECEDENTES.. 3
2.1. Objetivos.. 6
2.1.1. Objetivo General.. 6
2.1.2. Objetivos Específicos.. 6
3. METODOLOGÍA.. 7
4. DESARROLLO.. 9
4.1. Análisis.. 9
4.1.1. Conceptos básicos sobre redes semánticas.. 9
4.1.1.1. Conceptos.. 9
4.1.1.2. Relaciones.. 10
4.1.1.2.1. Relaciones Asimétricas.. 11
4.1.1.2.2. Relaciones Simétricas.. 11
4.1.1.3. Instancias.. 12
4.1.1.4. Reglas para la creación de redes semánticas.. 12
4.1.2. Requerimientos del sistema.. 13
4.1.3. Modelo de Objetos.. 14
4.1.3.1. Diccionario de Datos.. 16
4.1.3.1.1. Red Semántica.. 16
4.1.3.1.2. Instancia.. 17
4.1.3.1.3. Concepto.. 18
4.1.3.1.4. Relación.. 19
4.1.4. Modelo Dinámico.. 20
4.1.4.1. Escenario para la clase Red Semántica.. 20
4.1.4.2. Escenario para la clase Concepto.. 22
4.1.4.3. Escenario para la clase Relación.. 23
4.1.4.4. Escenario para la clase Instancia.. 24
4.1.4.5. Diagrama general de eventos.. 26
4.1.4.6. Diagrama de estados.. 26
4.1.4.6.1. Diagrama de estados para la clase Concepto.. 27
4.1.4.6.2. Diagrama de estados para la clase Relación.. 28
4.1.4.6.3. Diagrama de estados para la clase Instancia.. 29
4.1.4.7. Modelo Funcional.. 30
4.1.4.7.1. Diagrama de Flujo de Datos para la clase Red Semántica.. 30
4.1.4.7.2. Diagrama de Flujo de Datos para la clase Concepto.. 31
4.1.4.7.3. Diagrama de Flujo de Datos para la clase Relación.. 32
4.1.4.7.4. Diagrama de Flujo de Datos para la clase Instancia.. 33
4.2.Diseño de Sistemas.. 34
4.2.1. Arquitectura del Sistema.. 34
4.2.2. Relación entre Subsistemas.. 35
4.2.3. Manejo de recursos Globales.. 35
4.3. Diseño de objetos.. 36
4.3.1. Modelo de objetos detallado.. 36
4.3.1.1. Nombre, atributos y operaciones del modelo de objetos.. 37

4.3.2. Modelo Dinámico detallado........ 38
4.3.2.1.1. Escenarios y trazo de eventos para la clase Concepto........ 38
4.3.2.1.2. Escenario y trazo de eventos para la clase Relación........ 40
4.3.2.1.3. Escenario y trazo de eventos para la clase Instancia........ 42
4.3.2.2.1. Diagrama de estados para la clase Concepto........ 49
4.3.2.2.2. Diagrama de estados para la clase Relación........ 45
4.3.2.2.3. Diagrama de estados para la clase Instancia........ 46
4.3.3. Modelo Funcional Detallado........ 47
4.3.3.1. Modelo de Flujo de Datos para la clase Red Semántica........ 47
4.3.3.2. Modelo de Flujo de Datos para la clase Concepto........ 48
4.3.3.3. Modelo de Flujo de Datos para la clase Relación........ 49
4.3.3.4. Modelo de Flujo de Datos para la clase Instancia........ 50
4.3.4. Descripción de Algoritmos........ 51
4.3.4.1. Descripción de algoritmos de la clase Red Semántica........ 51
4.3.4.2. Descripción de algoritmos de la clase Concepto........ 55
4.3.4.3. Descripción de algoritmos de la clase Nodo........ 59
4.3.4.4. Descripción de algoritmos de la clase Relación........ 61
4.3.4.5. Descripción de algoritmos de la clase Flecha........ 62
4.3.4.6. Descripción de algoritmos de la clase Instancia........ 66
4.3.5. Implementación........ 68
5. RESULTADOS........ 69
6. DISCUSIÓN........ 79
7. CONCLUSIONES........ 82
8. LITERATURA CITADA........ 84
9. APÉNDICE A (implementación de clases del subsistema Máquina de representación del Conocimiento) 85
9.1. Clase Concepto........ 85
9.2. Clase Relación........ 89
9.3. Clase Flecha........ 91
9.4. Clase Nodo........ 95
9.5. Clase Instancia........ 97
9.6. Clase Red Semántica........ 99

LISTA DE FIGURAS

Figura 1.1. Figura general de red semántica.. 2
Figura 4.1. Ejemplo de una red semántica.. 10
Figura 4.2. Formas de representar instancias.. 12
Figura 4.3. Modelo de Objetos de RepConRS.. 15
Figura 4.4. Diagrama de Trazo de eventos de la clase Red Semántica.................. 21
Figura 4.5. Diagrama de Trazo de eventos de la clase Concepto........................... 23
Figura 4.6. Diagrama de Trazo de eventos de la clase Relación............................ 24
Figura 4.7. Diagrama de Trazo de eventos de la clase Instancia............................ 25
Figura 4.8. Diagrama General de eventos para RepConRs.................................... 26
Figura 4.9. Diagrama de estados de la clase Concepto.. 27
Figura 4.10. Diagrama de estados de la clase Relación... 28
Figura 4.11. Diagrama de estados de la clase Instancia... 29
Figura 4.12. Diagrama de Flujo de Datos para la clase Red Semántica................. 30
Figura 4.13. Diagrama de Flujo de Datos para la clase Concepto.......................... 31
Figura 4.14. Diagrama de Flujo de Datos para la clase Relación........................... 32
Figura 4.15. Diagrama de Flujo de Datos para la clase Instancia........................... 33
Figura 4.16. Arquitectura del Sistema.. 34
Figura 4.17. Modelo de objetos detallado.. 36
Figura 4.18. Nombre, atributos y operaciones del modelo de objetos.................... 37
Figura 4.19. Diagrama y trazo de eventos para la clase Concepto........................ 39
Figura 4.20. Diagrama y trazo de eventos para la clase Relación....................... 41
Figura 4.21. Diagrama y trazo de eventos para la clase Instancia........................ 43
Figura 4.22. Diagrama de estados para la clase Concepto.................................. 44
Figura 4.23. Diagrama de estados para la clase Relación...................................... 45
Figura 4.24. Diagrama de estados para la clase Instancia...................................... 46
Figura 4.25. Diagrama de Flujo de Datos para la clase Red Semántica................. 47
Figura 4.26. Diagrama de Flujo de Datos para la clase Concepto.......................... 48
Figura 4.27. Diagrama de Flujo de Datos para la clase Relación........................... 49
Figura 4.28. Diagrama de Flujo de Datos para la clase Instancia........................... 50
Figura 5.1. Muestra el concepto central sin título.. 71
Figura 5.2. Muestra la edición de un concepto central.. 71
Figura 5.3. Muestra la creación de una instancia.. 72
Figura 5.4. Muestra las instancias del concepto árbol... 72
Figura 5.5. Muestra las instancias del concepto planta.. 73
Figura 5.6. Muestra el diálogo para agregar texto al concepto............................. 74
Figura 5.7. Muestra el diálogo para agregar sonido al concepto........................... 74
Figura 5.8. Muestra el diálogo para agregar imagen al concepto........................... 75
Figura 5.9. Muestra la red semántica con el concepto central árbol y sus elementos agregados.. 76
Figura 5.10. Muestra el concepto árbol con la imagen agregada.......................... 76
Figura 5.11. Muestra el concepto árbol con el texto agregado.............................. 77
Figura 5.12. Muestra el concepto árbol con el sonido agregado............................ 77

1. INTRODUCCIÓN

La Inteligencia Artificial (IA) ha pasado de ser un pequeño aspecto de la ciencia informática a ser una de las aportaciones más importantes. Este rápido cambio se basa fundamentalmente en el éxito de los sistemas expertos que fueron los primeros productos de la IA de auténtico impacto comercial [Shildt, 1989].

Durante los años 60's y 70's gran parte de la comunidad de investigadores del área de Inteligencia Artificial se dedicaron a estudiar y entender el conocimiento humano para poder desarrollar Sistemas Expertos capaces de representar cualquier tipo de conocimiento. Un esquema de representación de conocimiento debe ser capaz de expresar cosas como objetos, eventos, relaciones, conceptos y metas. Los esquemas más importantes para la representación del conocimiento son:

- Las reglas de producción
- Los marcos
- Los objetos
- Las redes semánticas

Siendo este último uno de los esquemas más viejos y fáciles de comprender, las redes semánticas están compuestas de nodos y enlaces. Una red semántica es básicamente una descripción gráfica del conocimiento que muestra jerárquicamente relaciones entre objetos [Turban, 1992].

Las redes semánticas se representan mediante un grafo, donde los nodos corresponden a entidades, objetos o conceptos, mientras los enlaces o arcos corresponden a relaciones entre estos nodos [Bratko, 1990], véase figura 1.1.

Una herramienta de aprendizaje que utiliza el esquema de redes semánticas es SemNet [Fisher *et al*, 1990]. Esta herramienta apoya la construcción del conocimiento grupal o personal y se utiliza también para el análisis del conocimiento que ayuda a los usuarios a comprender textos complejos. La primer versión de SemNet se desarrolló para la plataforma Macintosh.

El objetivo del presente trabajo es desarrollar una herramienta con características similares a SemNet, que además pueda ejecutarse en diferentes plataformas de cómputo (Windows, Unix, Macintosh) con base en los requerimientos específicos del cliente.

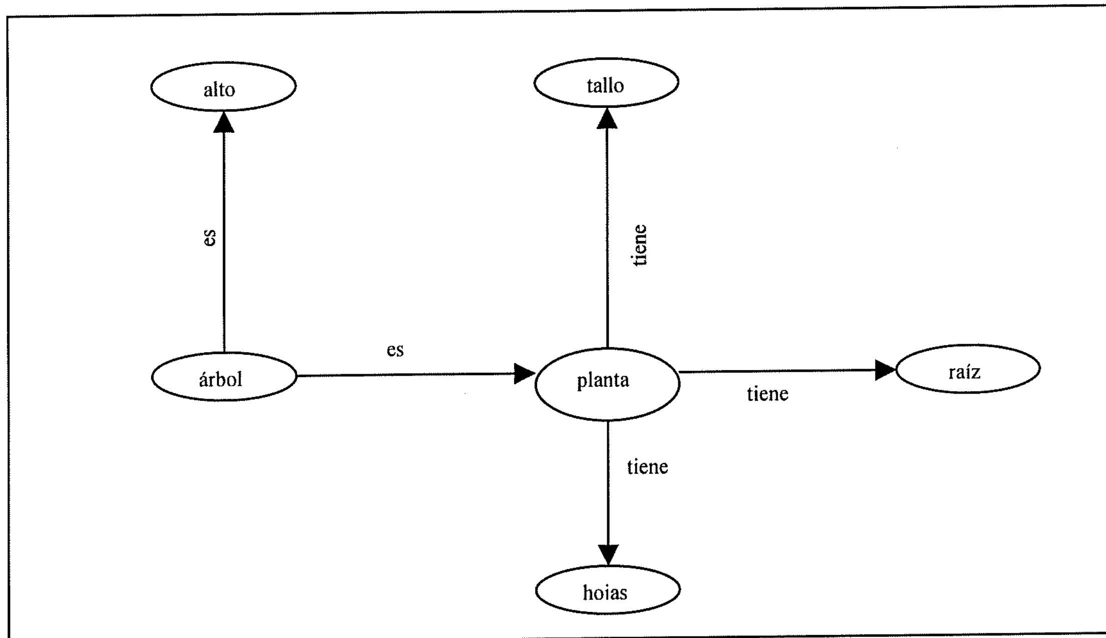


Figura 1.1. Figura general de red semántica.

2. ANTECEDENTES

Durante 1980 se desarrollaron los primeros sistemas expertos comerciales utilizando diferentes esquemas para la representación del conocimiento, la mayoría de estos sistemas fueron aplicados al campo de la medicina. A continuación se describen algunos sistemas expertos desarrollados bajo el esquema de redes semánticas.

- NEPHROS, desarrollado por ASBELL en 1981. Aplicado en el diagnóstico de insuficiencias renales.
- ABEL, desarrollado en el MIT en 1982. Aplicado en el diagnóstico y tratamiento de alteraciones en el balance electrolítico.
- AI / RHEUM, desarrollado por KINGSLAND, L. C. y LINDER, D en 1983. Aplicado en el diagnóstico en reumatología.
- MEDIKS, desarrollado por CHANG, L. C. y TOU, J. T. en 1984. Aplicado en el diagnóstico de medicina general [Sánchez, 1988].

Los sistemas expertos fueron las aplicaciones que más utilizaron las redes semánticas, sin embargo este esquema no sólo se puede aplicar a los sistemas expertos si no también a otras aplicaciones tal es el caso de SemNet. La primer versión del software SemNet fue desarrollado por un grupo de profesores de la Universidad de California en Davis [Fisher *et al*, 1990]. Esta versión fue usada primeramente como una herramienta de apoyo docente en el área de biología general en licenciatura. Cerca de 200 personas a nivel mundial realizaron pruebas como

usuarios Beta y muchos más adquirieron el software. Esta versión de SemNet se puede obtener de manera gratuita a través de Internet y el número actual de usuarios se acerca a miles [Fisher *et al*, 1990].

A través de las solicitudes de los usuarios alrededor del mundo, en años recientes se ha identificado la necesidad de tener una nueva herramienta que sea capaz de ejecutarse en diferentes plataformas [Fisher *et al*, 1990].

Para satisfacer esta necesidad, se estableció un grupo de trabajo conformado por profesores e investigadores de San Diego State University (SDSU), Universidad Autónoma de Baja California (UABC) y Universidad Popular Autónoma del Estado de Puebla (UPAEP) para desarrollar la nueva herramienta. Se han identificado cuatro roles para el desarrollo del proyecto y se han asignado los diferentes equipos. Los roles identificados y las partes a cargo de cada grupo son los siguientes:

Grupo: Colegio de Ciencias, SDSU.
Actividad: Administración general del proyecto.
Responsable: Dr. Roberto Pozos.

Grupo: Centro de Investigación en Matemáticas y Ciencias de la Educación, SDSU.
Actividad: Usuario y Proveedor de Especificación de Requerimientos.
Responsable: Dra. Kathleen Fisher.

Grupo: Dirección de Desarrollo en Informática, UPAEP

Actividad: Análisis, Diseño e Implementación de la Interfaz Gráfica de Usuario.

Responsable: Dr. Jaime Castillo.

Grupo: Facultad de Ciencias, UABC.

Actividad: Análisis, Diseño e Implementación de la Máquina de Representación de Redes Semánticas

Responsable: M.C. Alberto L. Morán y Solares

Siendo en este último punto la aportación principal de este trabajo.

2.1. Objetivos

2.1.1. Objetivo General

Desarrollar un producto de software específico que permita realizar la representación del conocimiento utilizando redes semánticas de acuerdo a los siguientes requerimientos especificados por el cliente SDSU.

- Inclusión de características y funcionalidad existente en SemNet versión 1.1.4.
- Inclusión de características nuevas.
  - Gráficos: mejorar el estilo de las gráficas.
  - Búsqueda: permitir búsqueda en toda la red semántica y poder remplazar palabras.
  - Capacidad de multimedia: capacidad de poder agregar sinónimos, texto, sonido e imágenes a conceptos y relaciones.
  - Pantalla inicial: mostrar un ejemplo de una red semántica.
  - Rediseño de la Interfaz Gráfica del Usuario.

2.1.2. Objetivos Específicos

Que el producto de software generado sea portable a diferentes plataformas de cómputo (Windows, Unix y Macintosh) para su utilización.

Que el producto de software desarrollado permita trabajar tanto en el idioma español como en el idioma inglés.

3. METODOLOGÍA

Para el desarrollo del trabajo se utilizó la técnica de modelado por objetos (OMT), la cual apoya a todo el ciclo de vida del desarrollo tal como lo propone la ingeniería del software [Rumbaugh *et al*, 1991].

Basándonos en ciclo de vida de la ingeniería del software bajo el modelo de desarrollo por versiones sucesivas [Pressman, 1992], se realizó lo siguiente:

1ª versión.

- Investigación documental sobre redes semánticas.
- Revisión del documento de Especificación Inicial de Requerimientos.
- Revisión de la funcionalidad del sistema SemNet versión 1.1.14 (disponible en Internet) para Macintosh con el objetivo de familiarizarnos con el sistema y entender mejor el problema.

2da versión.

- Revisión del código java de la versión preliminar, el cual fue desarrollado por L.C.C. José Manuel Alba Álvarez bajo la supervisión del Dr. Roberto Pozos.
- Elaboración de un documento de Especificación Funcional para esta versión del producto de software.
- Documentación de análisis cubriendo hasta la arquitectura del sistema, utilizando la metodología OMT.

3er versión.

- Elaboración del documento de Especificación de Requerimientos para la tercera versión.
- Elaboración del documento de Especificación Funcional de la tercera versión.
- Desarrollo del análisis, diseño e implantación. En esta etapa se obtuvo el análisis definitivo, resultante de la adaptación de versiones anteriores el cual fue la base para continuar el desarrollo.

4. DESARROLLO

4.1. Análisis

Para esta etapa del desarrollo del sistema se comenzó la investigación documental sobre redes semánticas con la finalidad de obtener una mejor visión del problema. A continuación se describen los conceptos básicos.

4.1.1. Conceptos básicos sobre redes semánticas

Las redes semánticas pueden entenderse como mapas de información, ellas nos ayudan a organizar las ideas conforme son aprendidas. Las piezas de información usadas para crear un mapa de información son llamados conceptos y relaciones [Bratko, 1990].

4.1.1.1. Conceptos

Un concepto es una idea que puede ser descrita por palabras o frases. Los conceptos aparecen en óvalos y rectángulos redondeados. Un **concepto** llamado el **concepto central** es la idea central que empieza a describirse. Todos los **conceptos** que son ligados a un concepto central son llamados **conceptos relacionados**. Las **relaciones,** que aparecen como flechas, describen los enlaces entre dos **conceptos** [Elaine, 1991].

En la figura 4.1 se muestra una red semántica, la red es leída desde el concepto central hacia la relación y finalmente al concepto relacionado.

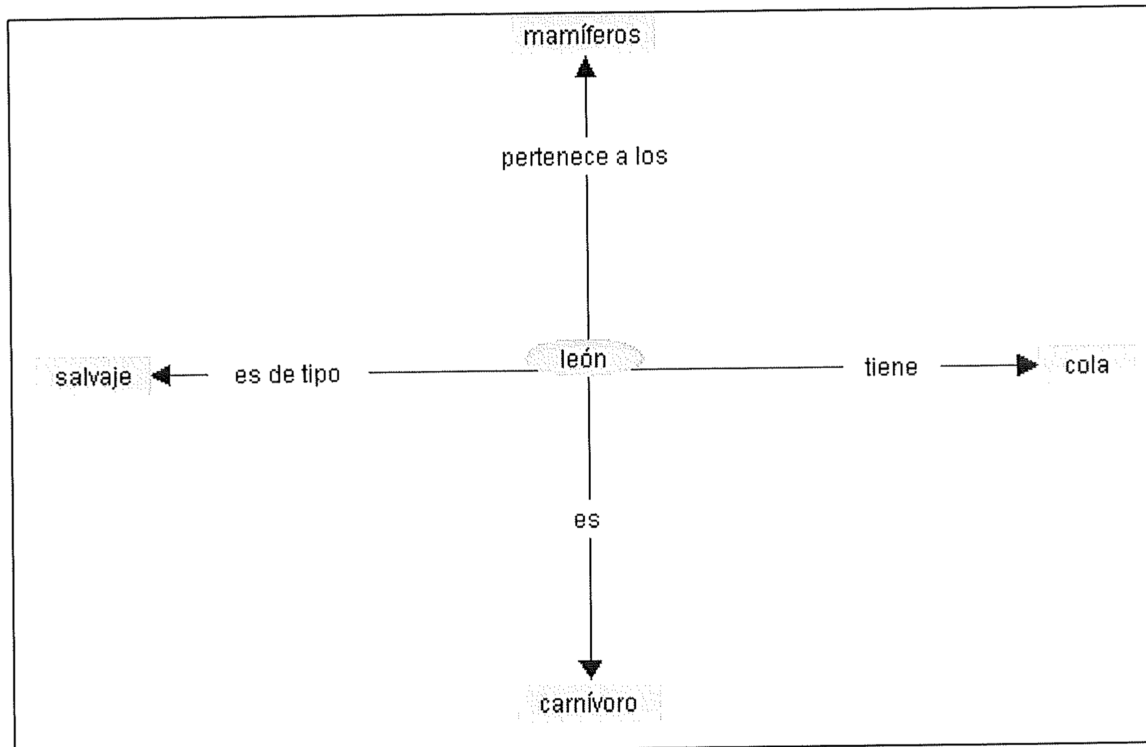


Figura 4.1. Ejemplo de una red semántica

La red de la figura anterior se lee como sigue:

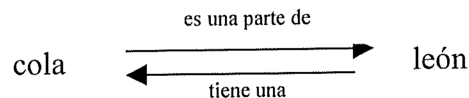
- El león pertenece a la familia de mamíferos
- El león es del tipo salvaje
- El león tiene cola
- El león es carnívoro

4.1.1.2. Relaciones

Una relación consiste de una palabra o frase que describe una conexión entre dos conceptos. Existen dos tipos de relaciones: las relaciones asimétricas y las relaciones simétricas [Elaine, 1991].

4.1.1.2.1. Relaciones Asimétricas

Las relaciones asimétricas necesitan una palabra diferente o frase para describir la relación propia en cada dirección. Por ejemplo la relación mostrada a continuación [Elaine, 1991].



La cual se puede describir como:

El león ***tiene como parte*** la cola

La cola **es *una parte del*** león

Tiene como parte*, y *es una parte del forman una sola relación entre león y cola, y puede ser escrita **león *tiene como parte / es parte de* cola**. Esto es llamado relación asimétrica porque diferentes palabras son usadas para describir la relación en cada dirección.

4.1.1.2.2. Relaciones Simétricas

Las relaciones simétricas son representadas por aquellas palabras que pueden ser usadas para describir la relación en ambas direcciones. Por ejemplo.

Cabeza ***está concectada*** al Cuello

Cuello ***está conectada*** a la Cabeza

4.1.1.3. Instancias

Una instancia es una unidad ***concepto - relación - concepto***. Una instancia incluye ambos sentidos de su relación. La figura 4.2. muestra las maneras de representar las instancias [Bratko, 1990].

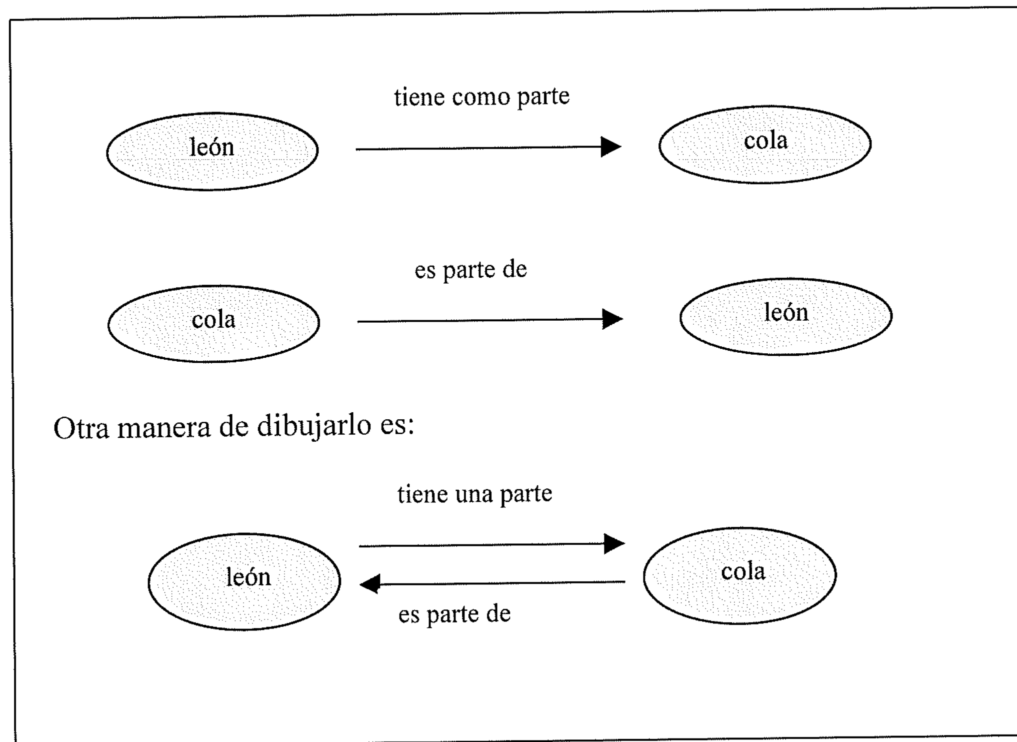


Figura 4.2. Formas de representar las instancias

4.1.1.4. Reglas para la creación de redes semánticas

- Cada relación (enlace) puede ser usada una y otra vez.
- Cada concepto o nodo puede ser ligado a algunos otros conceptos usando varias relaciones.

