Sistemas Informáticos
Curso 2004-2005

Herramienta CASE hipermedia. Plumbingmatic

Alejandro Blanco Martín
David Curieses Chamón
Óscar Ortega Mejías

Dirigido por:
Prof. Antonio Navarro Martín
Dpto. Sistemas Informáticos y Programación

Facultad de Informática
Universidad Complutense de Madrid

INDICE

Resumen

Esta memoria describe el trabajo realizado durante el desarrollo del proyecto *Herramienta CASE hipermedia. Plumbingmatic* en el contexto de la asignatura *Sistemas Informáticos* de la *Universidad Complutense de Madrid*.

Desde un punto de vista estructural, la memoria describe la herramienta PlumbingMatic, proporciona un modelo UML del código, e incluye un anexo con la especificación de requisitos software de PlumbingMatic.

La herramienta PlumbingMatic es una herramienta CASE para la construcción de modelos Pipe de aplicaciones hipermedia. Dicha herramienta consta de tres grandes módulos. El primer módulo se centra en la creación de diagramas Pipe. El segundo módulo se centra en la generación de representaciones XML de dichos diagramas Pipe. Finalmente, el tercer módulo traduce las representaciones XML de los diagramas Pipe en prototipos ejecutables.

Abstract

This report describes the work done during the development of the project *Hypermedia CASE Tool. PlumbingMatic* in the context of the course *Computer Science Systems* at the *Universidad Complutense de Madrid*.

From a structural point of view, this report describes the tool PlumbingMatic, provides an UML model of the code, and includes an annex with the software requirements specification of PlumbingMatic.

The tool PlumbingMatic is a CASE tool intended for the development of Pipe models for hypermedia applications. Such a tool is composed by three modules. The first module is focused on the development of Pipe diagrams. The second module is focused on the generation of XML representations of such Pipe diagrams. Finally, the third module translates the XML representations of Pipe diagrams into running prototypes.

1. Introducción

El *World Wide Web Consortium* define *hipertexto* como texto que no está restringido a ser lineal, es decir, texto que contiene enlaces a otros textos [W3C 95]. De la misma forma, el consorcio define *hipermedia* como hipertexto que no está restringido a ser texto, es decir, hipertexto que por ejemplo, puede incluir gráficos, sonido o videos[1].

El concepto de hipertexto fue introducido por primera vez en el año 1945 cuando Vannevar Bush describió su sistema Memex [Bush 45]. Memex era un sistema que por medios ópticos y electromecánicos permitía la presentación de la información en base a las relaciones semánticas existentes entre sus distintas partes. De esta forma se vencía la limitación de la linealidad del texto.

Cincuenta años después el concepto de hipertexto ha evolucionado bastante. En la actualidad se consideran relacionadas con el hipertexto diferentes áreas como las librerías digitales, la interacción persona computadora o incluso recuperación de información [HT 05].

En el contexto actual, es innegable la importancia que la *World Wide Web* ha tenido en la difusión del hipertexto. Por ejemplo, *Java 2 Platform Standard Edition 5.0 API Specification* [Sun 04] es un hipertexto disponible en la Web consultado a diario por multitud de programadores en todo el planeta. En este sentido cabe destacar la diferencia entre aplicaciones Web e hipertextos. Según Conallen [Conallen 99], una *aplicación Web* es un sistema Web (servidor, red, HTTP, browser) en el que la entrada del usuario (navegación e introducción de datos) afecta al estado del negocio. Así, un sistema para la gestión Web del correo electrónico podríamos considerarla como una aplicación Web, pero no un hipertexto. De la misma forma, un hipertexto desplegado en CD-ROM sería un hipertexto no Web.

En el año 2002, y en el seno de las aplicaciones hipermedias, fue presentada la tesis *Conceptualización, Prototipado y Proceso de Aplicaciones Hipermedia* [Navarro 02b]. En dicha tesis se proponía *Pipe* [Navarro 04b], una notación gráfica formalizada para caracterizar la etapa de conceptualización de aplicaciones hipermedia [Fraternali 99]. Además se presentaba una técnica de prototipado basada en XML y Java que permitía la traducción de los diagramas Pipe a un formato adecuado a la generación automática de prototipos [Nanard 95]. Finalmente, todas estas técnicas se integraban en dos modelos de procesos, *Plumbing* y *PlumbingXJ* [Navarro 04a].

En dicha tesis se proponía la construcción de *PlumbingMatic* una herramienta CASE (*Computer-Aided Software Engineering*) que fuese capaz de crear diagramas de la notación Pipe y de representarlos automáticamente en formato XML. Dicha herramienta debería integrarse con el motor de generación

[1] En esta memoria, y como es habitual en el área, se utilizará de forma indistinta los términos hipertexto e hipermedia.

de prototipos *GAP* (Generador Automático de Prototipos) presentado en la tesis.