- No se pueden tener dos nombres (conceptos) iguales en una red. En caso de que esto llegue a ocurrir deben utilizarse sobrenombres o algún otro significado para hacer los nombres diferentes unos de otros.
- No debe borrarse una relación o concepto sin antes estar bien seguro de lo que se va a hacer, por que cuando se borra una relación o un concepto, éste se elimina en toda la red, aún si la relación está siendo usada 75 veces o si el concepto ha estado conectado a otros 20 conceptos, afectando a toda la red.
- Se puede borrar una instancia inmediatamente, como una instancia es simplemente una conexión entre dos conceptos. Borrar una instancia sólo afecta a dos conceptos en la red.
- La mínima red que puede ser representada es: ***concepto - relación - concepto***, es decir una instancia.

4.1.2. Requerimientos del sistema

La especificación de requerimientos fue elaborada con apoyo del grupo de trabajo de SDSU. A continuación describimos los requerimientos pertenecientes a la tercera versión.

Las características del sistema serán las siguientes:

- El sistema deberá ejecutarse en diferentes plataformas de cómputo.
- El sistema debe de incluir las características y funcionalidades de SemNet ver 1.1.4c.
- El sistema debe de incluir características nuevas:

- Mejorar la apariencia, así como invitar al usuario a entrar a la red y asistir al usuario en la construcción y revisión de las redes.
- Mejorar el estilo de las gráficas.
- Búsqueda. Implementar el comando *Encontrar* que trabaje en todas las partes de la red y el comando *Encontrar y Remplazar* que trabaje también sobre toda la red semántica.
- Capacidad de multimedia. Se refiere a los elementos que pueden ser insertados a los conceptos, las relaciones y las instancias. Permitir ahora agregaciones a conceptos y relaciones y diseñar para la inclusión futura de agregaciones a las instancias.
- Los elementos agregados deberán ser fotos, sonidos, sinónimos y URL's.
- Rediseño de la interfaz gráfica del usuario (GUI).
- Todos los diálogos deben de ser móviles y poder variar el tamaño del diálogo.
- Se deben de soportar nombres de longitud ilimitada para conceptos y relaciones.
- Permitir diferentes estilos de letra, tamaño y color del texto.

4.1.3. Modelo de Objetos

La figura 4.3 muestra el modelo de objetos inicial. Este diagrama describe las clases para la máquina de representación del conocimiento. La clase ***Red Semántica*** es

la encargada de manipular todos los objetos de la red. Red Semántica está formada por cero o muchas Instancias, dos conceptos y una relación son parte de una Instancia.

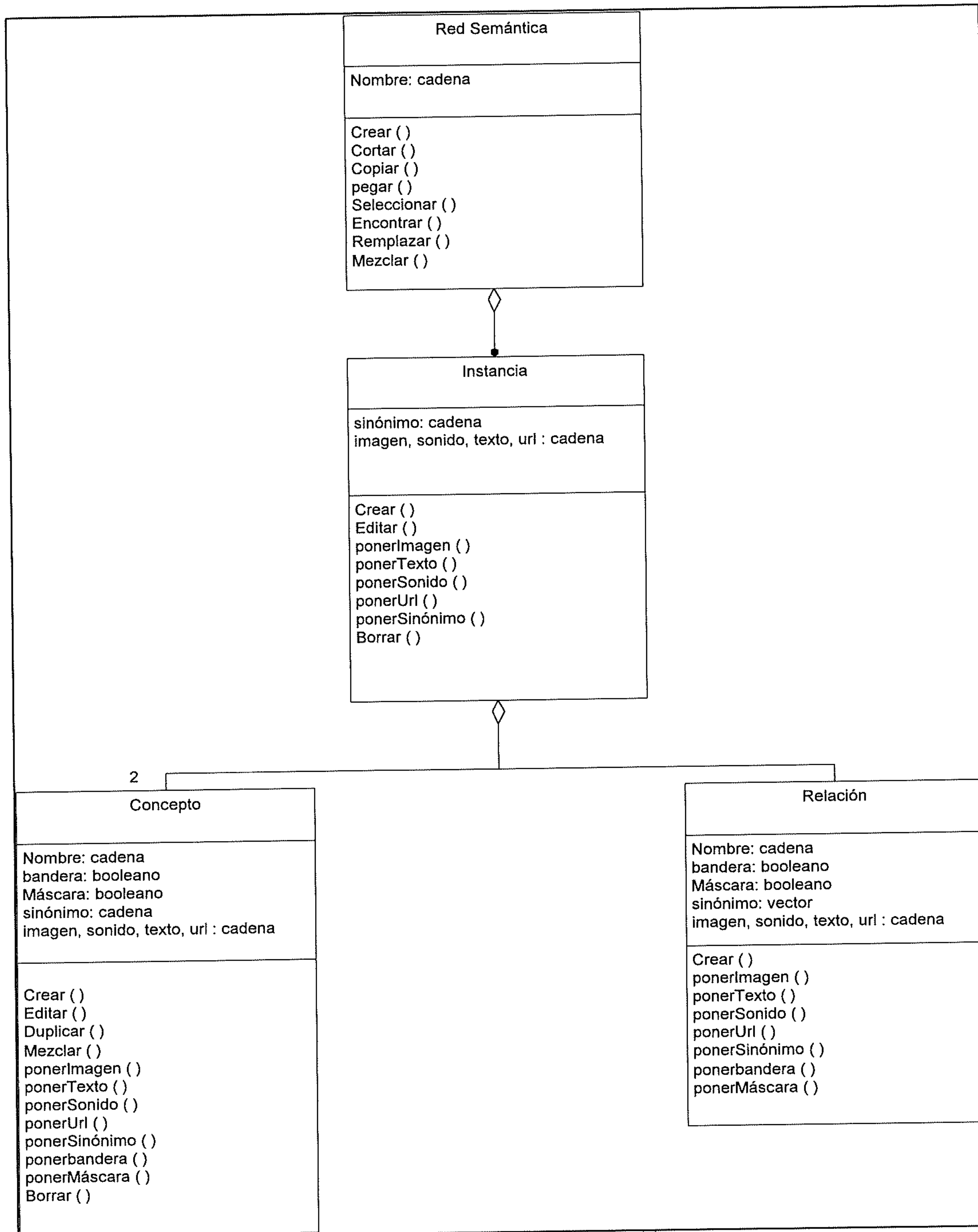


Figura 4.3. Modelo de Objetos

4.1.3.1. Diccionario de Datos

A continuación se describen los métodos de las clases referentes al modelo de objetos descrito anteriormente.

4.1.3.1.1. Red Semántica.

Esta clase contiene todas las propiedades de la red semántica, conceptos, relaciones e instancias.

Atributos:

Nombre. Nombre de la red semántica. Tipo: *cadena.*

Operaciones:

Crear( ). Crea una red semántica sin nombre.

Cortar( ). Corta la red semántica o una parte de la red semántica.

Copiar( ). Copia una red semántica o lo seleccionado.

Pegar ( ). Este método copia la información que fue seleccionada.

Seleccionar( ). Selecciona una red semántica o una parte de esta.

Encontrar( ). Método que búsca una palabra en toda la red.

Reemplazar ( ). Método que encuentra y reemplaza el nombre de una palabra en toda la red semántica.

Mezclar ( ). Método que mezcla una red semántica con otra.

4.1.3.1.2. Instancia.

La instancia se representa mediante un ***concepto – relación – concepto***, donde un concepto es llamado concepto central y el otro concepto relacionado.

Atributos:

Sinónimo. Secuencia de cadenas que conforman el sinónimo para la instancia. *Tipo: cadena.*

Imagen. Imagen agregada a la instancia. Tipo: *cadena.*

Sonido. Sonido agregado a la instancia. Tipo: *cadena.*

Texto. Texto agregado a la instancia. Tipo: *cadena.*

Url. Enlace agregado a la instancia. Tipo: *cadena.*

Operaciones:

Crear( ). Crea una nueva instancia.

Editar( ). Permite cambiar las características de la instancia.

PonerImagen ( ). Método que permite agregar una imagen a la instancia.

PonerTexto ( ). Método que permite agregar texto a la instancia.

PonerSonido ( ). Método que permite agregar sonido a la instancia.

PonerUrl ( ). Método que permite agregar una liga a la instancia.

PonerSinónimo ( ). Método que permite agregar sinónimos a la instancia

Borrar( ). Método que elimina la instancia.

4.1.3.1.3. Concepto.

Un concepto en una red semántica representa objetos, eventos y situaciones del mundo real.

Atributos:

Nombre. Nombre del concepto. Tipo: *cadena.*

Bandera. Marca para el concepto elaborado por el usuario. Tipo: *booleano.*

Máscara. Máscara para el concepto elaborado por el usuario. Tipo: *booleano.*

Sinónimo. Secuencia de cadenas que conforman el sinónimo para el concepto. Tipo: *cadena.*

Imagen. Imagen agregada al concepto. Tipo: cadena.

Texto. Texto agregado al concepto. Tipo: *cadena.*

Sonido. Sonido agregado al concepto. Tipo: *cadena.*

Url. Enlace agregado al concepto. Tipo: cadena.

Operaciones:

Crear( ). Método para crear un nuevo concepto.

Editar( ). Permite cambiar las características del concepto.

Duplicar ( ). Método que permite duplicar un concepto.

Mezclar( ). Método que mezcla con otro concepto.

PonerImagen ( ). Método que permite agregar una imagen al concepto.

PonerTexto ( ). Método que permite agregar texto a un concepto.

PonerSonido ( ). Método que permite agregar sonido a un concepto.

PonerUrl ( ). Método que permite agregar una liga al concepto.

PonerSinónimo (). Método que permite agregar un sinónimos al concepto.

PonerBandera (). Método que permite poner una marca al concepto.

AgregarMáscara (). Método que permite poner una máscara al concepto.

Borrar(). Método que elimina un concepto.

4.1.3.1.4. Relación.

Una relación en una red semántica es un atributo que une a otro concepto.

Atributos:

Nombre: Nombre de la relación. Tipo: *cadena,*

Bandera. Marca para la relación.Tipo: *booleano.*

Máscara. Máscara para la relación. Tipo: *booleano*

Sinónimo. Sinónimo agregado a la relación. Tipo: *vector.*

Sonido. Sonido agregado al concepto. Tipo: *cadena.*

Imagen. Imagen agregada al concepto. Tipo: *cadena.*

Texto. Texto agregado al concepto. Tipo: *cadena.*

Url. Enlace agregado al concepto. Tipo: *cadena.*

Operaciones:

Crear (). Método que permite crear una nueva flecha.

PonerImagen (). Método que permite agregar una imagen a la relación.

PonerTexto (). Método que permite agregar texto a la relación.

PonerSonido (). Método que permite agregar sonido a la relación.

PonerUrl () Método que permite agregar un enlace a la relación.

PonerSinónimo (). Método que permite agregar sinónimo a la relación.

PonerBandera (). Método que permite agregar una marca a la relación.

PonerMáscara (). Método que permite agregar máscara a la relación.

4.1.4. Modelo Dinámico

En esta fase del desarrollo se describen los aspectos del sistema que cambian conforme pasa el tiempo. Se describen algunos escenarios, diagramas de trazo de eventos y diagramas de estados para cada clase.

4.1.4.1. Escenarios para la clase Red Semántica

1. El usuario inicia el sistema.
2. El sistema despliega el menú de opciones.
3. El usuario selecciona crear una **nueva** red semántica.
4. El sistema solicita crear la red semántica.
5. Red Semántica comienza a crear la red semántica.

6. El usuario selecciona **mezclar** una red semántica con otra.
7. El sistema selecciona la red semántica.
8. Red Semántica solicita los datos de los conceptos.

9. Los conceptos regresan datos.
10. Red Semántica solicita los datos de las relaciones.
11. Las relaciones regresan los datos.

12. El usuario selecciona **Encontrar** una palabra.
13. El usuario especifica qué palabra.
14. El sistema busca la palabra en Red Semántica.
15. El sistema notifica el resultado.

16. El usuario selecciona **reemplazar** una palabra.
17. El usuario especifica la palabra.
18. El sistema encuentra la palabra en Red Semántica y la reemplaza.
19. El sistema notifica el resultado.

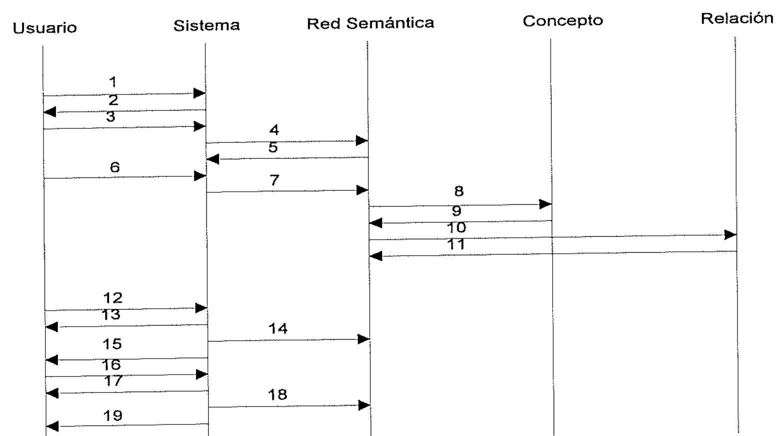


Figura 4.4. Diagrama de trazo de eventos de la clase Red Semántica.

4.1.4.2. Escenarios para la clase Concepto

1. El usuario selecciona **crear** un concepto.
2. Sistema crea un nuevo concepto y asigna el número en el que fue creado.

3. El usuario selecciona **editar** concepto.
4. Sistema busca el concepto seleccionado.
5. El concepto es encontrado.
6. Sistema modifica las características del concepto

7. El usuario selecciona **borrar** concepto.
8. Sistema valida la acción.
9. Sistema busca todas las instancias donde aparece el concepto.
10. Red Semántica regresa todas las instancias donde el concepto fue encontrado.
11. El sistema modifica el número de instancias del concepto.
12. Sistema modifica los conceptos relacionados.
13. Sistema modifica el número de instancias en la relación.
14. Sistema borra todas las instancias donde el concepto es encontrado.
15. Sistema borra el concepto.

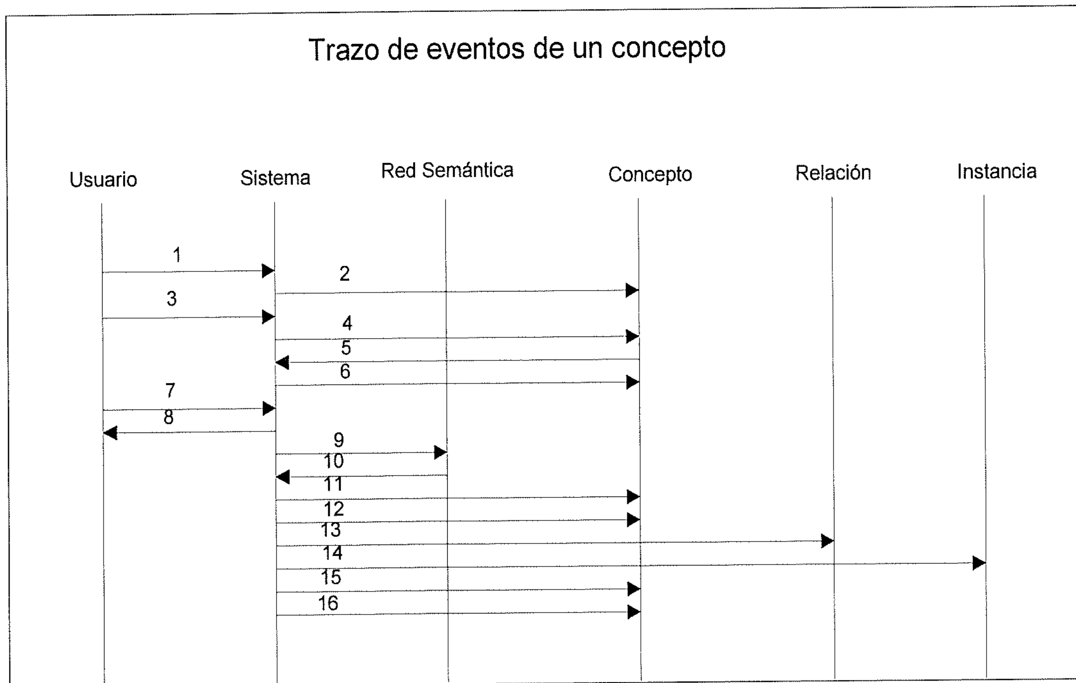


Figura 4.5. Diagrama de trazo de eventos de la clase Concepto.

4.1.4.3. Escenario para la clase Relación

1. El usuario selecciona **crear** una relación.
2. Sistema crea una nueva relación y asigna el número en el que fue creado.

3. El usuario selecciona **editar** relación.
4. Sistema busca la relación seleccionada.
5. La relación es encontrada.
6. Sistema modifica las características de la relación

7. El usuario selecciona **borrar** la relación.
8. Sistema valida la acción.

9. Sistema busca todas las instancias donde aparece la relación
10. Red Semántica regresa todas las instancias donde la relación fue encontrada.
11. Sistema modifica el número de instancias en la relación.
12. Sistema borra las instancias.
13. Sistema borra la relación.
14. Sistema ordena el número de relaciones creadas.

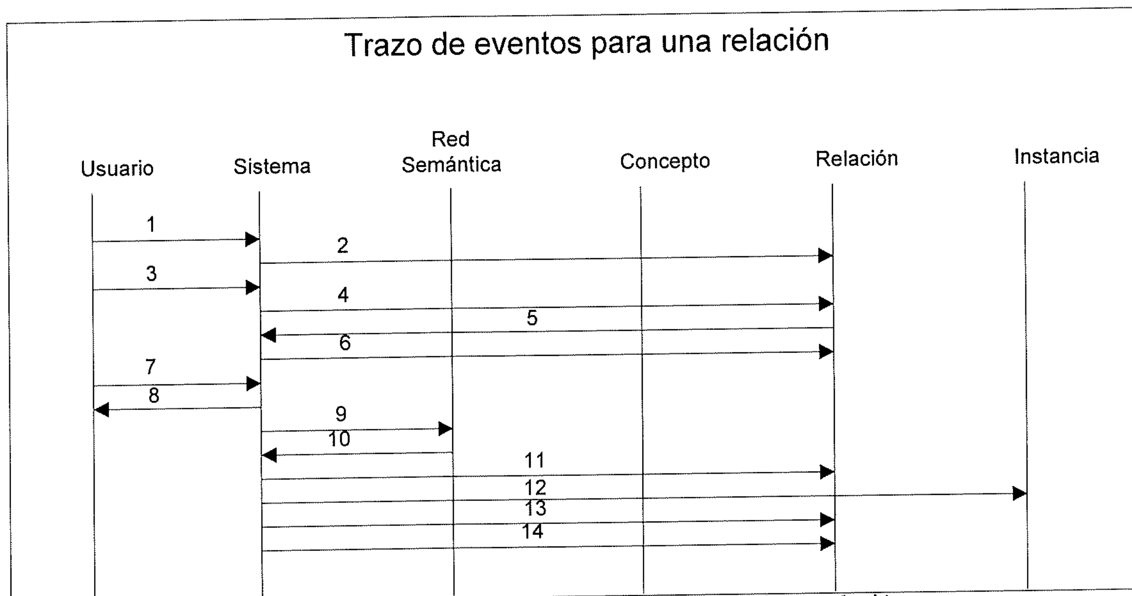


Figura 4.6. Diagrama de trazo de eventos de la clase Relación.

4.1.4.4. Escenario para la clase Instancia

1. El usuario selecciona **crear** instancia.
2. Sistema lista la relación en Red Semántica.
3. Sistema asigna el orden en que fue creada la instancia

4. Sistema agrega la instancia en Red Semántica.

5. El usuario selecciona **editar** instancia
6. Sistema busca la instancia seleccionada.
7. La instancia es encontrada.
8. Instancia regresa sus atributos.
9. Sistema modifica las características de la instancia.

10. El usuario selecciona **borrar** la instancia.
11. Sistema valida la acción.
12. Sistema modifica el número de instancias y conceptos relacionados con la instancia.
13. Sistema modifica el número de instancias en las relaciones.
14. Sistema borra las instancias.

Figura 4.7. Diagrama de trazo de eventos de la clase Instancia.

4.1.4.5. Diagrama general de eventos

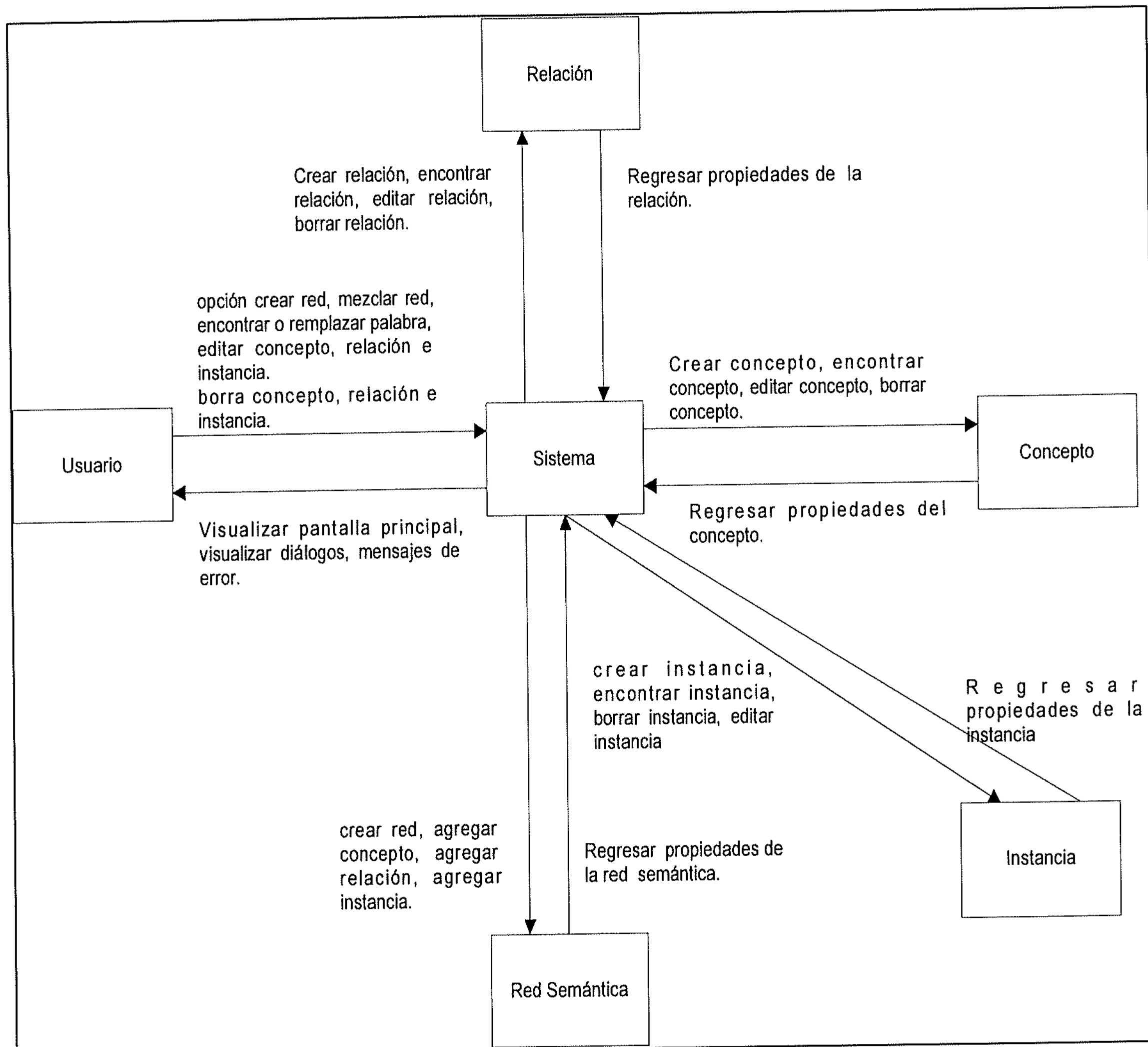


Figura 4.8. Diagrama general de eventos.

4.1.4.6. Diagrama de estados

En esta etapa se muestran los diagramas de estados para cada clase, los cuales describen el comportamiento dinámico como son los eventos y acciones.

4.1.4.6.1. Diagrama de estados para la clase Concepto

En los siguientes diagramas se presentan los estados principales en que puede estar un Concepto o una Relación: No Instanciado e Instanciado. A su vez, un concepto puede estar siendo Creado / inicializado, No seleccionado o seleccionado y finalmente dentro de este último, puede estar en Espera, en Edición o siendo Borrado.

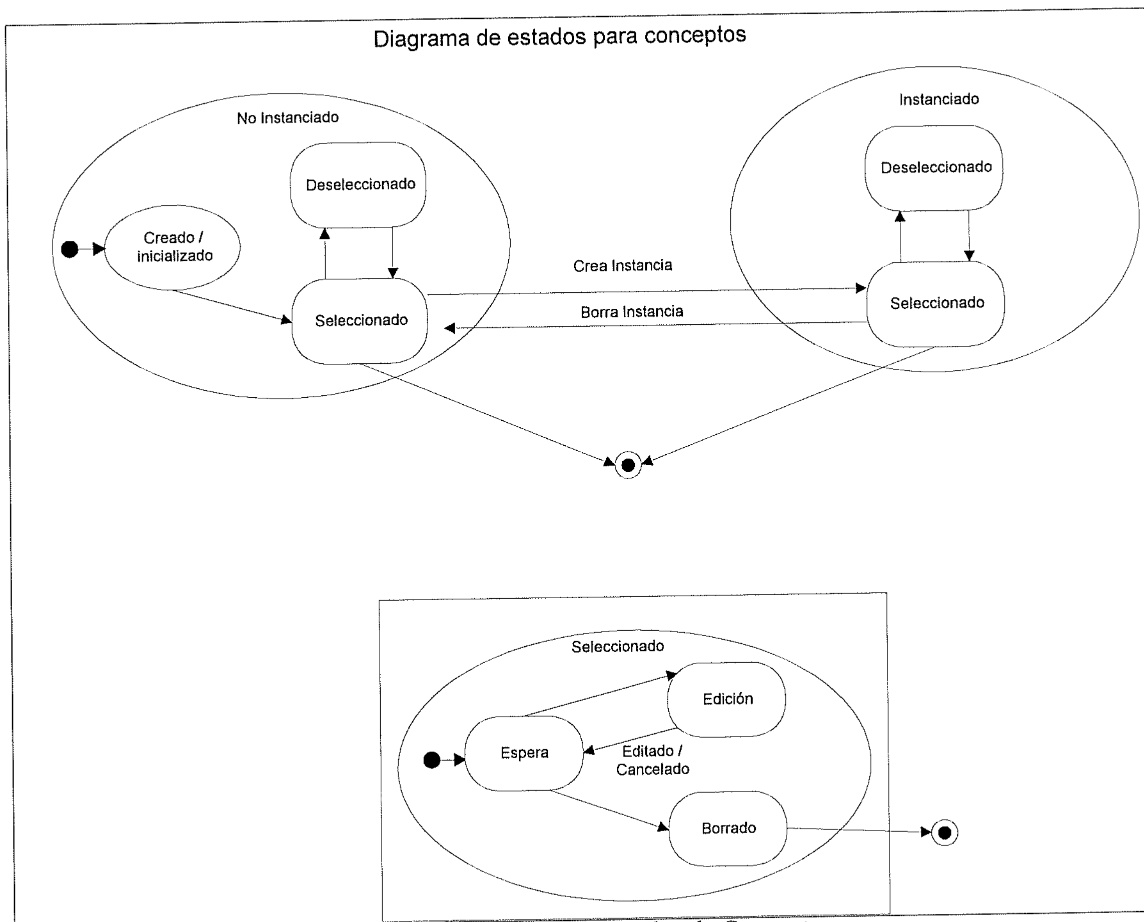


Figura 4.9. Diagrama de estados de Concepto.

4.1.4.6.2. Diagrama de estados para la clase Relación

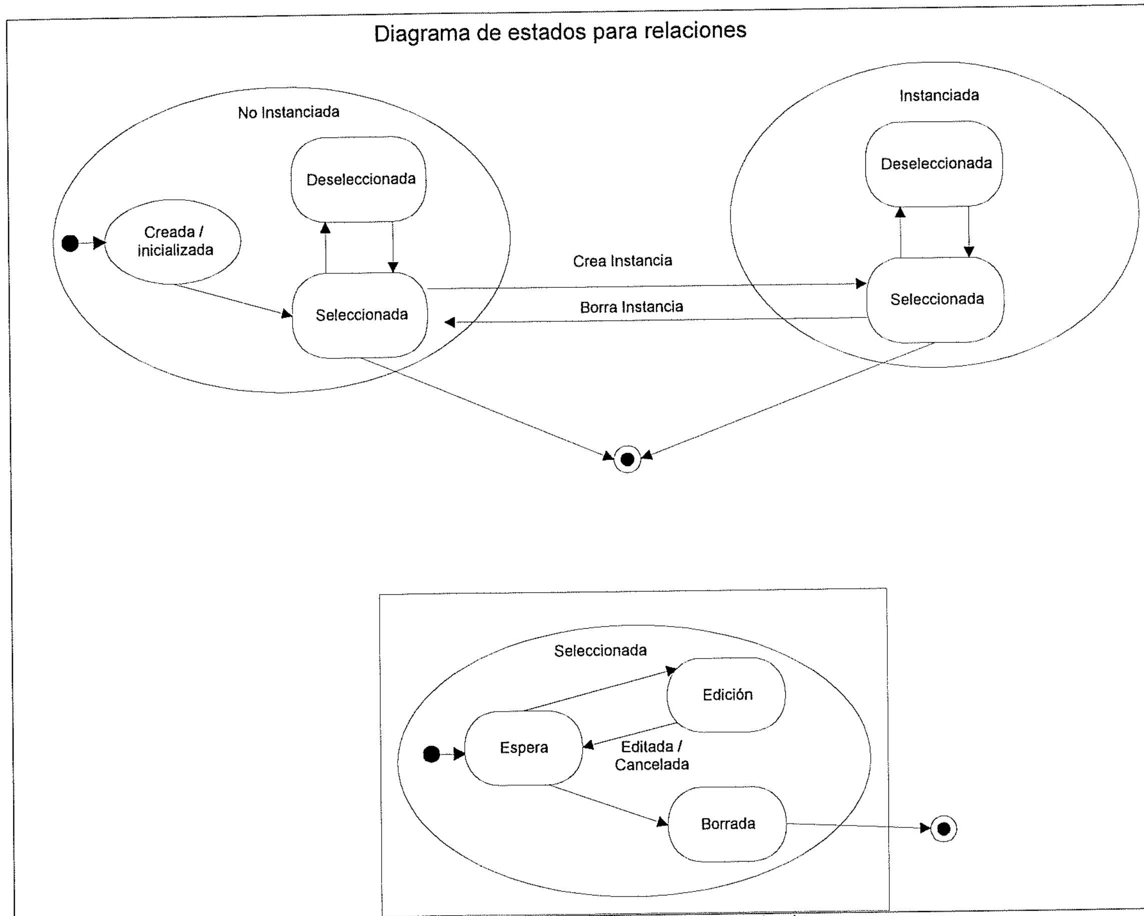


Figura 4.10. Diagrama de estados de Relación.

La figura 4.10. Muestra el diagrama de estados en los que puede estar una Relación estos son: No Instanciado e Instanciado. La Relación puede estar siendo creada / inicializada, No Seleccionada o Seleccionada y dentro de este último estado la Relación puede estar en Espera, Edición o siendo Borrada.

4.1.4.6.3. Diagrama de estados para la clase Instancia

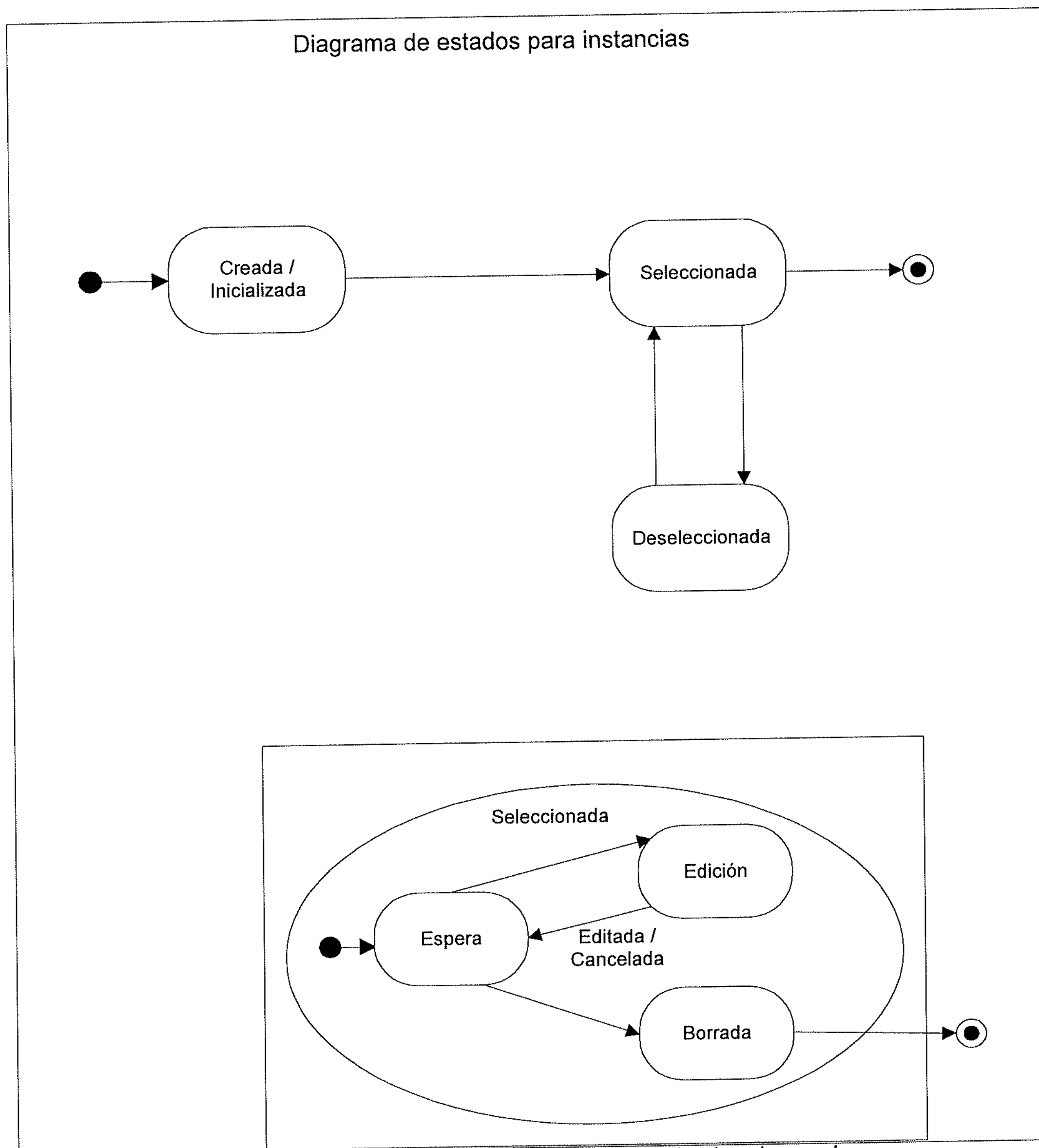


Figura 4.11. Diagrama de estados de la clase instancia.

La figura 4.11. Muestra el diagrama de estados para la clase Instancia, los estados en los que puede encontrarse son: Creada / inicializada, Seleccionada y Deseleccionada. Cuando la instancia se encuentra Seleccionada puede estar en el estado de Espera, Edición o siendo Borrada.

4.1.4.7. Modelo Funcional

En esta fase de la metodología se presentan los diagramas de flujo de datos para cada clase, describiendo el flujo de información de entrada y salida para cada proceso.

4.1.4.7.1. Diagrama de Flujo de Datos para la clase Red Semántica

La figura 4.12 muestra el diagrama de flujo de datos de la clase Red Semántica, dicha clase es de las más importantes ya que es la encargada de contener todos los objetos de la red como son conceptos, relaciones e instancias.

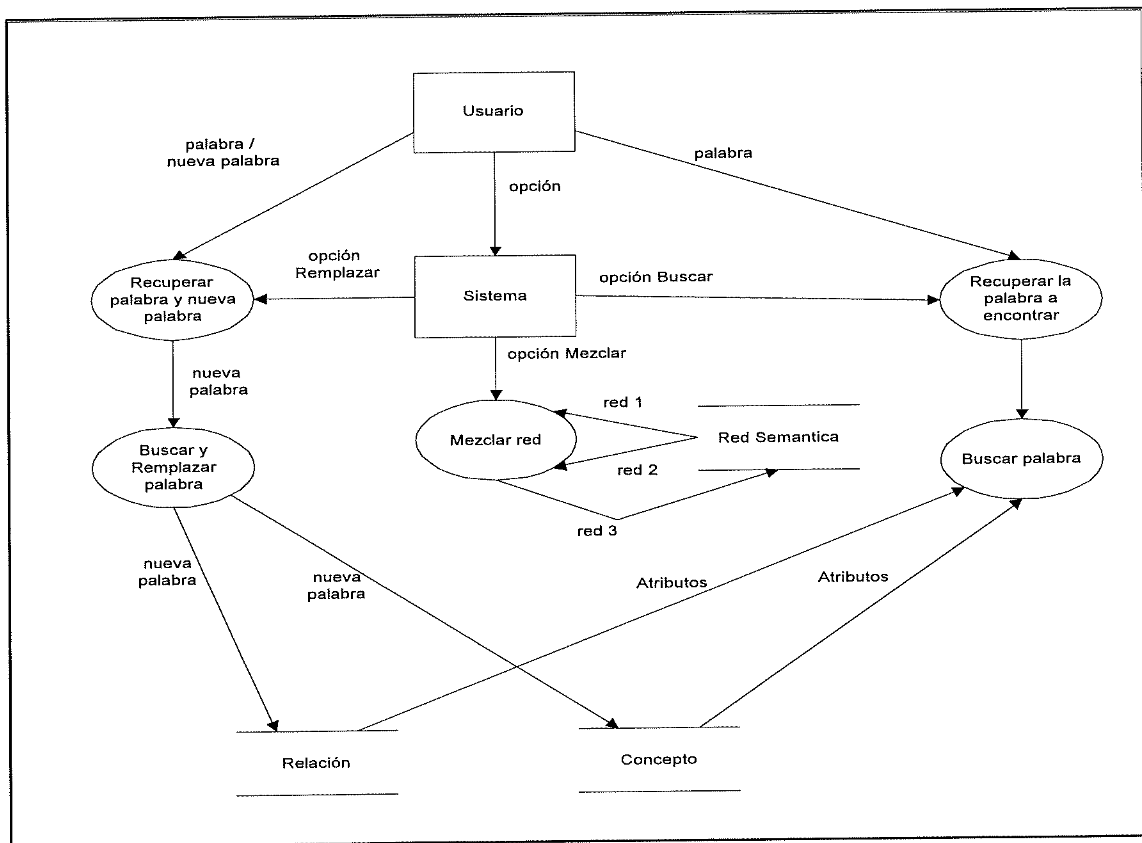


Figura 4.12. Diagrama de Flujo de Datos para la clase Red Semántica

4.1.4.7.2. Diagrama de Flujo de Datos para la clase Concepto

En la figura 4.13 se muestra la información de entrada y salida para cada proceso de la clase concepto.

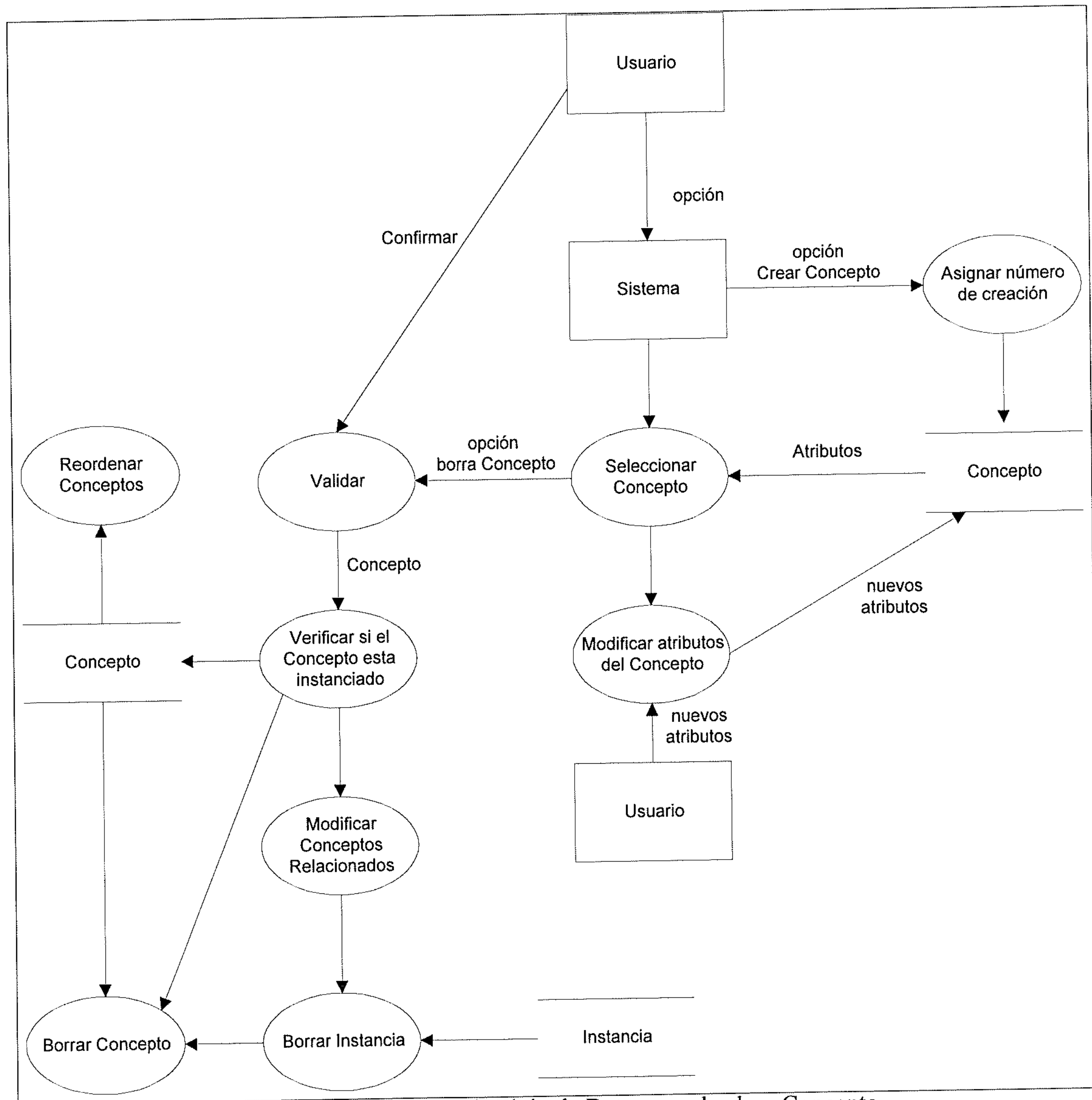


Figura 4.13. Diagrama de Flujo de Datos para la clase Concepto.

4.1.4.7.3. Diagrama de Flujo de Datos para la clase Relación

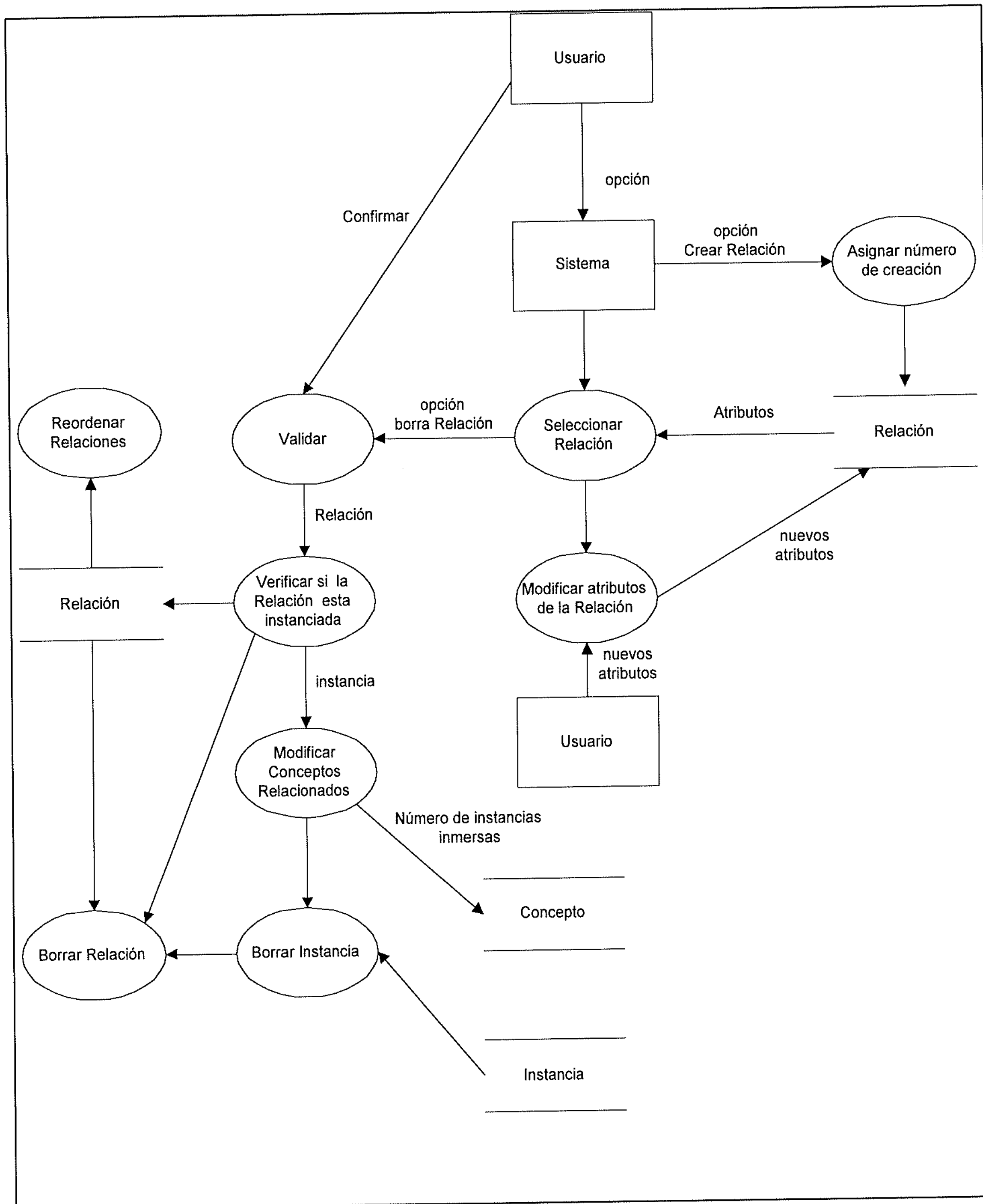


Figura 4.14. Diagrama de Flujo de Datos para la clase Relación

4.1.4.7.4. Diagrama de Flujo de Datos para la clase Instancia

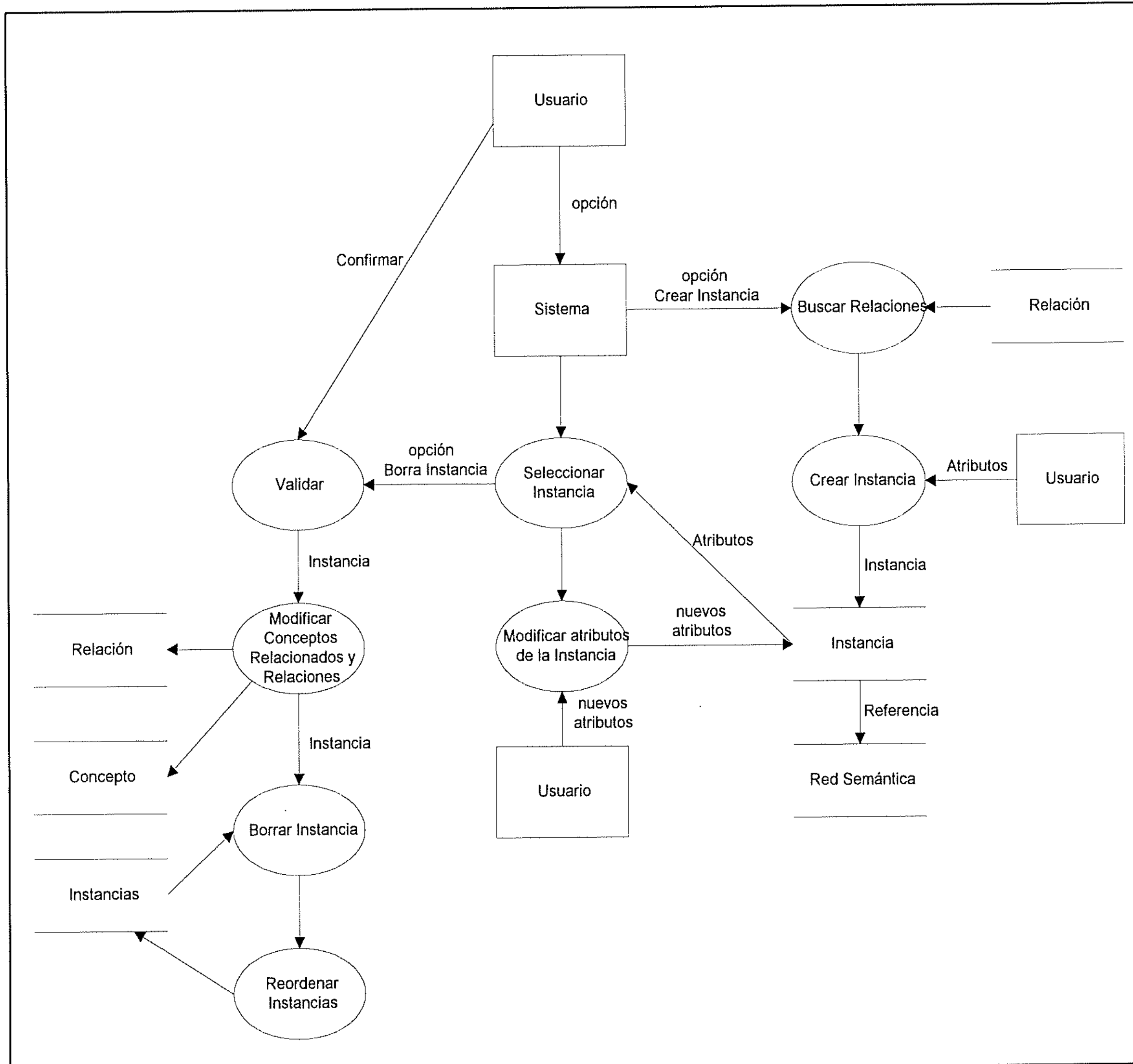


Figura 4.15. Diagrama de Flujo de Datos para la clase Instancia.

4.2. Diseño de Sistemas

El diseño de sistemas es la estrategia de alto nivel para resolver el problema y construir una solución. Incluye decisiones acerca de la organización del sistema así como componentes de hardware y software.

4.2.1. Arquitectura del sistema

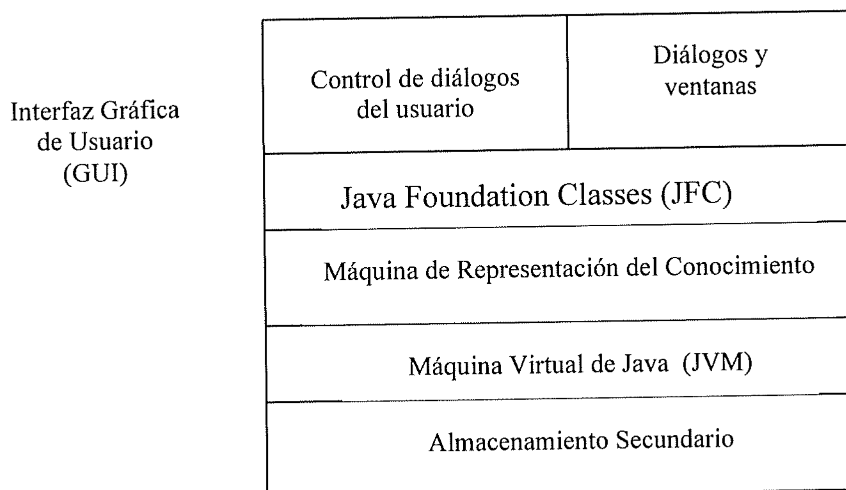


Figura 4.16. Arquitectura del sistema.

La arquitectura del sistema contiene los siguientes subsistemas:

Interfaz Gráfica del Usuario (GUI). Este subsistema está compuesto de dos capas, una de ellas dividida en dos particiones, la partición ***control de diálogos del usuario*** permite controlar los diálogos con los que el usuario interactúa. La partición ***Diálogos y ventanas*** se encarga de mantener los diálogos y ventanas de la interfaz gráfica. ***Java Foundation Classes (JFC)*** desarrollado por SUN Mycrosystem, este subsistema es el encargado de manejar los diálogos y ventanas así como el control de diálogos.

Máquina de Representación del Conocimiento. Subsistema que provee la capacidad de representar el conocimiento mediante redes semánticas.

Máquina Virtual de Java. Subsistema encargado de interpretar el sistema en cualquier arquitectura (desarrollado por Sun microsystem).

Almacenamiento Secundario. Subsistema referente al almacenamiento de la red semántica.

4.2.2. Relación entre subsistemas

La relación entre los subsistemas es cliente – proveedor ya que cuando uno necesita algún servicio el otro responde con un resultado. En este caso el subsistema GUI es el cliente y la Máquina de Representación del Conocimiento es el subsistema proveedor. La relación que existe entre estos dos subsistemas se instrumenta a través de mensajes con paso de parámetros necesarios para poder realizar operaciones como desplegar la red semántica en el GUI.

4.2.3. Manejo de Recursos Globales.

Para el manejo de recursos globales se cuenta con dos archivos los cuales contendrán la información necesaria para el manejo de los dos idiomas (inglés y español) en los diálogos y opciones del menú.

4.3. Diseño de objetos

En esta fase, el diseño de objetos, pasa a ser un proceso de adición de detalles y toma decisiones de implementación y consiste en optimizar, refinar y extender los modelos de objetos, dinámico y funcional hasta que sean suficientemente detallados para la implementación [Rumbaugh, *et al* 1991]. Enseguida se describen los modelos de objetos detallado, dinámico detallado y modelo funcional detallado así como la descripción de algoritmos para cada clase.

4.3.1. Modelo de objetos detallado

El modelo de objetos que se muestra en la figura 4.17 incluye dos clases nuevas que son: Nodo y Flechas, estas clases se incluyen con el objetivo de facilitar la implementación. Este modelo describe lo siguiente: la clase Red Semántica está formada por las clases Nodo e Instancias. Nodo es un concepto y a su vez el Nodo puede tener cero o muchas Instancias. Dos Conceptos y una Relación son parte de una Instancia. Y una Relación tiene dos flechas (los atributos y relaciones para cada clase se muestra en la figura 4.18).

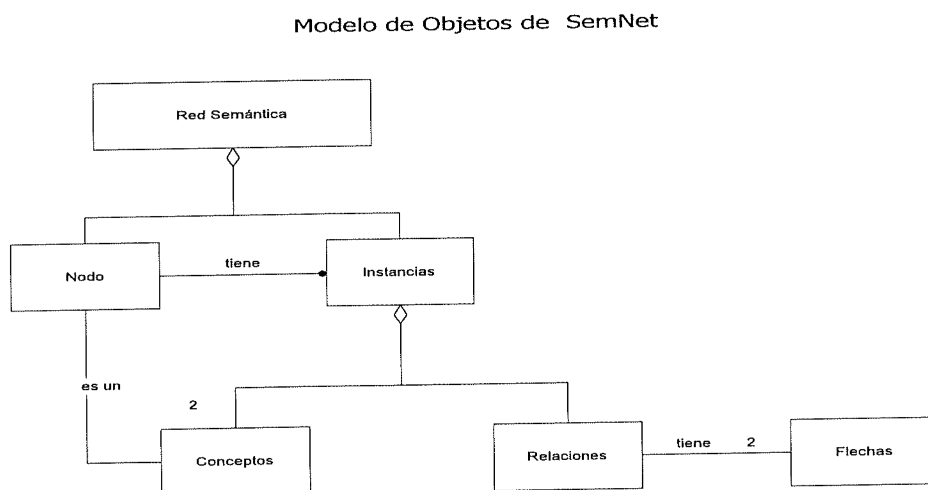


Figura 4.17. Modelo de objetos detallado

4.3.1.1. Nombre, atributos y operaciones del modelo de objetos

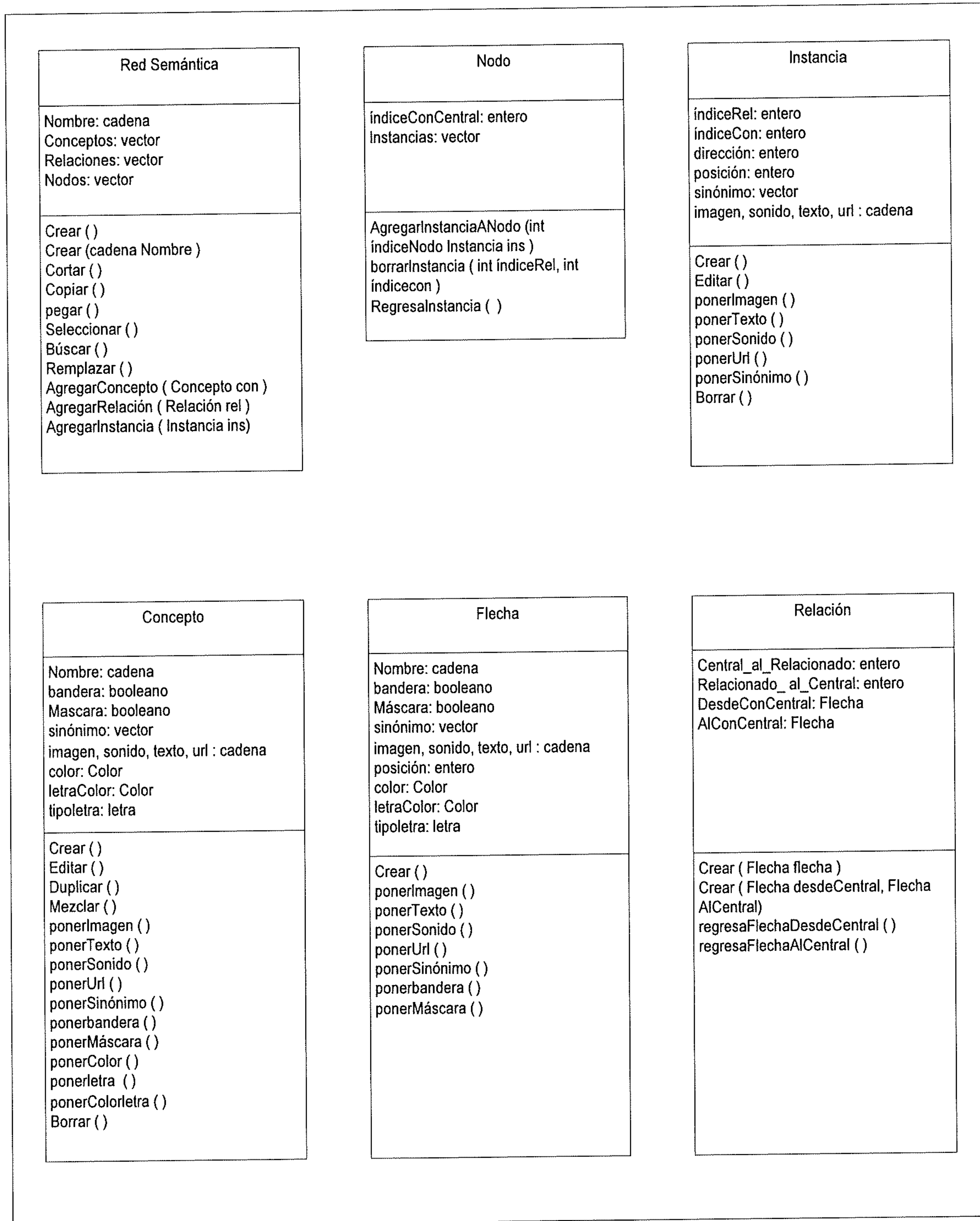


Figura 4.18. Nombre, atributos y operaciones del modelo de objetos

4.3.2. Modelo Dinámico detallado

4.3.2.1.1. Escenarios y trazos de eventos para la clase Concepto

1. El usuario selecciona **crear** concepto.
2. El GUI despliega diálogo para crear concepto.
3. El usuario captura las características del concepto.
4. El GUI crea un nuevo concepto y asigna el número en que fue creado.
5. El concepto es agregado en nodo.

6. El usuario selecciona **editar** concepto.
7. El sistema busca el concepto seleccionado.
8. El concepto es encontrado y regresa sus atributos.
9. El GUI obtiene las características del concepto.
10. El GUI despliega diálogo para editar concepto.
11. El usuario modifica las características del concepto.
12. El GUI actualiza las características del concepto.

13. El usuario selecciona **borrar** concepto.
14. El GUI busca el concepto seleccionado.
15. El concepto es encontrado y regresa sus atributos.
16. El GUI obtiene las características del concepto.
17. El GUI despliega diálogo para borrar concepto.

18. El usuario borra el concepto.

19. El GUI valida la acción.

20. El usuario confirma borrar concepto.

21. El GUI busca todas las instancias donde aparece el concepto.

22. Red Semántica regresa todas las instancias donde el concepto es encontrado.

23. El GUI modifica el número de instancias del concepto.

24. El GUI modifica los conceptos relacionados.

25. El GUI modifica el número de instancias de la relación.

26. El GUI borra todas las instancias donde el concepto fue encontrado.

27. El GUI borra el concepto de nodo.

28. El GUI borra el concepto.

29. El GUI ordena el número de conceptos creados.

La figura 4.19 muestra el Diagrama de trazo de eventos para la clase Concepto.

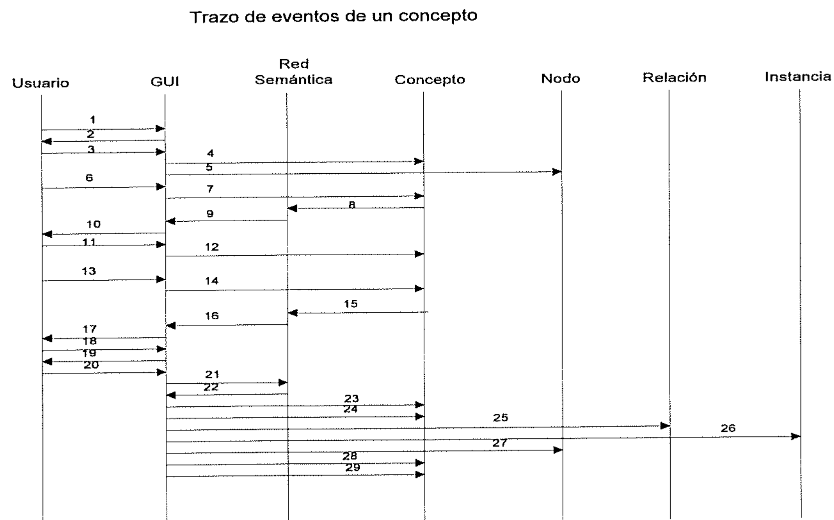


Figura 4.19. Diagrama de trazo de eventos para Concepto

4.3.2.1.2. Escenarios y trazos de eventos para la clase Relación

1. El usuario selecciona **crear** relación.
2. El GUI despliega diálogo para crear relación.
3. El usuario captura las características de la relación.
4. El GUI crea una nueva relación y asigna el número en que fue creado.

5. El usuario selecciona **editar** relación.
6. El GUI busca la relación seleccionada.
7. La relación es encontrada y regresa sus atributos.
8. El GUI obtiene las características de la relación.
9. El GUI despliega diálogo para editar relación.
10. El usuario modifica las características de la relación.
11. El GUI actualiza las características de la relación.

12. El usuario selecciona **borrar** relación.
13. El GUI busca la relación seleccionada.
14. La relación es encontrada y regresa sus atributos.
15. El GUI obtiene las características de la relación.
16. El GUI despliega diálogo para borrar relación.
17. El usuario borra la relación.
18. El GUI valida la acción.

19. El usuario confirma borrar la relación.

20. El GUI busca todas las instancias donde la relación es encontrada.

21. Red Semántica regresa todas las instancias donde la relación fue encontrada.

22. El GUI modifica el número de instancias y atributos del concepto.

23. El GUI borra las instancias.

24. El GUI borra la relación.

25. El GUI ordena el número de relaciones creadas.

La figura 4.20 muestra el diagrama de trazo de eventos para la clase Relación.

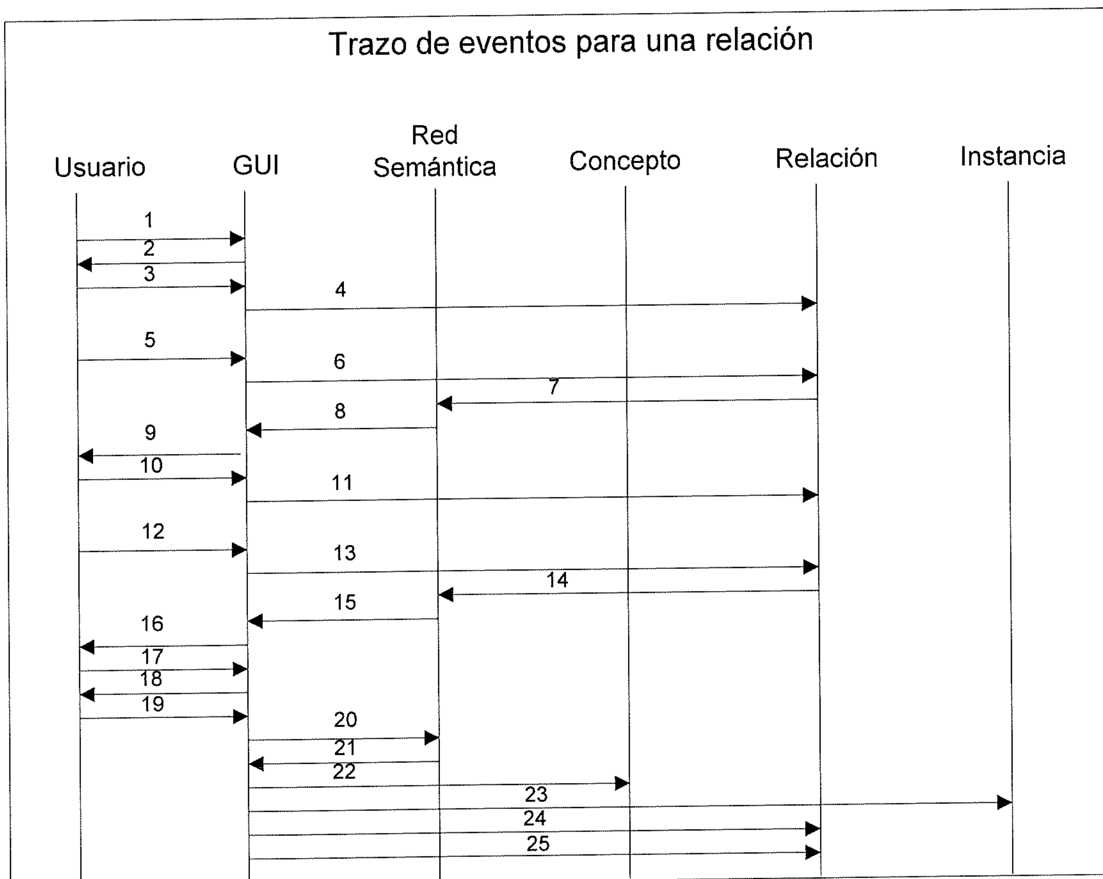


Figura 4.20. Diagrama de trazo de eventos para la clase Relación

4.3.2.1.3. Escenarios y trazos de eventos para la clase Instancia

1. El usuario selecciona **crear** instancia.
2. El GUI despliega diálogo para crear instancia.
3. El usuario captura las características de la instancia.
4. El GUI lista la relación.
5. El GUI asigna el número en que fue creada la instancia.
6. El GUI agrega la instancia en red semántica.
7. Red semántica agrega la instancia en nodo.

8. El usuario selecciona **editar** instancia.
9. El GUI busca la instancia seleccionada.
10. La relación es encontrada y regresa sus atributos.
11. El GUI obtiene las características de la relación.
12. El GUI despliega diálogo para editar relación.
13. El usuario modifica las características de la relación.
14. El GUI actualiza las características de la relación.

15. El usuario selecciona **borrar** instancia.
16. El GUI busca la instancia seleccionada.
17. La instancia es encontrada y regresa sus atributos.
18. El GUI obtiene las características de la instancia.

19. El GUI despliega diálogo para borrar instancia.

20. El usuario borra la instancia.

21. El GUI valida la acción.

22. El usuario confirma borrar instancia.

23. El GUI modifica el número de instancias y conceptos relacionados con la instancia,

24. El GUI modifica el número de instancias en las relaciones.

25. El GUI borra la instancia.

La figura 4.21 muestra el diagrama de trazo de eventos para la clase Instancia

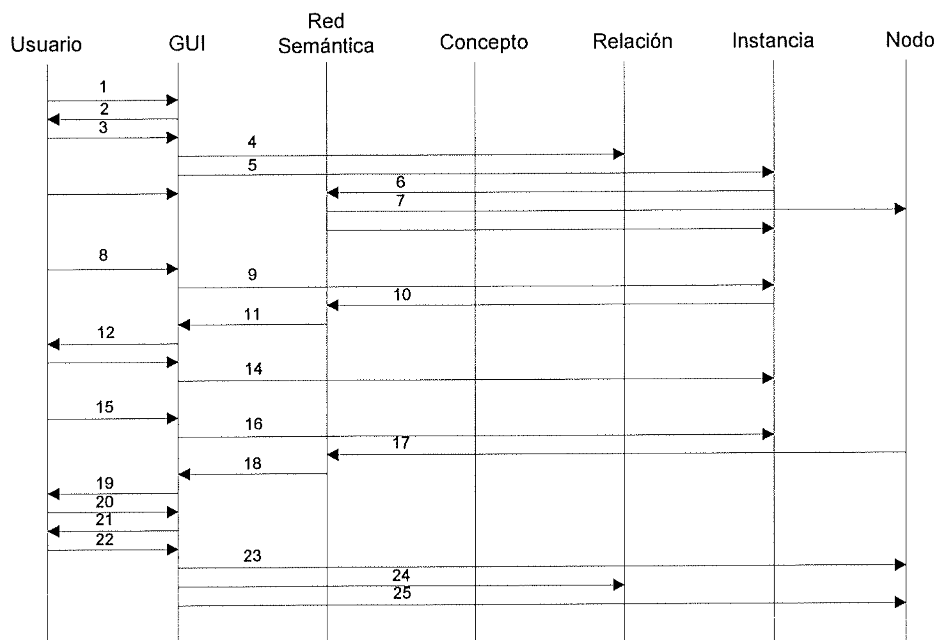


Figura 4.21. Diagrama de trazo de eventos para la clase Instancia

A continuación se muestran los diagramas de estados para las clases: Concepto (figura 4.22), Relación (figura 4.23) e Instancia figura (4.24).

4.3.2.2.1. Diagrama de estados para la clase Concepto

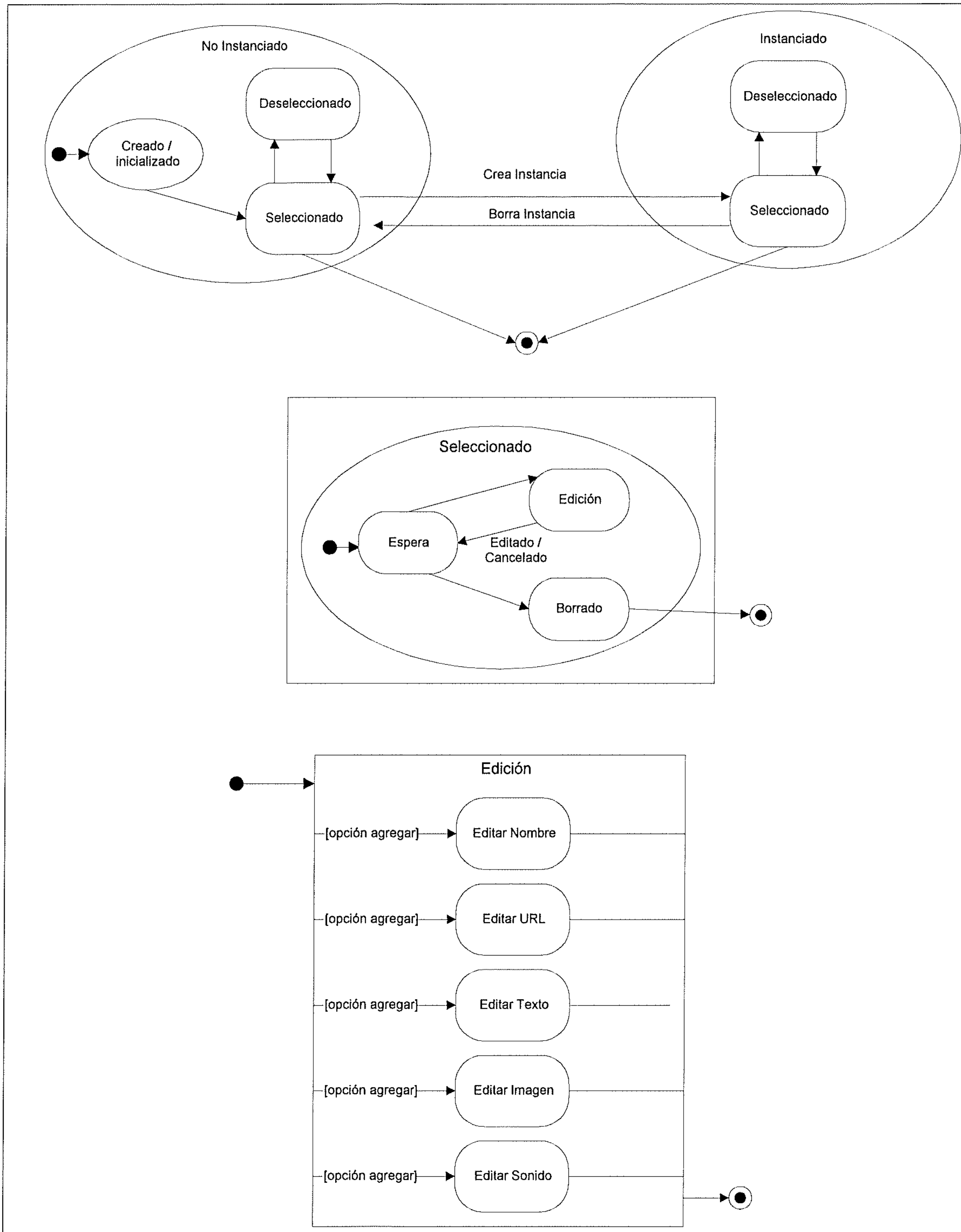


Figura 4.22. Diagrama de estados para la clase Concepto

4.3.2.2.2. Diagrama de estados para la clase Relación

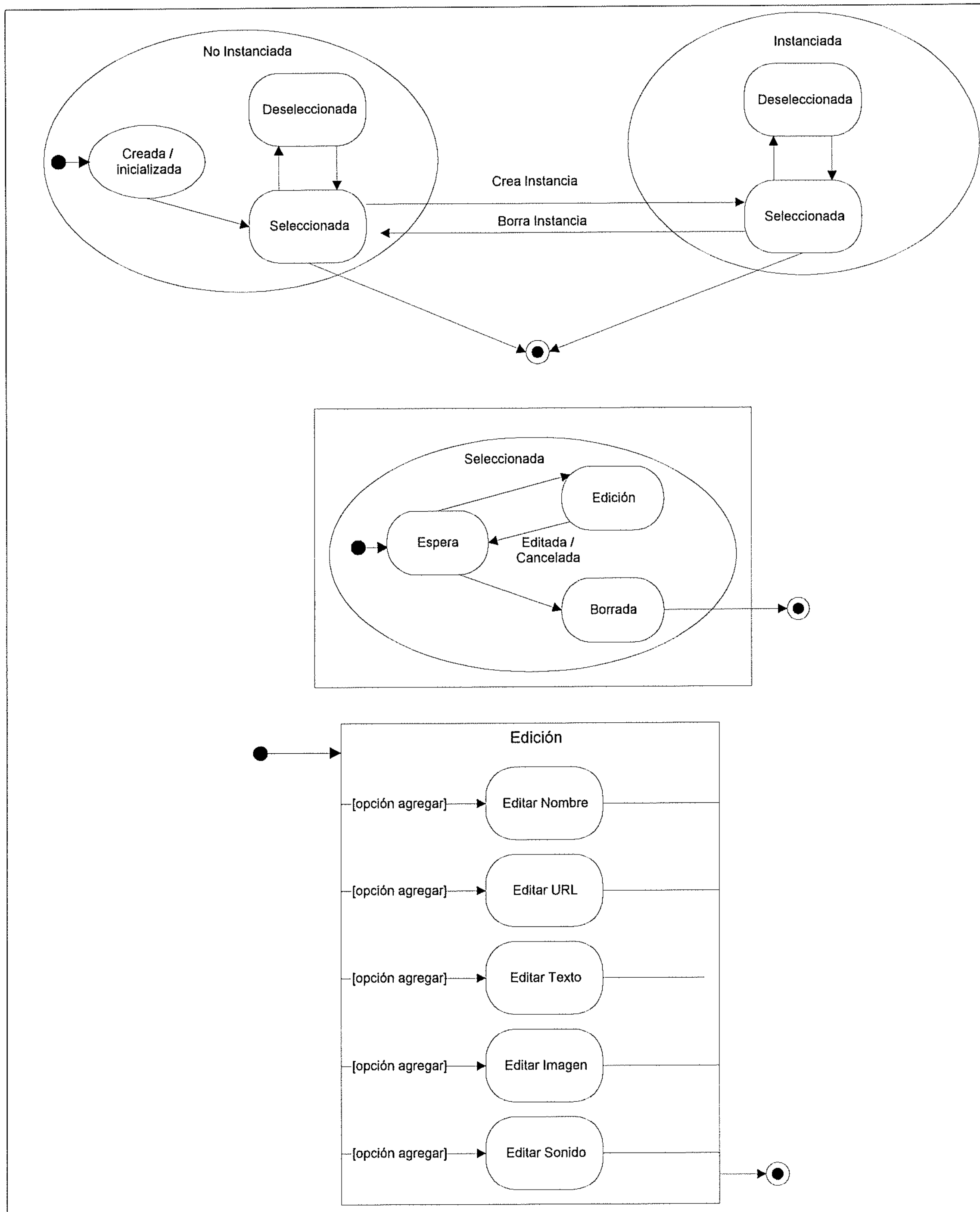


Figura 4.23. Diagrama de estados para la clase Relación

4.3.2.2.3. Diagrama de estados para la clase Instancia

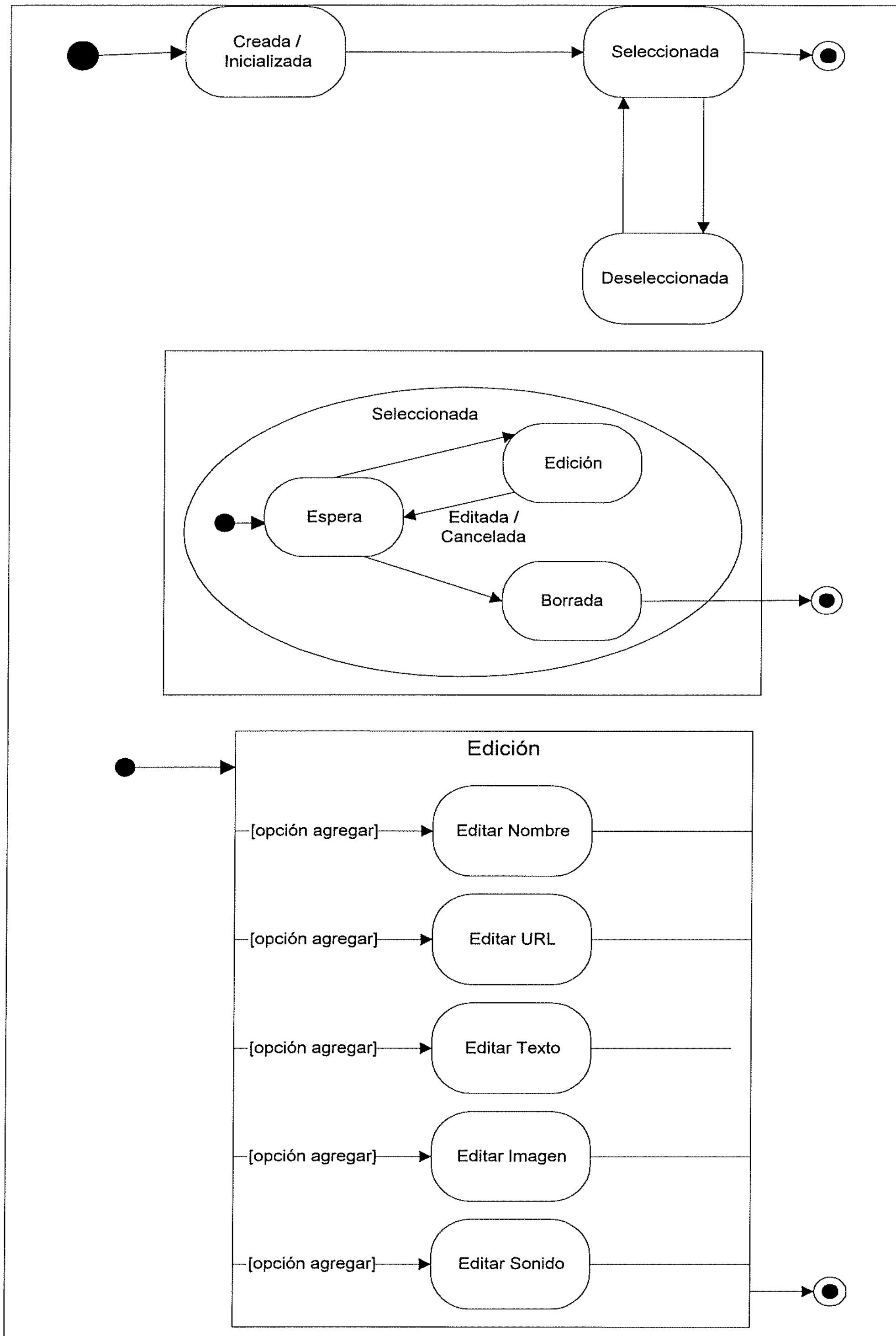


Figura 4.24. Diagrama de estados para la clase Instancia

4.3.3. Modelo Funcional Detallado

En esta fase se refina el modelo funcional basándonos en el modelo de objetos y modelo dinámico. El modelo funcional describe la forma en que responde el sistema frente a sucesos externos. A continuación se muestran los diagramas funcionales pertenecientes a las clases Red Semántica, Concepto, Relación e Instancia.

4.3.3.1. Diagrama de Flujo de Datos para la clase Red Semántica

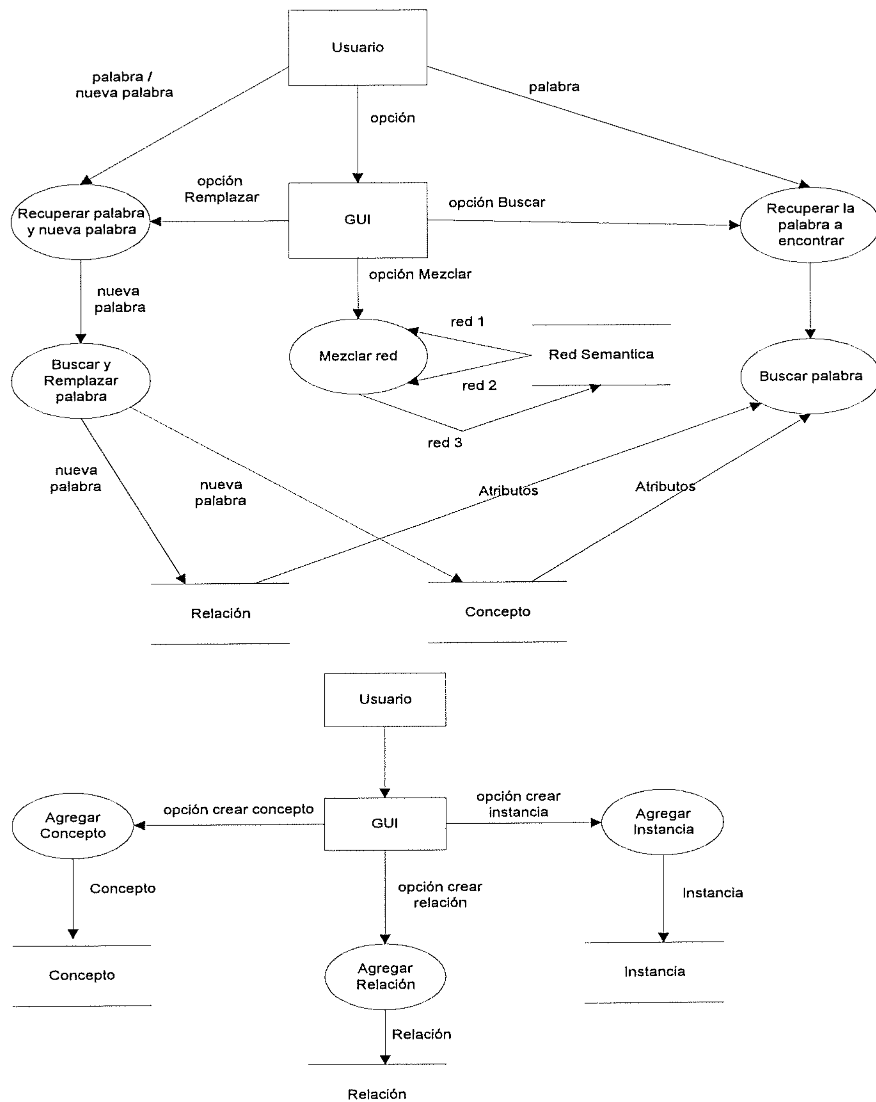


Figura 4.25. Diagrama de Flujo de Datos para la clase Red Semántica

4.3.3.2. Modelo de Flujo de Datos para la clase Concepto

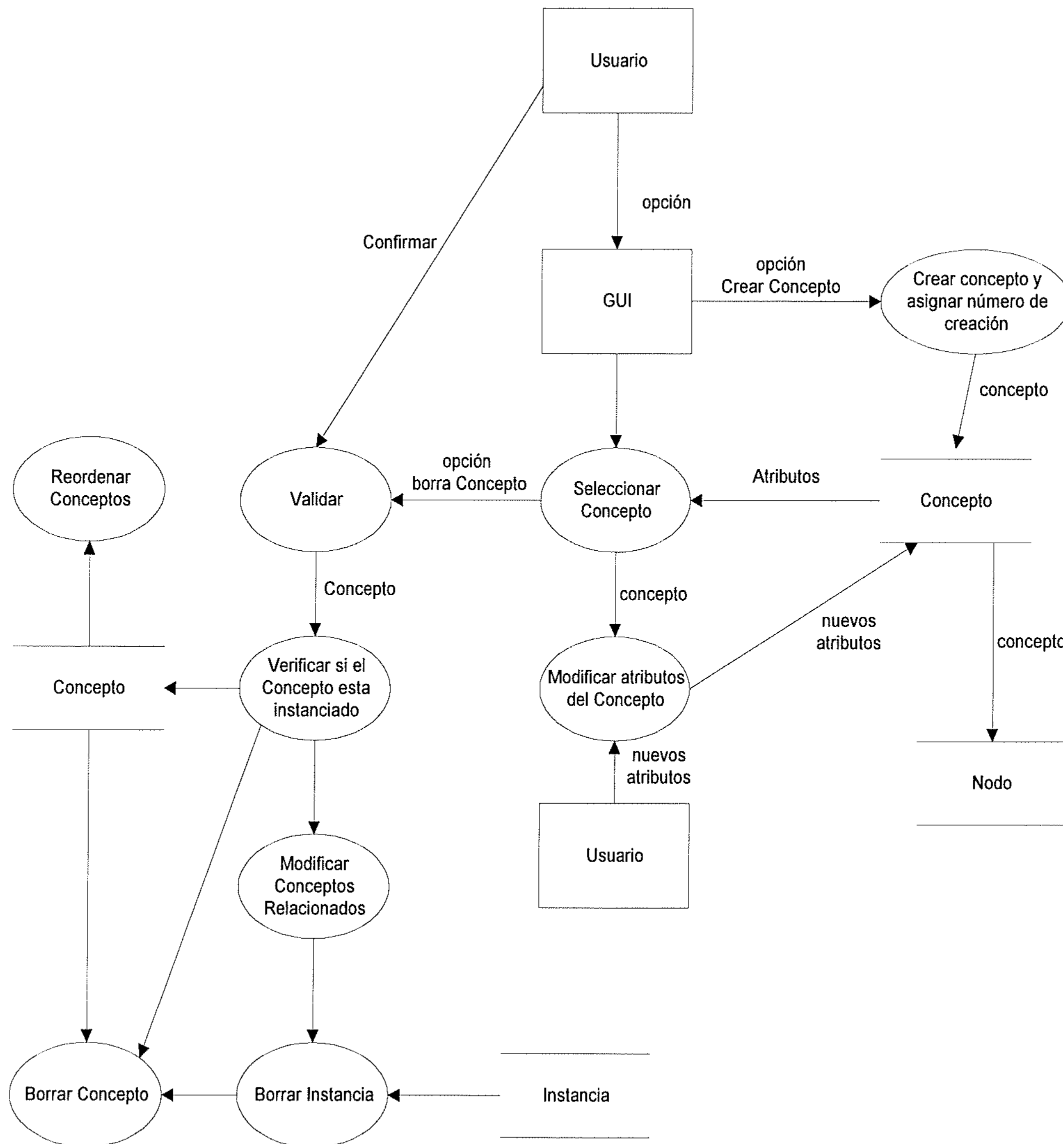


Figura 4.26. Diagrama de Flujo de Datos para la clase Concepto

4..3.3.3. Modelo de Flujo de datos para la clase Relación

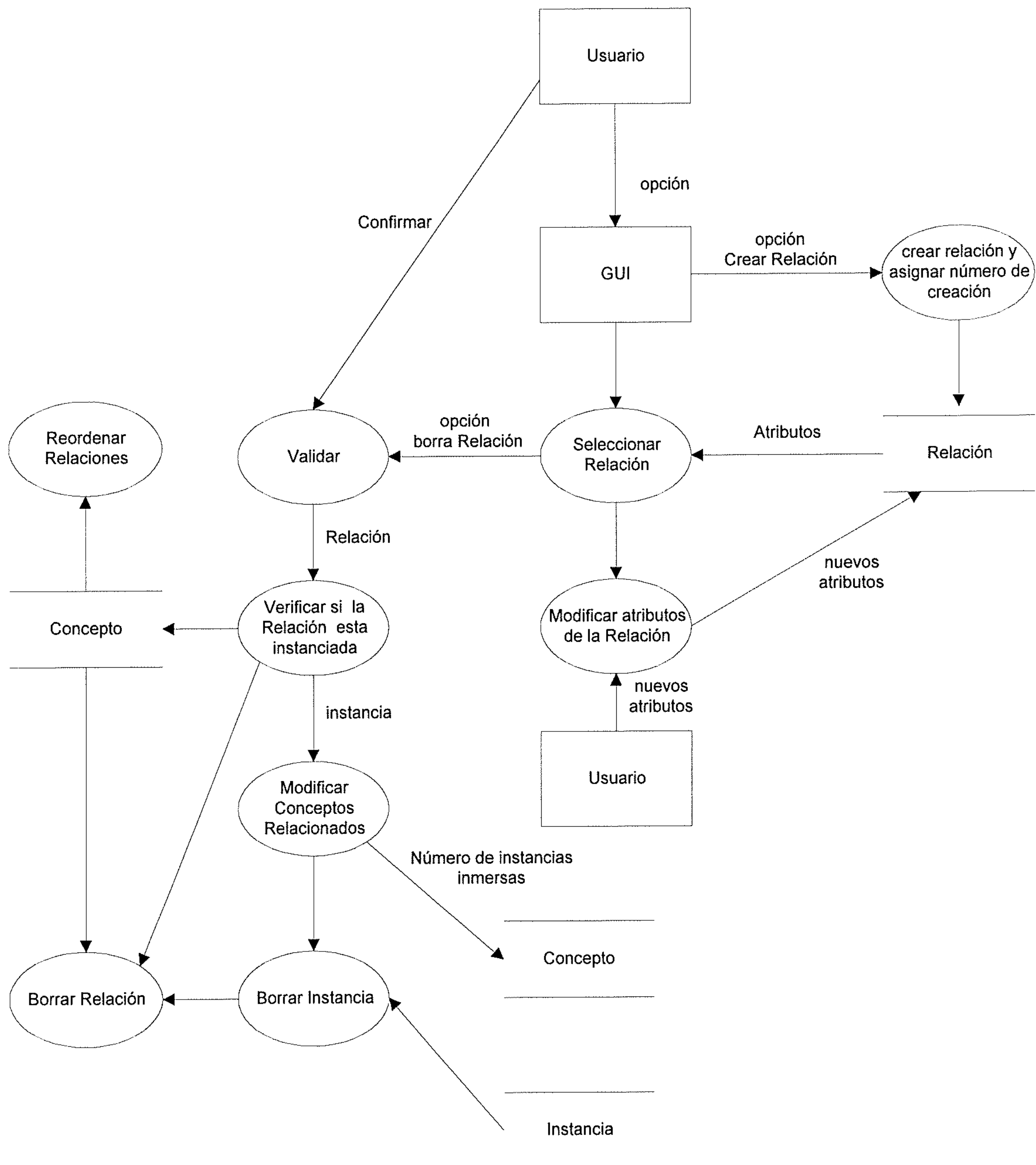


Figura 4.27. Diagrama de Flujo de Datos para la clase Relación

4.3.3.4. Modelo de Flujo de Datos para la clase Instancia

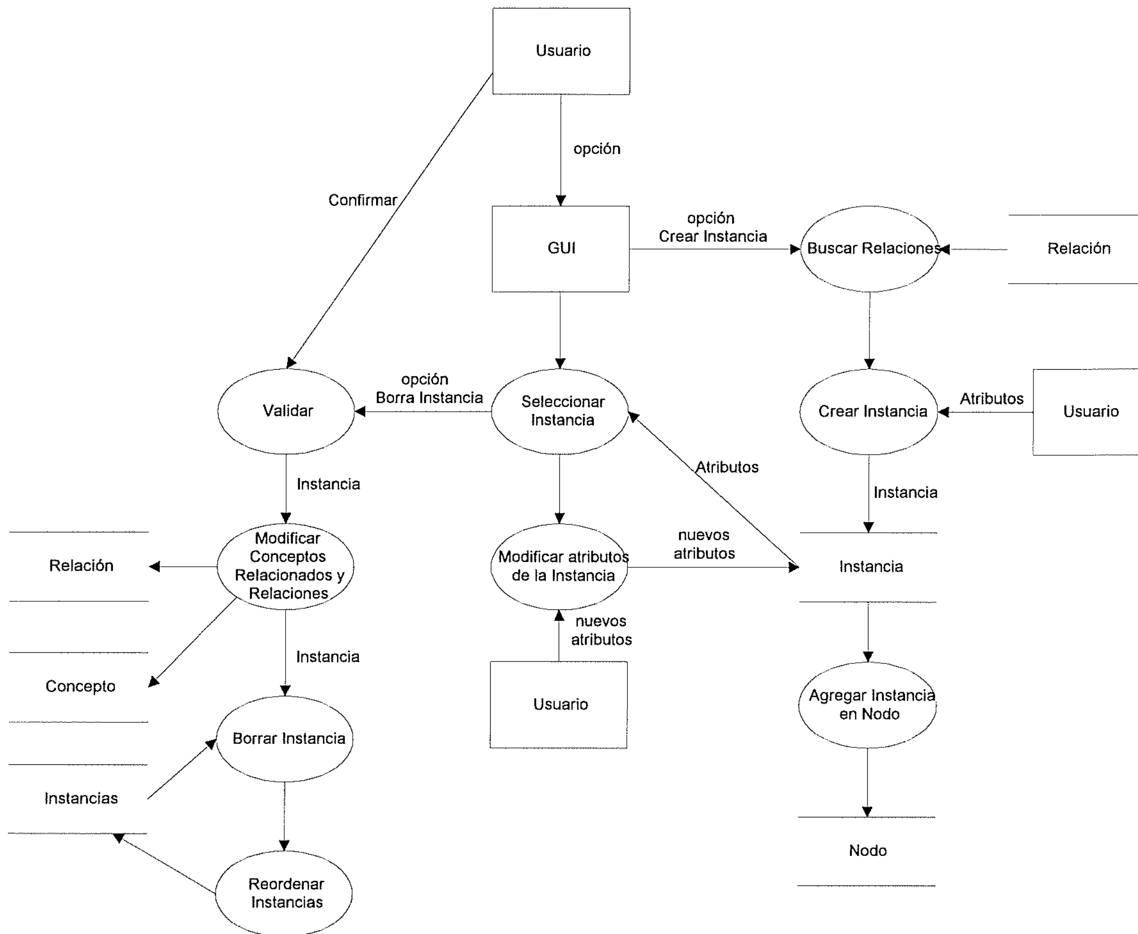


Figura 4.28. Diagrama de Flujo de Datos para Instancia

4.3.4. Descripción de algoritmos

A continuación se describen los métodos de las clases más importantes perteneciente al subsistema Máquina de Representación del Conocimiento.

4.3.4.1. Algoritmos de la clase Red Semántica

Nombre del método: *crear*

Descripción: crear una red semántica sin especificar el nombre del archivo de la red.

Crear( )

inicio

        Crear(SinNombre)

fin

Nombre del método: *Crear*

Parámetros de entrada: *NombreRed* de tipo Cadena

Descripción: crea una red semántica especificando el nombre de la red.

Crear(Cadena NombreRed)

inicio

        Nombre ← NombreRed

        Conceptos ← crear vector

        Relaciones ← crear vector

        Nodos ← crear vector

Fin

Nombre del método: *Encontrar*

Parámetros de entrada: *Nombre* de tipo Cadena, *tipoBúsqueda* de tipo Entero

Descripción: Encuentra la cadena introducida por el usuario especificándole el tipo de búsqueda como son conceptos o relaciones. Regresa el índice del concepto encontrado.

Encontrar(Cadena Nombre, Entero tipoBúsqueda)

inicio

Si tipoBúsqueda es igual a Concepto entonces

índice ← buscarConcepto (nombre)

regresa índice

otro si tipoBúsqueda es igual a Relación entonces

índice ← buscarRelación(nombre)

regresa índice

otro error no se especificó la búsqueda

fin

Nombre del método: *Remplazar*

Parámetros de entrada: *Nombre*, *NuevoNombe* de tipo cadena, *tipoBúsqueda* de tipo cadena.

Descripción: Encuentra y remplaza las cadenas introducidas por el usuario como es el nombre de la cadena a encontrar y la nueva cadena a remplazar.

Remplazar(Cadena Nombre, Cadena NuevoNombre, Entero tipoBúsqueda)

inicio

```
si tipoBúsqueda es igual a conceptos entonces
inicio
        con ← buscarConcepto(Nombre)
        si con es igual a nulo entonces
                "No se encontró el concepto"
        otro
                con.Nombre ← NuevoNombre
fin

si tipoBúsqueda es igual a Relación entonces
inicio
        rel ← buscarRelación(Nombre)
        si rel es igual a nulo entonces
                "No se encontró la Relación"
fin
otro
        rel.Nombre ← NuevoNombre
fin
```

Nombre del método: *AgregarConcepto*

Parámetros de entrada: *con* de tipo Concepto

Descripción: Este método agrega un nuevo concepto al vector de conceptos.

```
AgregarConcepto(Concepto con)
inicio
```

```
        índiceConcepto ← RegresaIndiceConcepto(con)

        Si índiceConcepto es igual a –1 entonces

        inicio

                Agrega  con al vector de conceptos

                AgregarNodo(con)

        fin

fin
```

Nombre del método: *AgregarRelación*

Parámetros de entrada: rel de tipo Relación

Descripción: Este método agrega una nueva relación al vector de relaciones.

```
AgregarRelación(Relación rel)

inicio

        índiceRelación ← RegresaIndiceRelación(rel)

        Si índiceRelación es igual a –1 entonces

        inicio

                Agrega  rel al vector de relaciones

        fin

fin
```

Nombre del método: *AgregarInstanciaANodo*

Parámetros de entrada: *índiceConceptoCentral* de tipo Entero, *insta* de tipo Instancia.

Descripción. Este método es el encargado de ligar las instancias al concepto central, es decir, que cada concepto es un concepto central y este puede tener cero o muchas instancias. Recibe como parámetros el índice del concepto y la instancia.

AgregaInstanciaANodo(Entero índiceConceptoCentral, Instancia insta)

inicio

nodo ← RegresaIndiceNodo(índiceConceptoCentral)

nodo.AgregaInstancia(insta)

fin

4.3.4.2. Algoritmos de la clase Concepto

Nombre del método: *Crear*

Parámetros de entrada: *NombreConcepto* de tipo cadena.

Descripción: crea un concepto especificando el nombre del concepto.

Crear(Cadena NombreConcepto)

inicio

Nombre ← NombreConcepto

Crear vector de sinónimos

fin

Nombre del método: *ponerImagen*

Parámetros de entrada: *nombreImagen* de tipo Cadena

Descripción: asigna el nombre del archivo de tipo imagen.

ponerImagen(Cadena nombreImagen)

inicio

imagen ← nombreImagen

fin

Nombre del método: *ponerTexto*

Parámetros de entrada: *descripción* de tipo Cadena

Descripción: método que agrega una cadena de texto al concepto.

ponerTexto(Cadena descripción)

inicio

Texto ← descripción

fin

Nombre del método: *ponerSonido*

Parámetros de entrada: *nombreSonido* de tipo cadena

Descripción: método que agrega el nombre del archivo de tipo sonido agregado al concepto.

ponerSonido(Cadena nombreSonido)

inicio

sonido ← nombreSonido

fin

Nombre del método: *ponerUrl*

Parámetros de entrada: *nuevoUrl* de tipo Cadena

Descripción: método que agrega un enlace al concepto.

ponerUrl(Cadena nuevoUrl)

inicio

url ← nuevoUrl

fin

Nombre del método: *ponerSinónimo*

Parámetros de entrada: *nuevoSinónimo* de tipo Vector

Descripción: método que agrega una cadena como sinónimo en el concepto.

ponerSinónimo(Vector nuevoSinónimo)

inicio

Sinónimo ← nuevoSinónimo

fin

Nombre del método: *ponerMáscara*

Parámetros de entrada: *masc* de tipo booleano

Descripción: agrega una máscara al concepto.

ponerMáscara(booleano masc)

inicio

Máscara ← masc

fin

Nombre del método: *ponerbandera*

Parámetros de entrada: *bandera* de tipo bolean

Descripción: agrega una bandera al concepto.

ponerbandera(boolean bandera)

inicio

flag ← bandera

fin

Nombre del método: *ponerColor*

Parámetros de entrada: *col* de tipo Color

Descripción: método que permite cambiar el color de concepto.

ponerColor(Color col)

inicio

Color ← col

fin

Nombre del método: *ponerLetra*

Parámetros de entrada: *fuente* de tipo Letra

Descripción: método que permite cambiar el estilo de la letra para el concepto.

ponerLetra(Letra fuente)

inicio

TipoLetra ← fuente

fin

Nombre del método: *ponerColorLetra*

Parámetros de entrada: *letraCol* de tipo Color

Descripción: método que permite cambiar el color de la letra para el concepto.

ponerColorLetra(Color letraCol)

inicio

LetraColor ← letraCol

fin

4.3.4.3. Algoritmos de la clase Nodo

Nombre del método: *ponerInstancia*

Parámetros de entrada: *ins* de tipo Instancia, *índiceNodo* de tipo Entero

Descripción: método que agrega una instancia al concepto central.

ponerInstancia(Instancia ins, Entero índiceNodo)

inicio

Si índiceNodo es menor que el número de instancias entonces

AgregaInstancia(ins,índiceNodo)

fin

Nombre del método: *borraInstancia*

Parámetros de entrada: *índiceRel* de tipo Entero, *índiceCon* de tipo Entero

Descripción: método que borra la instancia de nodo, eliminando el enlace de la relación y el concepto relacionado.

```
BorrarInstancia(Entero índiceRel, Entero índiceCon)
inicio
        Mientras exista instancias hacer
        inicio
                Irel ← regreda índice de relación
                Icon ← regresa índice de concepto
                Si índiceRel es igual a Irel e  índiceCon es igual a Icon entonces
                        Eliminar instancia
fin
```

Nombre del método: *RegresaInstancia*

Parámetros de entrada: *índiceInstancia* de tipo Entero

Descripción: Regresa la instancia, buscando con base en el índice de la instancia.

```
RegresaInstancia(Entero índiceInstancia)
inicio
        Si  índiceInstancia es menor que el número de instancias entonces
                regresa instancia(índiceInstancia)
        otro
                regresa nulo
fin
```

4.3.4.4. Algoritmos de la clase Relación

Nombre del método: *Crear*

Parámetros de entrada: *desdeCentral* de tipo Flecha y *AlCentral* de tipo Flecha

Descripción: Crea una nueva relación con sus dos sentidos como son: la relación que va desde el concepto central hacia el concepto relacionado y la relación que va desde el concepto relacionado al concepto central.

Crear(Flecha desdeCentral, Flecha AlCentral)

inicio

DesdeConCentral ← desdeCentral

AlConCentral ← AlCentral

fin

Nombre del método: *regresaFlechaDesdeCentral*

Parámetros de entrada: *ninguno*

Descripción: regresa la relación con sus atributos que va desde el concepto central hacia el concepto relacionado.

regresaFlechaDesdeCentral()

inicio

regresar DesdeConCentral

fin

Nombre del método: *regresaFlechaAlCentral()*

Parámetros de entrada: *ninguno*

Descripción: regresa la relación con sus atributos que va desde el concepto relacionado al concepto central.

regresaFlechaAlCentral()

inico

regresar AlConCentral

fin

4.3.4.5. Algoritmos de la clase Flecha

Nombre del método: *Crear*

Parámetros de entrada: *etiqueta* de tipo cadena, *nuevaPosición* de tipo Entero.

Descripción: crea una flecha la cual es la representación gráfica de la relación.

Crear(Cadena etiqueta, Entero nuevaPosición)

Inicio

Nombre ← etiqueta

Posición ← nuevaPosición

Fin

Nombre del método: *ponerImagen*

Parámetros de entrada: *nombreImagen* de tipo Cadena

Descripción: asigna el nombre del archivo de tipo imagen.

ponerImagen(Cadena nombreImagen)

inicio

imagen ← nombreImagen

fin

Nombre del método: *ponerTexto*

Parámetros de entrada: *descripción* de tipo Cadena

Descripción: método que agrega una cadena de texto a la relación.

ponerTexto(Cadena descripción)

inicio

Texto ← descripción

fin

Nombre del método: *ponerSonido*

Parámetros de entrada: *nombreSonido* de tipo cadena

Descripción: método que agrega el nombre del archivo de tipo sonido agregado a la relación.

ponerSonido(Cadena nombreSonido)

inicio

sonido ← nombreSonido

fin

Nombre del método: *ponerUrl*

Parámetros de entrada: *nuevoUrl* de tipo Cadena

Descripción: método que agrega un enlace a la relación.

ponerUrl(Cadena nuevoUrl)

inicio

url ← nuevoUrl

fin

Nombre del método: *ponerSinónimo*

Parámetros de entrada: *nuevoSinónimo* de tipo Vector

Descripción: método que agrega una cadena como sinónimo a la relación.

ponerSinónimo(Vector nuevoSinónimo)

inicio

Sinónimo ← nuevoSinónimo

fin

Nombre del método: *ponerMáscara*

Parámetros de entrada: *masc* de tipo booleano

Descripción: agrega una máscara a la relación.

ponerMáscara(boolean masc)

inicio

Máscara ← masc

fin

Nombre del método: *ponerbandera*

Parámetros de entrada: *bandera* de tipo bolean

Descripción: agrega una bandera a la relación.

ponerbandera(boolean bandera)

inicio

flag ← bandera

fin

Nombre del método: *ponerColor*

Parámetros de entrada: *col* de tipo Color.

Descripción: método que permite cambiar el color de la relación.

ponerColor(Color col)

inicio

Color ← col

fin

Nombre del método: *ponerLetra*

Parámetros de entrada: *fuente* de tipo Letra

Descripción: método que permite cambiar el estilo de la letra para la relación.

ponerLetra(Letra fuente)

inicio

TipoLetra ← fuente

fin

Nombre del método: *ponerColorLetra*

Parámetros de entrada: *letraCol* de tipo Color

Descripción: método que permite cambiar el color de la letra para la relación.

ponerColorLetra(Color letraCol)

inicio

LetraColor ← letraCol

fin

4.3.4.6. Algoritmos de la clase Instancia

Nombre del método: *Crear*

Parámetros de entrada: *índiceRel, índiceCon, dirección, posición* de tipo cadena.

Descripción: crea una instancia inicializando sus atributos.

Crear(Entero iRel, Entero iCon, Entero dir, Entero pos)

inicio

índiceRel ← iRel

índiceCon ← iCon

dirección ← dir

posición ← pos

fin

Nombre del método: *ponerImagen*

Parámetros de entrada: *nombreImagen* de tipo Cadena

Descripción: asigna el nombre del archivo de tipo imagen.

ponerImagen(Cadena nombreImagen)

inicio

imagen ← nombreImagen

fin

Nombre del método: *ponerTexto*

Parámetros de entrada: *descripción* de tipo Cadena

Descripción: método que agrega una cadena de texto a la instancia.

ponerTexto(Cadena descripción)

inicio

Texto ← descripción

fin

Nombre del método: *ponerSonido*

Parámetros de entrada: *nombreSonido* de tipo cadena

Descripción: método que agrega el nombre del archivo de tipo sonido agregado a la instancia .

ponerSonido(Cadena nombreSonido)

inicio

sonido ← nombreSonido

fin

Nombre del método: *ponerUrl*

Parámetros de entrada: *nuevoUrl* de tipo Cadena

Descripción: método que agrega un enlace a la intancia.

ponerUrl(Cadena nuevoUrl)

inicio

url ← nuevoUrl

fin

Nombre del método: *ponerSinónimo*

Parámetros de entrada: *nuevoSinónimo* de tipo Vector

Descripción: método que agrega una cadena como sinónimo a la instancia.

ponerSinónimo(Vector nuevoSinónimo)

inicio

Sinónimo ← nuevoSinónimo

fin

4.3.5. Implementación

La implementación del sistema fue utilizando el lenguaje Java usando el compilador JDK 1.2 de Sun Microsystems. Para el desarrollo de la interfaz gráfica de usuario se utilizó el paquete Java Foundation Classes (JFC). En el apéndice A (véase página 84) se muestra parte de la implementación de las clases pertenecientes al subsistema de la máquina de representación del conocimiento.

5. RESULTADOS

Durante el desarrollo del sistema se utilizó la técnica de modelado por objetos (OMT) obteniéndose resultados en cada fase de la metodología y cubriendo el objetivo general planteado en esta tesis. A continuación se describen los resultados que se obtuvieron en cada fase de la metodología OMT:

En la fase de **Análisis** se obtuvo lo siguiente:

Modelo de Objetos (página 14) captura los objetos del sistema y sus relaciones.

- Diagrama de clases
- Diccionario de Datos.

Modelo Dinámico (página 20) describe la reacción de los objetos del sistema frente a eventos y las iteraciones entre objetos.

- Escenarios y trazo e eventos
- Diagrama general de flujo de eventos
- Diagrama de estados para cada clase importante

Modelo Funcional (página 30) especifica las transformaciones de objetos y las restricciones aplicables a estas transformaciones.

- Diagrama de flujo de datos
- Algoritmos

Para la fase de **Diseño de Sistemas** se obtuvo:

- Arquitectura del sistema, en donde se decide la organización del sistema así como sus componentes de hardware y software.

En la fase de **Diseño de Objetos** los resultados obtenidos fueron:

- Modelo de objetos detallado
- Modelo dinámico detallado
- Modelo funcional detallado

Antes de comenzar el diseño de objetos se procedió a documentar el código de la primer versión de java la cual fue desarrollada por L.C.C. José Manuel Alba Álvarez y se realizó una iteración más del análisis con la finalidad de homologar el código existente en java con el análisis propuesto y hacer las modificaciones necesarias en el diseño de objetos e incluir las características faltantes.

En la fase de **Implementación** se continuó desarrollando bajo el lenguaje Java según el análisis realizado previamente.

Para demostrar las capacidades y resultados de la herramienta de representación del conocimiento, se utilizará un ejemplo propuesto por Sánchez (1990), el cual se describe a continuación.

" Los árboles son plantas, con raíz, tallo y hojas cuya altura en edad adulta supera los seis metros; se distinguen de los arbustos por su tronco. Dos de los árboles más característicos de la península ibérica son el pino piñero y la encina. La encina es un árbol robusto de copa redondeada, que puede alcanzar los 15 ó 20 metros de altura, la corteza es gris, las hojas son gruesas y ovaladas, su fruto es la bellota" [Sánchez, 1990].

Con base en la descripción antes mencionada se realizan los siguiente pasos para la **construcción de la red semántica** usando la herramienta.

1. Presione **Archivo** en la barra de menú y seleccione la opción **Nuevo**. Una ventana sin nombre será desplegada y aparecerá en el centro un concepto central sin título (Figura 5.1). Este es la idea central que será descrita.

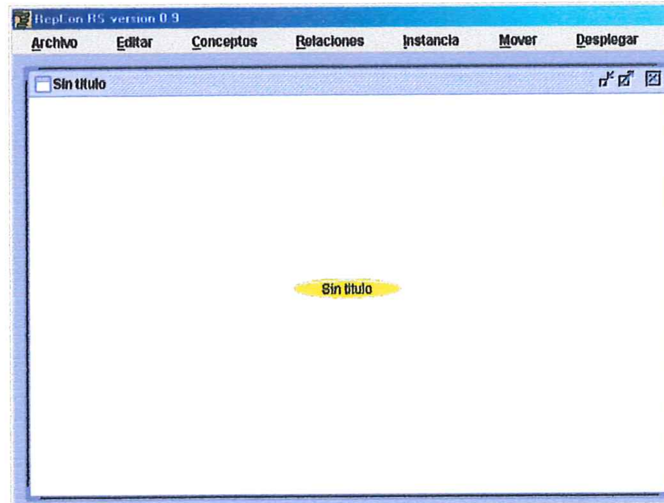


Figura 5.1. Muestra el concepto central sin título

2.- Ahora falta insertar el nombre del concepto. Esto se hace presionando el **concepto central** y enseguida aparecerá un diálogo (Figura 5.2) que permite editar el nombre del concepto. Ahora escriba el nombre del concepto: árbol.

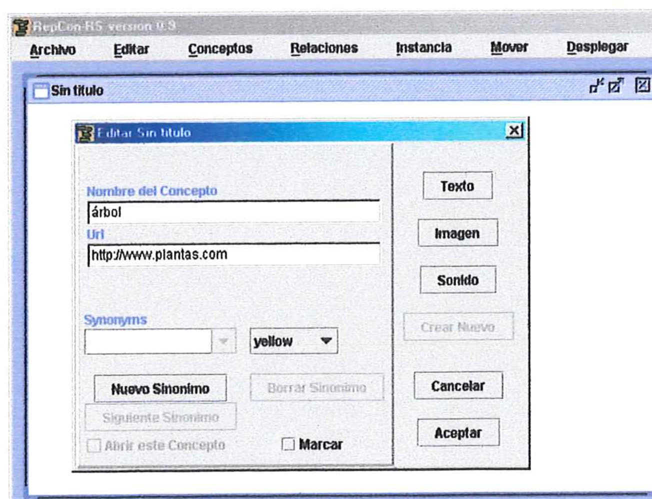


Figura 5.2 Muestra la edición de un concepto central.

3.- Para crear una instancia, se selecciona **Instancia** en la barra de menú y se selecciona la opción **crear**, enseguida aparecerá el diálogo para crear la instancia (Figura 5.3). Escriba ahora los nombres de las relaciones y el nombre del concepto relacionado.

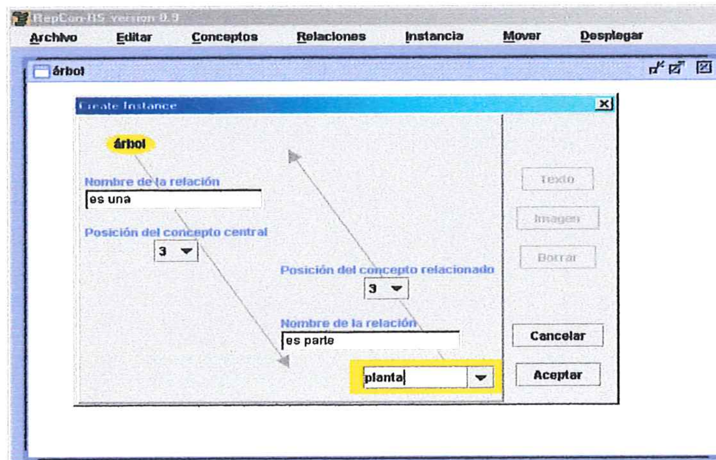


Figura 5.3. Muestra la creación de una instancia.

4. – Una vez completado lo anterior, repita el proceso para crear las otras instancias para los conceptos ***árbol*** y ***planta***. El resultado se muestra en las Figuras 5.4 y 5.5.

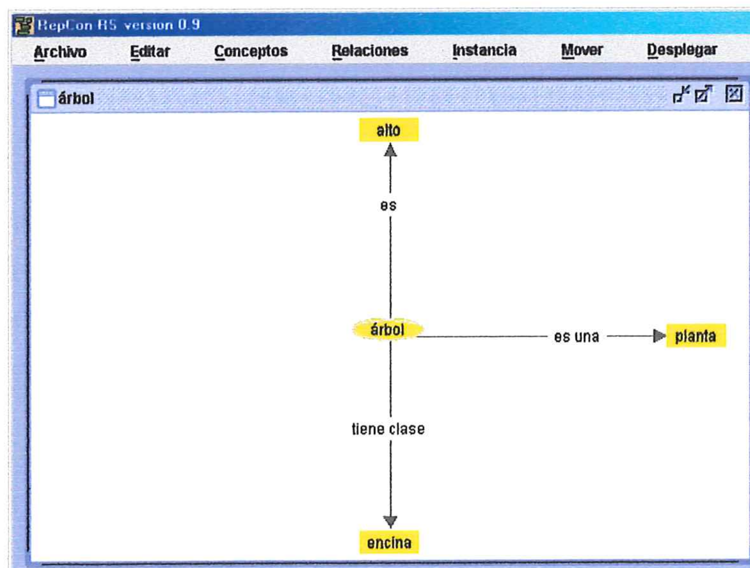


Figura 5.4. Muestra las instancias del concepto árbol.

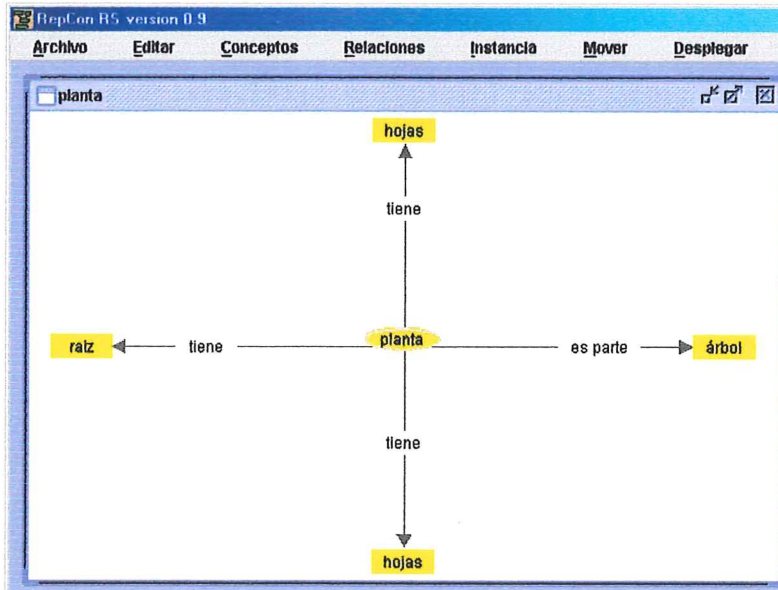


Figura 5.5. Muestra las instancias del concepto planta.

Además de construir una red semántica simple, la herramienta permite agregar elementos multimedia como texto, imágenes, sonidos y sinónimos para ampliar la descripción del concepto o idea. A continuación se describe el proceso para agregar estos elementos multimedia.

1.- Para agregar **texto** a un concepto se presiona el **concepto** y enseguida se desplegará el diálogo que permite editar las propiedades del concepto, después se selecciona el botón **Text,** o presione **Concepto** en la barra de menú y seleccione la opción **Texto** el cual mostrará el diálogo para agregar el texto tal como lo muestra la figura 5.6.

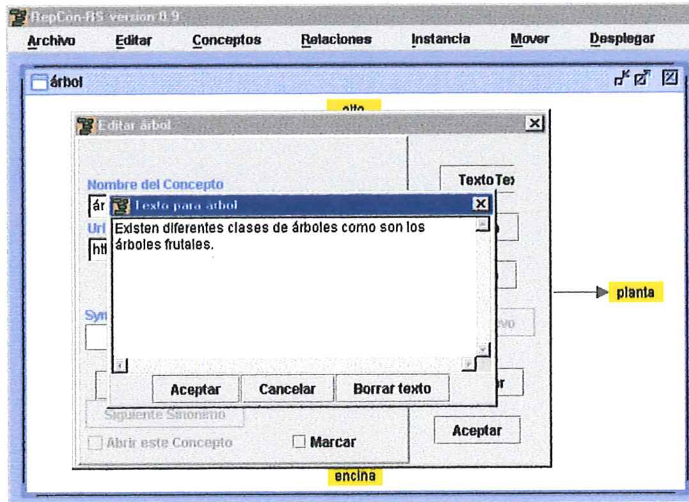


Figura 5.6. Muestra el diálogo para agregar texto al concepto.

2.- Para agregar **sonido** a un concepto se presiona el **concepto** después se selecciona el botón **Sonido,** o presione **Concepto** de la barra de menú y seleccione la opción **Sonido** y enseguida se desplegará el diálogo que permite editar las propiedades del concepto, el cual mostrará el diálogo para agregar el sonido tal como lo muestra la figura 5.7.

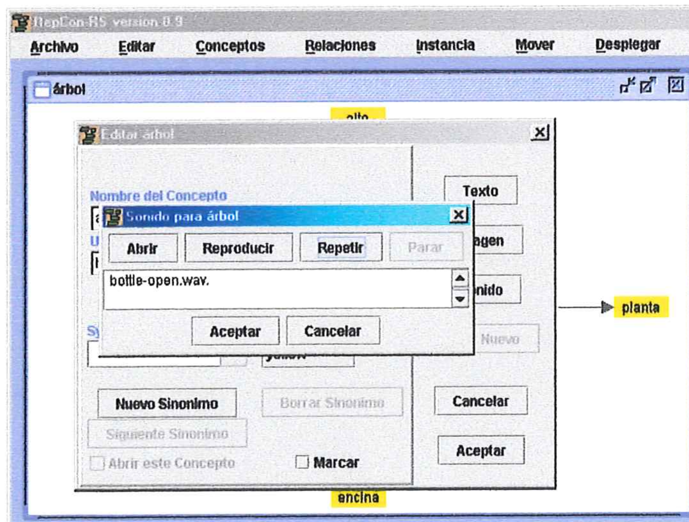


Figura 5.7. Muestra el diálogo para agregar sonido al concepto.

3.- Para agregar una **imagen** a un concepto, presionamos el concepto deseado, enseguida se desplegará el diálogo que permite editar las propiedades del concepto, después se selecciona el botón **Imagen**, o si prefiere presione **Concepto** de la barra de menú y seleccione **Sonido,** el cual mostrará el diálogo para agregar la imagen tal como lo muestra la figura 5.8.

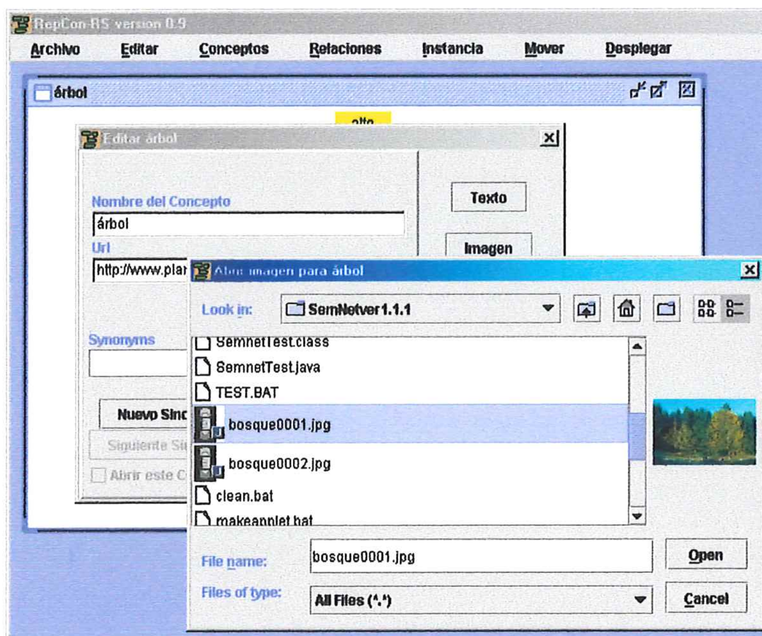


Figura 5.8. Muestra el diálogo para agregar una imagen al concepto.

6.- Una vez creada la red semántica la navegación de conceptos o ideas es fácil, sólo se tiene que presionar el concepto para que este despliegue sus conceptos relacionados así como sus elementos agregados.

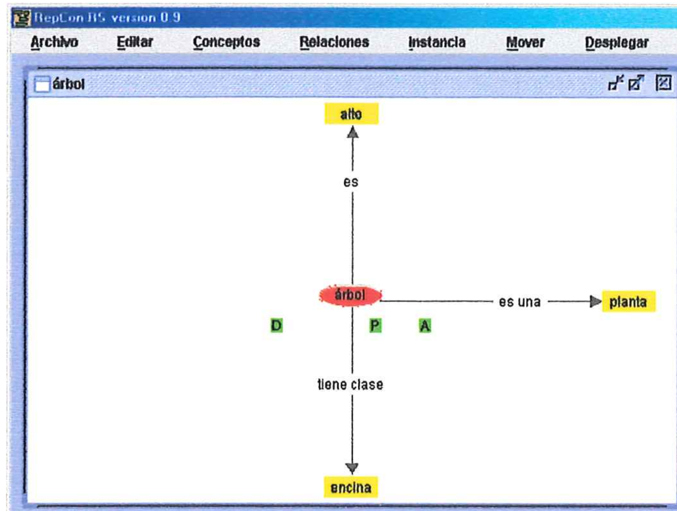


Figura 5.9. Muestra la red semántica con el concepto central árbol y sus elementos agregados.

Para visualizar los elementos agregados sólo se presiona el icono de la letra **D** para visualizar el texto agregado al concepto, **P** para visualizar la imagen, y **A** para escuchar el audio agregado al concepto.

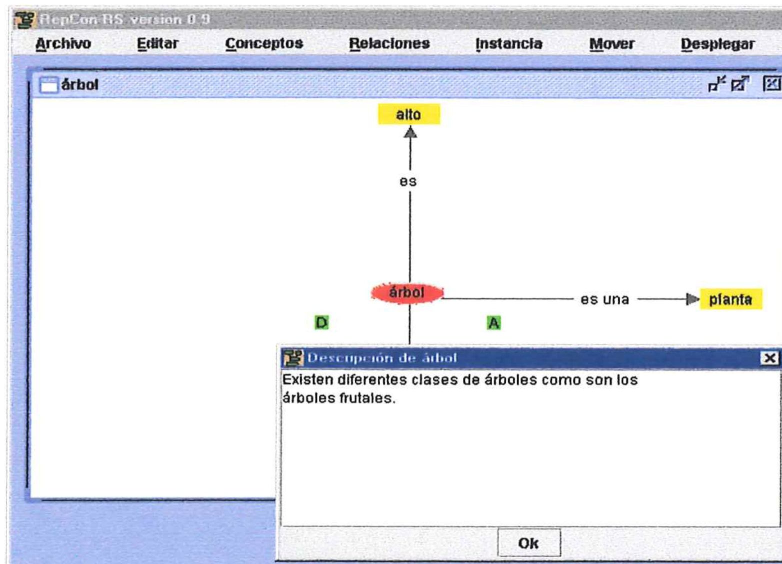


Figura 5.10. Muestra el concepto árbol con texto agregado.

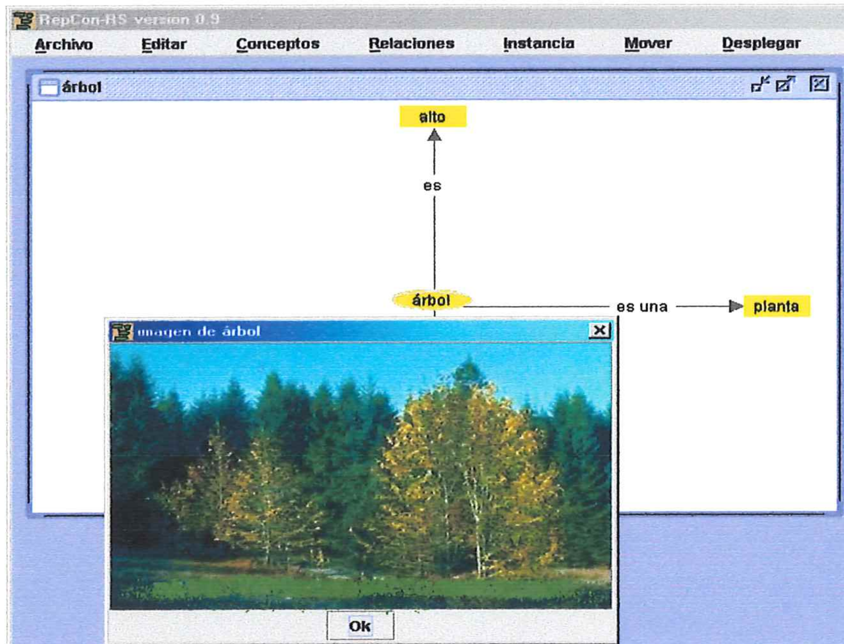


Figura 5.11. Muestra el concepto árbol con imagen agregada.

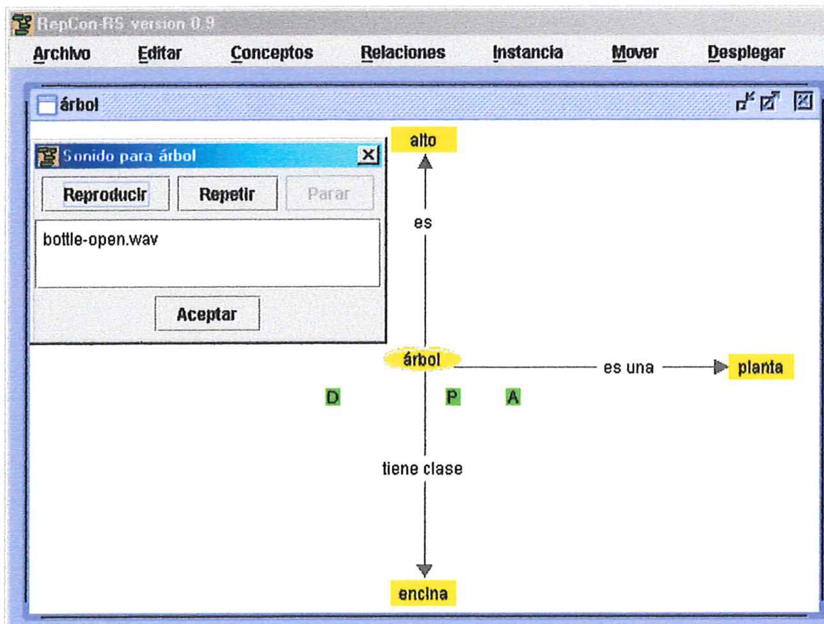


Figura 5.12. Muestra el concepto árbol con sonido agregado

Los resultados que se describieron son sólo la parte más importante de la herramienta de representación del conocimiento las demás funciones se describen en el manual de usuario.

6. DISCUSIÓN

Las redes semánticas han sido utilizadas para una gran variedad de propósitos, ofrecen una descripción de la información jerárquica, donde los nodos pueden representar clases o instancias de objetos y las relaciones pueden definir las características entre las clases o instancias. Por último las redes semánticas proveen una descripción del conocimiento debido a que proporciona una descripción gráfica de la información representada [Elaine, 1991].

En el presente trabajo, se desarrolló el análisis, diseño e implementación de una herramienta para la representación del conocimiento usando redes semánticas (RepConRS), utilizando una metodología orientada a objetos. El modelado y diseño orientado a objetos constituye una nueva forma de pensar los problemas empleando modelos que se han organizado tomando como base conceptos del mundo real. Los modelos orientados a objetos son útiles para comprender problemas, comunicarse con expertos de una aplicación, modelar empresas, preparar documentación y diseñar programas y bases de datos. La esencia del desarrollo orientado a objetos es la identificación y organización de conceptos del dominio de la aplicación, y no de su representación final en un leguaje de programación [Rumbaugh *et al*, 1991].

Para el desarrollo del sistema se recurrió al ciclo de vida de la ingeniería del software bajo el modelo de versiones sucesivas, este modelo es de gran utilidad ya que permite llevar un mejor control de los avances logrados durante el desarrollo.

Para la primera versión se comenzó a revisar la especificación de requerimientos así como revisar la funcionalidad del sistema SemNet de la versión Macintosh, una vez familiarizados con el problema se inicio la segunda versión la cual inicio en el análisis cubriendo hasta la arquitectura del sistema, el motivo por la que se llego hasta esta etapa fue por que se entrego la primer versión preliminar de java y se tuvo que realizar la documentación necesaria debido a que carecía de ella para seguir desarrollando. Se documentó el código fuente y se obtuvo un documento de análisis para la versión de java. En la tercer versión se desarrollo el análisis, diseño e implementación, obteniéndose el análisis definitivo resultado de las versiones anteriores.

La herramienta (RepConRS) permite organizar la información acerca de cualquier tema mediante la representación del conocimiento usando redes semánticas de una manera fácil. Además, permite incluir información de otros tipos, como imágenes, textos, gráficas, sinónimos y sonidos al mismo tiempo que representa el concepto o la idea. RepConRS es fácil de usar y es ideal para la construcción del conocimiento grupal o personal, también ayuda a crear material de referencia ofreciendo un acceso fácil y rápido a múltiples perspectivas.

La implementación fue utilizando el lenguaje Java, este permite la portabilidad en múltiples plataformas. Java es un lenguaje de programación de propósito general para desarrollar programas que sean fáciles de usar y portables a gran variedad de plataformas [Lemay, 1997].

Java como lenguaje de programación, tiene ventajas significativas sobre otros lenguajes y otros ambientes que lo hacen apropiado para realizar cualquier tarea de programación. Enseguida se describe algunas de esas ventajas.

- **Simple**

  En Java no se incorporan muchos aspectos confusos y raramente usados de otros lenguajes que según la experiencia, en vez de beneficiar perjudican al desarrollador.

- **Orientado a objetos**

  Java incorpora la técnica de programación orientada a objetos lo que permite utilizar en su totalidad las ventajas de la metodología orientada a objetos y su capacidad de construir programas flexibles y modulares, así como reutilizar código.

- **Robusto**

  Java fue creado para escribir programas (applets y aplicaciones) confiables.

- **Portable**

  Java es portable ya que permite la ejecución de un programa en diferentes plataformas, tanto en código fuente como en binario.

- **Seguro**

  Java permite la construcción de sistemas libres de virus.

- **Multitarea**

  Java permite que un programa realice más de una tarea a la vez.

7. CONCLUSIONES

- La presente herramienta permite organizar la información acerca de cualquier tema mediante la representación del conocimiento usando redes semánticas de una manera sencilla.

- Permite incluir información de otros tipos, como imágenes, textos, gráficas, sinónimos y sonidos al mismo tiempo que representa el concepto o la idea.

- Los estudiantes pueden utilizar la herramienta en diferentes momentos para aprender y entender mejor la información obtenida durante sus procesos de estudio para el auto-aprendizaje.

- El estudiante extiende su red semántica al agregar más información conforme van avanzando las clases y va relacionando el conocimiento anterior con el recién adquirido, llegando a crear redes de temas extensos.

- Una vez representado el conocimiento del tema (por él mismo o por alguien más), el estudiante puede repasar sus propias redes semánticas, debido a la facilidad que la herramienta le proporciona para la navegación de la información.

- El comportamiento de dicho lenguaje fue satisfactorio debido a que la herramienta fue probada en las plataformas Windows NT, Windows 98 y LINUX pudiéndose comprobar la portabilidad para diferentes sistemas operativos.

- Una vez revisado los objetivos planteados en la tesis así como los resultados obtenidos se puede afirmar que las metas fueron cumplidas satisfactoriamente, debido a que se cumplió con los requerimientos establecidos en los objetivos.

8. LITERATURA CITADA

Bratko I. 1990. "Prolog, Programing for Artificial Intelligence". Addison – Wesley, Segunda edición. Menlo Park, Calif. 348 p.

Elaine Rich 1991. "Artificial Intelligence". International Edition, Second Edition. Singapore.

Fisher, K., J. Faletti, H. Patterson, R. Thornton, J. Lipson & Spring. 1990. "SemNet Software for the Macintosh, version 1.1", http://trumpet.sdsu.edu/semnet.html;.

Lemay Laura y L. Perkins 1997, "Teach Yourself Java 1.1 in 21 Days". Sams.Net Publishing, Segunda edición. 4p – 11p.

Pressman, R.S. 1992. "Software engineering. A practitioner's approach". McGraw-Hill, Inc. Tercera edición. New York. 824 pp.

Rumbaugh J., M. Blaha, W. Premerlani, F. Eddy, & W. Lorensen. 1991. "Object-oriented modeling and design". Prentice-Hall, Inc. Primera Edición, Englewood, NJ.

Sanchez y Beltrán. 1990. "Sistemas Expertos una metodología de programación". Macrobit. México. 201p - 205p.

Shildt Herbert. 1988. "Utilización de C en Inteligencia Artificial". McGraw-Hill. España. 1p.

Turban Efraim. 1992. "Expert Systems and Applied Artificial Intelligence". Editorial Macmillan. New York. 174 p.

9. APÉNDICE A (Implementación de clases del subsistema Máquina de Representación del Conocimiento)

En base al diseño de objetos se muestra la implementación de las clases del subsistema perteneciente a la Máquina de Representación del Conocimiento.

9.1. Clase Concepto (Concept)

```
/* Concept.java */
package SemNet.DataRepresentation;

import java.awt.Color;
import java.awt.Font;
import java.io.Serializable;
import java.util.Vector;
import java.util.Enumeration;

/**
 * Clase Concept, contiene todas la características de los conceptos
 * @see java.awt.Color
 * @see java.io.Serializable
 * @see java.util.Vector
 * @see java.util.Enumeration
*/
public class Concept implements Serializable {
        /** nombre del concepto */
        private String name              = null;
       /** descripción, texto agregado al concepto */
       private String description        = null;
       /** url, liga agregada al concepto */
       private String url                                    = null;
       /**  Sinónimos agregados al concepto */
       private Vector synonyms           = null;
      /**  marca para el concepto */
      private boolean masked            = false;
      /** Marca o bandera para conceptos (*) */
      private boolean flag              = false;
      /**  nombre del archivo de la imagen agregada al concepto */
      private String imageFileName      = null;
      /** Nombre del archivo de sonido agregado al concepto */
      private String soundFileName      = null;
      /** letra para el concepto */
      private Font fontConcept = null;
      /** color para el concepto sin seleccionar color por default*/
      private Color  color              = Color.yellow;
      /** color del concepto cuando es seleccionado */
      private Color  highLightColor    = Color.red;
      /** color de la letra para el concepto */
      private Color  fontColor          = Color.black;

      /** constructor crea un nuevo concepto sin nombre */
      public Concept() {
         this("");
      }

      /** crea un nuevo concepto con su nombre
       *@param conceptName nombre del concepto
       */
      public Concept(String conceptName) {
                                  name = conceptName;
```

```
                        synonyms = new Vector();
                }

/** crea un nuevo concepto con su nombre y el texto agregado
 * @param conceptName nombre del concepto
 * @param conceptDesc texto agregado al concepto
 */
public Concept(String conceptName, String conceptDesc) {
  this(conceptName);
  description = conceptDesc;
}

/** Crea un nuevo concepto con su nombre, su texto agregado y su URL
 * @param conceptName nombre del concepto
 * @param conceptDesc texto agregado al concepto
 * @param conceptUrl url agregado al concepto
 */
public Concept(String conceptName, String conceptDesc,
      String conceptUrl) {
  this(conceptName,conceptDesc);
  url = conceptUrl;
}

/** pone nombre al concepto
 * @param newName nombre del concepto*/
public void setName(String newName) {
                name = newName;
        }

/** pone la descripción ( el texto agregado )
 * @param newDescription texto agregado al concepto
 */
        public void setDescription(String newDescription) {
                description = newDescription;
        }

/** pone liga (URL) a una página web
 * @param newUrl liga agregada al concepto
 */
                public void setUrl(String newUrl) {
                        url = newUrl;
                }

/** pone sinonimos al concepto
 * @param newSynonymsVector sinonimos agregados al concepto
 */
public void setSynonyms(Vector newSynonymsVector) {
        synonyms = newSynonymsVector;
}

/** pone mascara al concepto
 * @param masked mascara para el concepto
 */
public void setMasked(boolean mask) {
  masked = mask;
}

/** pone marca al concepto
 * @param flag marca para el concepto
 */
        public void setFlaged(boolean Flag){
          flag = Flag;
        }

/** pone el nombre de la imagen agregada al concepto
 * @param fName nombre del archivo de la imagen
```

```
 */
public void setImageFileName(String fName) {
        imageFileName = fName;
}

/** pone el nombre del sonido agregada al concepto
 * @param sName nombre del archivo sonido
 */
public void setSoundFileName(String sName) {
        soundFileName = sName;
}

/** pone el tipo de letra para el concepto
  * @param newfont nuevo font para el concepto
  */
public void setFontConcept(Font newfont){
    fontConcept = newfont;
}


/** pone el color al concepto
 * @param newColor color del concepto
 */
        public void setColor(Color newColor) {
                        color = newColor;
        }

/** pone color al concepto cuando es seleccionado
 * @param newHighLightColor color cuando es selecionado el concepto
 */
public void setHighLightColor(Color newHighLightColor) {
        highLightColor = newHighLightColor;
}

/** pone  color a la letra del concepto
 * @param newFontColor tipo de letra para el concepto
 */
public void setFontColor(Color newFontColor) {
        fontColor = newFontColor;
}

/** regresa el nombre del concepto
*@return name nombre del concepto
*/
public String getName() {
        return name;
}

/** regresa el texto agregado al concepto
*@return description agregada al concepto
*/
public String getDescription() {
                        return this.description;
        }

/** regresa el url
*@return url liga agregada al concepto
*/
        public String getUrl() {
                        return this.url;
        }

/** regresa los sinónimos
*@return synonyms es un vector de sinónimos, regresa todos los elementos del     vector     */
public Enumeration getSynonyms() {
    return synonyms.elements();
```

```
    }

/** regresa el vector de sinónimos *
 * @return synonyms
 */
    public Vector getSynonymsVector() {
                    return synonyms;
        }

/** regresa el número de sinonimos del concepto
 * @return synonyms.size
 */
        public int getNumberOfSynonyms() {
                    return synonyms.size();
        }

    /** regresa si tiene mascara  el concepto
     * @return masked
     */
    public boolean isMasked() {
      return masked;
    }

    /** regresa si tiene marca el concepto
     * @return flag
     */
    public boolean isFlaged() {
      return flag;
    }

    /** regresa el nombre del archivo de la imagen agregada
     * @return imageFileName  */
    public String getImageFileName () {
          return imageFileName;
    }

    /** regresa el nombre del archivo sonido agregado
     * @return soundFileName  */
    public String getSoundFileName () {
          return soundFileName;
    }

    /** regresa el tipo de letra del concepto
     * @param fontConcept */
     public Font getFontConcept(){
       return fontConcept;
     }


    /** regresa el color del concepto
     * @return color
     */
          public Color getColor() {
                    return color;
          }

    /** regresa el color cuando el concepto es seleccionado
     *@return hightColor*/
          public Color getHighLightColor() {
                    return highLightColor;
          }

    /** regresa el color de la letra
     *@return fontColor
     */
    public Color getFontColor() {
```

```
        return fontColor;
    }

    /** agrega sinónimos al concepto
     *@param newSynony el nombre del sinónimo a agregar
     */
     public void addSynonym(String newSynonym) {
      int synIndex = synonyms.indexOf(newSynonym);
       if (synIndex == -1)
             synonyms.addElement(newSynonym);
    }

    /** borra el sinónimo
     * @param synonym el nombre del sinónimo
     */
     public void deleteSynonym(String synonym) {
       synonyms.removeElement(synonym);
     }

    public String toString(){
     return(name);
    }

     /** regresa la longitud del nombre del concepto */
                    public int getNameLength() {
                            if (name.length() < 6)
                                    return 6;
                            return name.length();
                    }

} /* fin de la clase Concept */
```

9.2. Clase Relación (Relation)

```
/* Relation.java */
package SemNet.DataRepresentation;
import java.io.Serializable;
/**
 * Esta clase contiene todas las características de las relaciones
 *@see package SemNet.DataRepresentation
 *@see java.io.Serializable
 */
  public class Relation implements Serializable {
   // sentido de la relación
   public static final int CENTRAL_TO_RELATED = 0; // c. central a c. relacionado
   public static final int RELATED_TO_CENTRAL = 1; // c. relacionado a c. central
   // sentido de la flecha
   private Arrow fromCentralConcept  = null; // desde el concepto central al relacionado
   private Arrow toCentralConcept   = null; // al concepto central desde el relacionado

   /** crea una relación con el nombre de la relacion desde el concepto central
    * y la relación que va  al concepto central */
    public Relation(String fromCentral, String toCentral) {
             fromCentralConcept = new Arrow(fromCentral);
             toCentralConcept = new Arrow(toCentral);
          }

          /** crea una relación sin nombre */
          public Relation() {
                  this("");
          }
```

```
/** crea la relación con el mismo nombre para ambos sentidos
 * (relación simétrica) */
public Relation(String relName) {
        this(relName,relName);
}

/** crea la relación con ambos sentidos desde el concepto central y
  * hacia el concepto central */
public Relation(Arrow fromCentral, Arrow toCentral) {
                fromCentralConcept = fromCentral;
                toCentralConcept = toCentral;
        }

/** crea una relación con el mismo nombre (relación simétrica)
 *  con la diferencia que el parámetro viene desde la flecha
 */
public Relation(Arrow arrow) {
  this(arrow,arrow);
}

/** pone una nueva relación desde el concepto central */
  public void setArrowFromCentral(Arrow newArrow) {
        fromCentralConcept = newArrow;
  }

/** pone una nueva relación desde el concepto central */
  public void setArrowToCentral(Arrow newArrow) {
        toCentralConcept = newArrow;
  }

/** pone la relación en dos los sentidos */
  public void setArrows(Arrow fromRelation, Arrow toRelation) {
        fromCentralConcept = fromRelation;
        toCentralConcept = toRelation;
  }

/** regresa el sentido de la relación (flecha)
  * CENTRAL_TO_RELATED = desde el concepto central al concepto relacionado
  * REALTED_TOCENTRAL = desde el concepto relacionado al concepto central
  */
public Arrow getArrow(int arrowDirection) {
  if (arrowDirection == CENTRAL_TO_RELATED)
    return fromCentralConcept;
  if (arrowDirection == RELATED_TO_CENTRAL)
    return toCentralConcept;
  return null;
}

/** regresa la flecha o relación que va desde el concepto central hacia el
  * concepto relacionado
  */
public Arrow getArrowFromCentral() {
  return fromCentralConcept;
}

/** regresa la flecha o relación que va hacia el concepto central (es decir parte
  * del concepto relacionado
  */
public Arrow getArrowToCentral() {
  return toCentralConcept;
}

/** regresa el nombre de la relación que va desde el concepto central */
  public String getFromCentralName() {
        return fromCentralConcept.getName();
```

```
    }

  /** regresa el nombre de la relación que viene desde el concepto relacionado */
  public String getToCentralName() {
     return toCentralConcept.getName();
  }


  /* regresa el nombre de la relación que va desde el concepto relacionado */
  public String toString() {
     return fromCentralConcept.getName();
  }

} /* fin de clase Relation */
```

9.3. Clase Flecha (Arrow)

```
/* Arrow.java */
package SemNet.DataRepresentation;

import java.awt.Color;
import java.io.Serializable;

/**
 * Clase Arrow que representa un camino de la relación entre dos
 * Conceptos. Esta clase contiene todas las propidades de la Relación se representa
 * mediante una flecha.
 *
 * @see java.io.Serializable
 * @see java.awt.Color
 * @see javax.swing.ImageIcon
*/
public class Arrow implements Serializable {

   /**Representa las 12 horas de la posición de la flecha en pantalla */
   public static final int POSITION_12        = 0;
   /**Representa las 1 horas de la posición de la flecha en pantalla */
   public static final int POSITION_1                   = 1;
   /**Representa las 2 horas de la posición de la flecha en pantalla  */
   public static final int POSITION_2                   = 2;
   /**Representa las 3 horas de la posición de la flecha en pantalla */
   public static final int POSITION_3                   = 3;
   /**Representa las 4 horas de la posición de la flecha en pantalla */
   public static final int POSITION_4                   = 4;
   /**Representa las 5 horas de la posición de la flecha en pantalla */
   public static final int POSITION_5                   = 5;
   /**
    * Representa las 6 horas de la posición de la flecha en pantalla */
   public static final int POSITION_6                   = 6;
   /**Representa las 7 horas de la posición de la flecha en pantalla */
   public static final int POSITION_7                   = 7;
   /**Representa las 8 horas de la posición de la flecha en pantalla */
   public static final int POSITION_8                   = 8;
   /**Representa las 9 horas de la posición de la flecha en pantalla */
           public static final int POSITION_9                   = 9;
   /**Representa las 10 horas de la posición de la flecha en pantalla */
           public static final int POSITION_10           = 10;
   /**Representa las 11 horas de la posición de la flecha en pantalla */
   public static final int POSITION_11          = 11;

   /**nombre de un sentido de la relación */
   private String              name                = null;
   /**descripción texto agregado a la relación */
   private String                          description            = null;
```

```
/**posición en pantalla de una flecha o relación */
private int                         position               = POSITION_12;

/**
 * Mascara o marca, información acerca de la relación. Si el valor es
 * verdadero, el usuario no pede visualizar el nombre de la relación
 * en el desplegador gráfico. Si el valor de la marca es falso el nombre de
 *la relación se puede visualizar */
private boolean                     masked                 = false;

/**
 * Nombre del archivo de la imagen agregado a la relación
 */
 private String          imageFileName     = null;
/**
 * nombre del archivo de sonido agregado a la relación
 */
  private String    SoundFileName     = null;

/**
 * color para la caja donde se encuentra el nombre de la flecha
 */
  private Color                       color                          = Color.white;

/**
 * highlightcolor es el color usado cuando la relación o flecha es seleccionada
 */
  private Color                       highLightColor     = Color.lightGray;

/**
 * color usado para imprimir en pantalla el nombre de la relación
 * (fuente de la relación)
 */
  private Color          fontColor     = Color.black;

/**
 * crea una nueva flecha con el nombre de la relación
 * @param label nombre de la relación
 */
  public Arrow(String label) {
          this.name = label;
  }

/**
 * crea una nueva flecha con el nombre de la relación y la posición
 * en pantalla
 * @param label nombre de la relación
 * @param newPos la posición en pantalla
 */
  public Arrow(String label, int newPos) {
          this(label);
          position = newPos;
  }

  /**
   * crea una nueva flecha con el nombre de la relación, la nueva descripción
   * y la nueva posición en pantalla
   * @param label nombre de la relación
   * @param newDesc el texto agregado para la relación
   */
   public Arrow(String label, String newDesc, int newPos) {
      this(label,newPos);
      description = newDesc;
   }

   /**
```

```
 * pone el nombre de la flecha o relación
 * @param  label nombre de la relación
 */
public void setName(String label) {
                        this.name = label;
          }

/**
 * pone el texto agregado a la relación
 * @param text la descripción a hacer agregada en la relación
 */
public void setDescription(String text) {
                        this.description = text;
          }

/**
 * pone la posición en pantalla de la flecha
 * @param newPos la nueva posición en pantalla de la relación
 */
public void setPosition(int newPos) {
                        this.position = newPos;
          }

/**
 * pone la marca de la flecha si la marca es verdadera se encuentra
 * marcada, si es falsa la flecha se encuentra desmarcada o sin marca
 * @param flag marca de tipo boleana
 */
public void setMasked(boolean flag) {
                        this.masked = flag;
          }


public void setImageFileName(String fname){
               imageFileName = fname;
}

public void setSoundFileName(String sName){
        SoundFileName = sName;
}

/**
 * pone el color de la flecha
 * @param newColor el nuevo color para la relación
 */
  public void setColor(Color newColor) {
                        this.color = newColor;
          }

/**
 * pone de un nuevo color la relación cuando es seleccionada
 * @param newHighLightColor color cuando se selecciona la relación
 */
  public void setHighLightColor(Color newHighLightColor) {
                        this.highLightColor = newHighLightColor;
          }

  /**
   * pone el color de la letra usado en la relación
   * @param newFontColor color para la letra o fuente de la relación
   */
  public void setFontColor(Color newFontColor) {
           fontColor = newFontColor;
  }

  /**
```

```
 * regresa la cadena que representa el nombre de la relación
 * @return name nombre de la relación
 */
public String getName() {
   return(name);
}

/**
 * regresa la cadena con el texto agregado de la relación
 * @return description
 */
public String getDescription() {
   return(description);
}

/**
 * regresa la posición de la flecha
 * @return position
 */
public int getPosition() {
   return position;
}

/**
 * regresa el estado de la marca. si el valor es verdadero
 * indica que la relación se encuentra marcada, si el valor
 * de la marca es falso se encuentra desmarcada la relación
 * @return masked
 */
public boolean isMasked() {
    return masked;
}

public String getImageFileName(){
            return imageFileName;
 }

public String getSoundFileName(){
            return SoundFileName;
 }


 /**
  * regresa el color usado para pintar la flecha
  * @return color
  */
 public Color getColor() {
          return this.color;
  }

/**
* regresa el color usado cuando la flecha es selecciona
* @return highLightColor
*/
  public Color getHighLightColor() {
                     return this.highLightColor;
          }

  /**
   * regresa el color de la letra para la flecha
   * @return fontColor
   */
  public Color getFontColor() {
           return fontColor;
  }
```

```
/**
 * regresa el nombre de la flecha o relación
 * @return name
 */
public String toString() {
   return(name);
}

/**
 * regresa la longitud de la cadena que representa el nombre de la flecha.
 * Si la longitud de name es menor que 6, este método regresa 6.
 *
 * este método es para manejar el despliegue del botón de la relación en pantalla
 * @return longitud de la cadena
 */
public int getNameLength() {
        if (name.length() < 6)
                return 6;
        return name.length();
}

} /* fin de la clase Arrow */
```

9.4. Clase Nodo (Node)

```
/* Node.java */
package SemNet.DataRepresentation;

import java.io.Serializable;
import java.util.Vector;
import java.util.Enumeration;

/** Clase Node
 * Esta clase representa el nodo para la red semántica
 * @see package SemNet.DataRepresentation
 * @see java.util.Vector
 * @see java.util.Enumeration
*/
public class Node implements Serializable {

   private int centralConceptIndex = -1; // índice del concepto central
        private Vector instances; // vector de instancias

        /** constructor inicializa un vector de instancias y el índice del concepto central */
   public Node(int conceptIndex) {
        centralConceptIndex = conceptIndex;
      instances = new Vector();
   }

   /** pone el índice del concepto central */
   public void setCentralConceptIndex(int newIndex) {
      centralConceptIndex = newIndex;
   }

   /** pone el vector de instancias */
   public void setInstancesVector(Vector newInstancesVector) {
        instances = newInstancesVector;
   }

   /** pone una instancia en el vector instances, en el índice especificado por index */
   public void setInstance(Instance newInstanceValue,int index) {
```

```
        if (index < instances.size())
            instances.setElementAt(newInstanceValue,index);
}

/** pone la instancia con los índices de concepto, relación e instancia */
public void setInstance(int relIndex, int conIndex, Instance newInst) {
  Enumeration enum = this.instances.elements();
  Instance instance = null;
  int instanceIndex = 0;

    while (enum.hasMoreElements()) {
      instance = (Instance)enum.nextElement();
      if (relIndex == instance.getRelationIndex() && conIndex == instance.getConceptIndex())
            setInstance(newInst,instanceIndex);
      instanceIndex++;
    }
}

/** regresa el índice del concepto central
*@return centraConceptIndex*/
public int getCentralConceptIndex() {
   return centralConceptIndex;
}

/** regresa los elementos de la instancia */
public Enumeration getInstances() {
   return instances.elements();
}

/** regresa el vector de instancias
*@return instances*/
public Vector getInstancesVector() {
   return instances;
}

/** regresa la instancia, busca en base el índice*/
public Instance getInstance(int instanceIndex) {
        if (instanceIndex < instances.size())
            return (Instance)instances.elementAt(instanceIndex);
   return null;
}

/** regresa la instancia que comienza en el cocepto relacionado */
public Instance getInstanceFromRelatedConcept(int conIndex) {
  Enumeration insts = getInstances();
  Instance ins = null;
    while (insts.hasMoreElements()) {
      ins = (Instance)insts.nextElement();
      if (ins.getConceptIndex() == conIndex)
          return ins;
    }
    return null;
}

/** regresa la instancia desde la relación */
public Instance getInstanceFromRelation(int relIndex) {
  Enumeration insts = getInstances();
  Instance ins = null;
    while (insts.hasMoreElements()) {
      ins = (Instance)insts.nextElement();
      if (ins.getRelationIndex() == relIndex)
          return ins;
    }
    return null;
}
```

```
/** regresa la posición del nodo */
public int[] getNodePositions() {
 int pos[] = { -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1};
  Enumeration insts = getInstances();
  Instance conceptLinked = null;
  while (insts.hasMoreElements()) {
     conceptLinked = (Instance)insts.nextElement();
     pos[conceptLinked.getPosition()] = conceptLinked.getConceptIndex();
  } /* end while */
  return pos;
}

/** agrega una instancia con índice de relación, concepto y dirección de la relación */
public void addInstance(int relIndex, int conIndex, int relDir) {
        instances.addElement(new Instance(relIndex,relDir,conIndex));
}

/** agrega instancia con índice de relación, concepto, dirección y posición */
public void addInstance(int relIndex, int conIndex, int relDir, int relPos) {
   instances.addElement(new Instance(relIndex,conIndex,relDir,relPos));
}

/** agrega instancia */
public void addInstance(Instance instance) {
   instances.addElement(instance);
}

/** borra la instancia */
public void deleteInstance(int relIndex, int conIndex) {
 Enumeration enum = this.instances.elements();
 Instance instance = null;

   while (enum.hasMoreElements()) {
      instance = (Instance)enum.nextElement();
      if (relIndex == instance.getRelationIndex() &&
         conIndex == instance.getConceptIndex())
            this.instances.removeElement(instance);
      }
}

/** borra la instancia por la relación */
public void deleteInstanceByRelation(int relIndex) {
 Enumeration enum = this.instances.elements();
 Instance instance = null;

   while (enum.hasMoreElements()) {
      instance = (Instance)enum.nextElement();
      if (relIndex == instance.getRelationIndex() )
         this.instances.removeElement(instance);
   }
}

} /* End of class Node */
```

9.5. Clase Instancia (Instance)

```
/* Instance.java */
package SemNet.DataRepresentation;

import java.io.Serializable;

/** clase Instance
 * esta clase contiene todas las propiedades de las intancias
```

```
 * instancia = concepto relación concepto
 * @see java.io.Serializable
*/
public class Instance implements Serializable {

   private int relationIndex = -1; // índice de la relación
   private int conceptIndex = -1; // índice del concepto
   private int direction = Relation.CENTRAL_TO_RELATED; // dirección o sentido de la relación
   private int position = Arrow.POSITION_12;// posición en pantalla

/** constructor Instance crea los índices de las relaciones y conceptos */
   public Instance(int relIndex, int conIndex) {
        relationIndex = relIndex;
        conceptIndex = conIndex;
   }

/** constructor Instance crea los índices de las relaciones y conceptos y dirección de la relación ( central a relacionado o relacionado
a central ) */
   public Instance(int relIndex, int conIndex, int dir) {
            this(relIndex,conIndex);
      direction = dir;
   }

/** constructor Instance crea los índices de las relaciones y conceptos, dirección de la relación (central a relacionado o relacionado a
central)
      posición en pantalla de la instancia*/
   public Instance(int relIndex, int conIndex, int dir, int pos) {
      this(relIndex,conIndex,dir);
      position = pos;
   }

 /** pone el índice del concepto
   *@param conIndex*/
   public void setConceptIndex(int conIndex) {
            conceptIndex = conIndex;
   }

 /* pone el índice de la relación
  *@param relIndex*/
   public void setRelationIndex(int relIndex) {
            relationIndex = relIndex;
   }

 /** pone la dirección de la instancia */
   public void setDirection(int newDir) {
            direction = newDir;
   }

 /** pone la posición en pantalla
   *@param newPos*/
   public void setPosition(int newPos) {
            position = newPos;
   }

 /** regresa el índice del concepto
   *@return conceptIndex*/
   public int getConceptIndex() {
      return conceptIndex;
   }

  /** regresa el índice de la relación
   *@return relationIndex*/
   public int getRelationIndex() {
      return relationIndex;
   }
```

```
/** regresa la dirección
  *@return direction*/
  public int getDirection() {
           return direction;
  }

/** regresa la posición
  *@return getPosition*/
  public int getPosition() {
     return position;
  }

} /* fin de clase Instance */
```

9.6. Clase Red Semántica (SemanticNet)

```
/* SemanticNet.java */
package SemNet.DataRepresentation;

import java.util.Vector;
import java.util.Enumeration;
import java.io.Serializable;
import java.awt.Graphics;

/** Clase SemanicNet esta clase contiene todas las propiedades de la red semántica
  * conceptos relaciones instancias.
  * @see package SemNet.DataRepresentation
  * @see java.util.Vector
  * @see java.util.Enumeration
  * @see java.io.Serializable
  * @see java awt.Graphics
  */
public class SemanticNet implements Serializable {

    private String name = null; // nombre de la red
    private Vector concepts = null; // vector de conceptos
    private Vector relations = null; // vector de relaciones
    private Vector netNodes = null;            // vector de nodos de la red

   /** constructor crea una red semántica sin nombre */
   public SemanticNet() {
      this("Untitled");
   }

   /** constructor crea una red semantica con el nombre del archivo de la red*/
   public SemanticNet(String fileName) {
            name = fileName; // nombre del la red semántica
            netNodes = new Vector();
            concepts = new Vector();
            relations = new Vector();
            this.addNode(new Concept(fileName));
   }

   /** pone nombre a la red semántica */
   public void setName(String newName) {
      name = newName;
   }

   /** pone concepto  en el índice especificado*/
   public void setConcept(Concept con, int conIndex) {
      if (conIndex >=0 && conIndex < concepts.size())
         concepts.setElementAt(con,conIndex);
   }
```

```
/** pone relación en el índice especificado */
 public void setRelation(Relation rel, int relIndex) {
    if (relIndex >=0 && relIndex < concepts.size())
       relations.setElementAt(rel,relIndex);
 }

/** regresa el nombre de la red semántica */
 public String getName() {
    return name;
 }

/** regresa el vector de conceptos (la lista de conceptos)*/
 public Vector getConceptsVector() {
    return concepts;
 }

/** regresa el vector de relaciones (la lita de relaiones)*/
 public Vector getRelationsVector() {
    return relations;
 }

/** regresa el vector de nodos de la red (instancas) */
 public Vector getNetNodesVector() {
    return netNodes;
 }

/** regresa el concepto, busca en base al nombre del concepto */
 public Concept getConcept(String conName) {
   int conIndex = getConceptIndex(conName); // buscamos el índice del concepto
    if (conIndex != -1) // lo encontro
       return (Concept)concepts.elementAt(conIndex);
    else
       return null; // no lo encontro
 }

 /** regresa el concepto, busca en base al índice */
 public Concept getConcept(int n) {
    if (n >=0 && n < concepts.size())
       return (Concept)concepts.elementAt(n);
    else
       return null;
 }

 /** regresa el índice del concepto */
 public int getConceptIndex(Concept con) {
    return concepts.indexOf(con);
 }

 /** regresa el índice del concepto, busca en base al nombre del concepto */
 public int getConceptIndex(String conName) {
   Enumeration enum = this.concepts.elements();
   Concept storedConcept = null;
    while (enum.hasMoreElements()) {
       storedConcept = (Concept)enum.nextElement();
       if (conName.equals(storedConcept.getName())) // si el concepto se encuentra
          return concepts.indexOf(storedConcept);  // regresamos el índice
    }

    return -1; // no esta el concepto
 }

 /** regresa la relación, busca en base al nombre de la relación */
 public Relation getRelation(String relName) {
   int relIndex = getRelationIndex(relName);
    if (relIndex != -1)
```

```
        return (Relation)relations.elementAt(relIndex);
     else
        return null;
  }

/** regresa la relación, busca en base a los dos nombres de la relación */
  public Relation getRelation(String relNameCentral, String relNameRelated) {
    int relIndex = getRelationIndex(relNameCentral,relNameRelated);
     if (relIndex != -1)
        return (Relation)relations.elementAt(relIndex);
     else
        return null;
  }

/** regresa la relación, busca en base al índice */
  public Relation getRelation(int n) {
     if (n >= 0 && n < relations.size())
        return (Relation)relations.elementAt(n);
     else
        return null;
  }

  /** regresa el índice de la relación */
  public int getRelationIndex(Relation rel) {
     return relations.indexOf(rel);
  }


/** regresa el índice de la relación, busca en base al nombre de la relación*/
  public int getRelationIndex(String relName) {
    Enumeration enum = this.relations.elements();
    Relation storedRelation = null;
     while (enum.hasMoreElements()) {
        storedRelation = (Relation)enum.nextElement();
        if (relName.equals(storedRelation.getArrowFromCentral().getName())) // si la encuentra
           return relations.indexOf(storedRelation);
     }
     return -1; // no la encontro
  }

  /** regresa el índice de la relación de cualquier dirección, busca en base al nombre
   * de la relación */
  public int getRelationIndexAnyDirection(String relName) {
    Enumeration enum = this.relations.elements();
    Relation storedRelation = null;
     while (enum.hasMoreElements()) {
        storedRelation = (Relation)enum.nextElement();
        if (relName.equals(storedRelation.getArrowFromCentral().getName()) ||
            relName.equals(storedRelation.getArrowToCentral().getName()))
           return relations.indexOf(storedRelation);
     }
     return -1;
  }

/* regresa el índice de la relación, busca el nombre de las dos relaciones
 * relNameCentral = nombre de la relación que va del concepto central al concepto relacionado
 * relNameRelated = nombre de la relación que va del concepto relacionado al concepto central
 */
  public int getRelationIndex(String relNameCentral, String relNameRelated) {
    Enumeration enum = this.relations.elements();
    Relation storedRelation = null;
     while (enum.hasMoreElements()) {
        storedRelation = (Relation)enum.nextElement();
        if (relNameCentral.equals(storedRelation.getArrowFromCentral().getName()) &&
            relNameRelated.equals(storedRelation.getArrowToCentral().getName()) )
           return relations.indexOf(storedRelation);
```

```
    }

    return -1;
  }

  /** regresa el nodo o concepto, busca en base al nombre */
  public Node getNode(String nodeName) {
   int nodeIndex = getNodeIndex(nodeName);
    if (nodeIndex != -1)
       return getNode(nodeIndex);
    else
       return null;
  }

  /** regresa el nodo, busca en base al índice */
  public Node getNode(int n) {
    return (Node)netNodes.elementAt(n);
  }

  /** regresa el índice del nodo, busca en base al concepto */
  public int getNodeIndex(Concept con) {
    return getNodeIndex(con.getName());
  }

  /** regresa el índice del nodo, busca en base al nombre */
  public int getNodeIndex(String nodeName) {
   Enumeration enum = netNodes.elements();
   Node storedNode = null;
   int nodeIndex = 0;
    while (enum.hasMoreElements()) {
       storedNode = (Node)enum.nextElement();
       if (nodeName.equals(getConcept(storedNode.getCentralConceptIndex()).getName())) {
          return nodeIndex;
       }
       nodeIndex++;
    }
    return -1;
  }

  /** regresa la instancia, busca por el índice del nodo y el nombre de la relación
  */
  public Instance getInstanceNodeFromRelation(int nodeIndex, String relName) {
   Enumeration insts = ((Node)(netNodes.elementAt(nodeIndex))).getInstances();
   Instance ins = null;
   Relation rel = null;

    while (insts.hasMoreElements()) {
       ins = (Instance)insts.nextElement();
       rel = getRelation(ins.getRelationIndex());
       if (relName.equals(rel.getFromCentralName()) ||
           relName.equals(rel.getToCentralName()) )
          return ins;
    }
    return null;
  }

  /** regresa el número de nodos o conceptos de la red */
  public int getNumberOfNodes() {
    return netNodes.size();
  }

  /** agrega un concepto */
  public void addConcept(Concept con) {
      int conIndex = getConceptIndex(con.getName());// busca si ya esta el concepto en la lista de conceptos
          if (conIndex == -1) { // si no esta, entonces
             concepts.addElement(con); // agregamos el concepto en la lista
```

```
            addNode(con); // lo agregamos en la lista de nodos
        }
}

/** agregamos un concepto en base al nombre  */
 public void addConcept(String conName) {
  int conIndex = getConceptIndex(conName); // busca si esta en la lista
  Concept newCon = null;
   if (conIndex == -1) {
         newCon = new Concept(conName);// se crea el concepto
      concepts.addElement(newCon); // se agrega a la lista
      addNode(newCon);
   }
 }

/** agregamos la relación */
 public void addRelation(Relation rel) {
  int relIndex = getRelationIndex(rel.getFromCentralName());
   if (relIndex == -1)
            relations.addElement(rel);
 }

 /** agregamos la relación en base al nombre */
 public void addRelation(String relName) {
  int relIndex = getRelationIndex(relName); // se busca si existe la relación
   if (relIndex == -1)
      relations.addElement(new Relation(relName));
 }

/** agregamos las dos relaciones relNameCentral = CENTRAL_AL_RELACIONADO
   relNameERelated = RELACIONADO_AL_CENTRAL  */
 public void addRelation(String relNameCentral, String relNameRelated) {
  int relIndex = getRelationIndex(relNameCentral,relNameRelated);
   if (relIndex == -1)
      relations.addElement(new Relation(relNameCentral,relNameRelated));
 }

 /** agrega el nodo o concepto en la lsita de nodos de la red */
 public void addNode(Concept con) {
    int conIndex = -1;
    int nodeIndex = -1;
    conIndex = getConceptIndex(con.getName()); // busca el índice del concepto
    if (conIndex == -1) { // no esta en la lista de conceptos entonce se agrega a las listas de conceptos y de nodos
             concepts.addElement(con);
            conIndex = concepts.indexOf(con);
            netNodes.addElement(new Node(conIndex));
     }
     else {
                    nodeIndex = getNodeIndex(con.getName()); // se busca en la lista de nodos
                    if (nodeIndex == -1) {
                       netNodes.addElement(new Node(conIndex));
                    }
            }
     }


 /** agrega el nodo en base al nombre del concepto */
 public void addNode(String conName) {
  int conIndex = -1;
  int nodeIndex = -1;
  Concept newCon = null;
   conIndex = getConceptIndex(conName);
   if (conIndex == -1) {
         newCon = new Concept(conName);
      concepts.addElement(newCon);
      conIndex = concepts.indexOf(newCon);
```

```
      netNodes.addElement(new Node(conIndex));
    }
    else {
      nodeIndex = getNodeIndex(conName);
      if (nodeIndex == -1) {
        netNodes.addElement(new Node(conIndex));
      }
    }
  }


/** agrega una instancia
   nodeName= Nombre del concepto central
   relName = Nombre de la relación
   relatedConName = Nombre del concepto relacionado
 */
public void addInstance(String nodeName, String relName, String relatedConName) {
  int relIndex = getRelationIndex(relName);
         if (relIndex == -1)
     addInstance(nodeName,relName,relatedConName,Arrow.POSITION_12,Arrow.POSITION_12);
   else {
     Relation rel = getRelation(relIndex);
     addInstance(nodeName,relName,relatedConName,
        rel.getArrowFromCentral().getPosition(),
        rel.getArrowToCentral().getPosition());
   }
 }

/** agrega una instancia
   nodeName = nombre del concepto central
   relNameDown = nombre de la relacion CENTRAL_AL_RELACIONADO
   relNameUp = nombre de la relación RELACIONADO_AL_CENTRAL
   relatedConName = nombre del concepto relacionado
   positionDown = posición en pantalla del concepto relacionado
   positionUp = posición en pantalla del concepto central
 */
public void addInstance(String nodeName, String relNameDown, String relNameUp,
      String relatedConName, int positionDown, int positionUp) {

  int centralConIndex = getConceptIndex(nodeName);
  int relatedConIndex = getConceptIndex(relatedConName);
  int relationIndex = getRelationIndex(relNameDown,relNameUp);
  int centralNodeIndex = -1, relatedNodeIndex = -1;

   if (centralConIndex == -1) {
     this.addConcept(new Concept(nodeName));
     centralConIndex = getConceptIndex(nodeName);
   }
   if (relatedConIndex == -1) {
          this.addConcept(new Concept(relatedConName));
     relatedConIndex = getConceptIndex(relatedConName);
   }
   if (relationIndex == -1) {
          this.addRelation(new Relation(relNameDown,relNameUp));
     relationIndex = getRelationIndex(relNameDown,relNameUp);
   }

   centralNodeIndex = getNodeIndex(nodeName);
   relatedNodeIndex = getNodeIndex(relatedConName);

   if (centralNodeIndex == -1) {
     addNode(nodeName);
     centralNodeIndex = getNodeIndex(nodeName);
   }
   if (relatedNodeIndex == -1) {
          addNode(relatedConName);
```

```
      relatedNodeIndex = getNodeIndex(relatedConName);
    }

    addInstanceToNode(centralNodeIndex, new Instance(relationIndex,relatedConIndex,
      Relation.CENTRAL_TO_RELATED,positionDown));
    addInstanceToNode(relatedNodeIndex, new Instance(relationIndex,centralConIndex,
      Relation.RELATED_TO_CENTRAL,positionUp));
  }

 /** agrega instancia */
  public void addInstance(String nodeName, String relName,
        String relatedConName, int positionDown, int positionUp) {
         int centralConIndex = getConceptIndex(nodeName);
         int relatedConIndex = getConceptIndex(relatedConName);
         int relationIndex = getRelationIndex(relName);
         int centralNodeIndex = -1, relatedNodeIndex = -1;

        if (centralConIndex == -1) {
           this.addConcept(new Concept(nodeName));
          centralConIndex = getConceptIndex(nodeName);
        }

           if (relatedConIndex == -1) {
           this.addConcept(new Concept(relatedConName));
          relatedConIndex = getConceptIndex(relatedConName);
        }

        if (relationIndex == -1) {
           this.addRelation(new Relation(relName));
           relationIndex = getRelationIndex(relName);
        }
      centralNodeIndex = getNodeIndex(nodeName);
     relatedNodeIndex = getNodeIndex(relatedConName);

     if (centralNodeIndex == -1) {
        addNode(nodeName);
        centralNodeIndex = getNodeIndex(nodeName);
     }

      if (relatedNodeIndex == -1) {
        addNode(relatedConName);
        relatedNodeIndex = getNodeIndex(relatedConName);
     }

     addInstanceToNode(centralNodeIndex, new Instance(relationIndex,relatedConIndex,
        Relation.CENTRAL_TO_RELATED,positionDown));
     addInstanceToNode(relatedNodeIndex, new Instance(relationIndex,centralConIndex,
        Relation.RELATED_TO_CENTRAL,positionUp));
  }

  /** regresa el nombre de la red */
  public String toString() {
     return name;
  }

  /** agrega instancia al nodo, se manda como parámetro el  índice del concepto central
     y la instancia */
  private void addInstanceToNode(int nodeIndex, Instance instance) {
    Node node = (Node)netNodes.elementAt(nodeIndex);
     node.addInstance(instance);
  }
} /* fin de clase SemanticNet */
```