Durante el curso académico 2004-2005 en la Facultad de Informática de la Universidad Complutense de Madrid se propuso un proyecto de la asignatura Sistemas Informáticos cuya finalidad era la construcción de la herramienta CASE PlumbingMatic. Precisamente, entre otros, esta memoria recoge el diseño, especificación de requisitos software y manual de usuario de la herramienta construida en el contexto de dicho proyecto.

En esta memoria hemos decidido no incluir ningún capítulo que presente la notación visual Pipe, ya que puede encontrarse en la literatura. Una breve descripción de toda la aproximación puede encontrarse en [Navarro 02a]. [Navarro 04b] describe con mayor nivel de detalle toda la aproximación, formalización incluida, mientras que [Navarro 04a] omite dicha formalización y se centra en los aspectos dinámicos del prototipado.

Por lo tanto, la estructura de esta memoria es la siguiente. En el capítulo 2 se muestra una descripción de Plumbing Matic. El capítulo 3 se describen los casos de uso y diagramas de actividades [OMG 04] de la herramienta CASE. El capítulo 4 se centra en describir diagramas de clases e interacción [OMG 04]. Hemos decidido no proporcionar modelo de análisis y diseño por separado [Jacobson 99] para no incurrir en una memoria demasiado extensa. En su lugar, hemos optado por separar casos de uso/actividades de clases/interacción. También hemos optado por no proporcionar diagramas de transición de estados, componentes o despliegue [OMG 04] porque añadían poca información relevante sobre el modelo de la aplicación. Finalmente, la sección 5 presenta las conclusiones. La memoria incluye también una sección con referencias (sección 6), y unos apéndices con la especificación de requisitos software en formato IEEE 830 [IEEE 830].

1.1 Descripción General

Digamos que esta aplicación tiene tres puntos de gran interés, el primero es la notación gráfica, es decir los mecanismos que se le ofrece al usuario para poder diseñar desde un interfaz hasta el contenido de una aplicación hipermedia, además por supuesto de poderlos relacionar. El segundo punto de interés de la herramienta es que se permite guardar y cargar interfaces, contenidos y relaciones previamente diseñadas por dicho usuario. En este punto es importante indicar que el formato de los ficheros que se guardan es xml, lo que implica que no vale cualquiera ya que tiene que seguir una dtd definida. Además se ofrece la posibilidad de guardar en dos formatos cada apartado diseñado en la aplicación, uno es para guardarlo en formato que entiende el motor de prototipado, del cual hablaremos ahora y el otro simplemente para cargarlo en la herramienta. El tercer punto de interés de esta herramienta case es la capacidad de generar prototipos bien desde un fichero xml que se ha guardado previamente como hemos dicho antes o bien cogiendo los datos desde el diseño actual que contenga la aplicación. Esta generación de prototipos consiste únicamente en invocar a un motor de prototipos que ya

estaba previamente implementado, el cual se encarga de generar físicamente el diseño que se ha estado haciendo con la herramienta case anteriormente.

2. Descripción de PlumbingMatic

Esta herramienta CASE va a permitir al usuario de forma sencilla, realizar interfaces y contenidos de aplicaciones hipermedias. Además dicha aplicación va a permitir como es lógico relacionar el interfaz diseñado con los contenidos de una manera muy sencilla como se explicará a lo largo de este manual. Dichos interfaces, contenidos, así como las relaciones existentes entre elementos de ambos módulos podrán ser salvados y posteriormente recuperados desde ficheros xml. Además está aplicación permitirá generar y crear prototipos mediante un motor específico para dicha generación.

La intención de este manual es ofrecer una guía rápida de manejo de la aplicación, explicando que relaciones pueden existir entre los elementos tanto de interfaz como de contenidos, al igual que se intentará dar una visión de que elementos de dichos módulos pueden estar relacionados.

INSTALACIÓN Y EJECUCIÓN

Antes de nada nos vamos a centrar en que es necesario para ejecutar el programa y como debemos llevar a cabo dicha ejecución. Esta herramienta CASE para el desarrollo de aplicaciones hipermedia está desarrollada en Java por tanto es necesario tener instalada un máquina virtual de Java para poder ejecutar la aplicación. La aplicación es entregada en un jar con lo que es independiente de la plataforma, solo se necesita como se ha dicho antes una máquina virtual.

Los pasos necesarios para ejecutar el programa son los siguientes:

- Copiamos el fichero case.jar en el directorio que nosotros queramos, por ejemplo en **c:\herramientaCase**
- Una vez copiado el archivo tenemos dos formas de ejecutarlo:
    - Haciendo doble click sobre el fichero case.jar.
    - Abriendo una ventana de ms-dos.

      Inicio -> Ejecutar ->cmd -> Aceptar

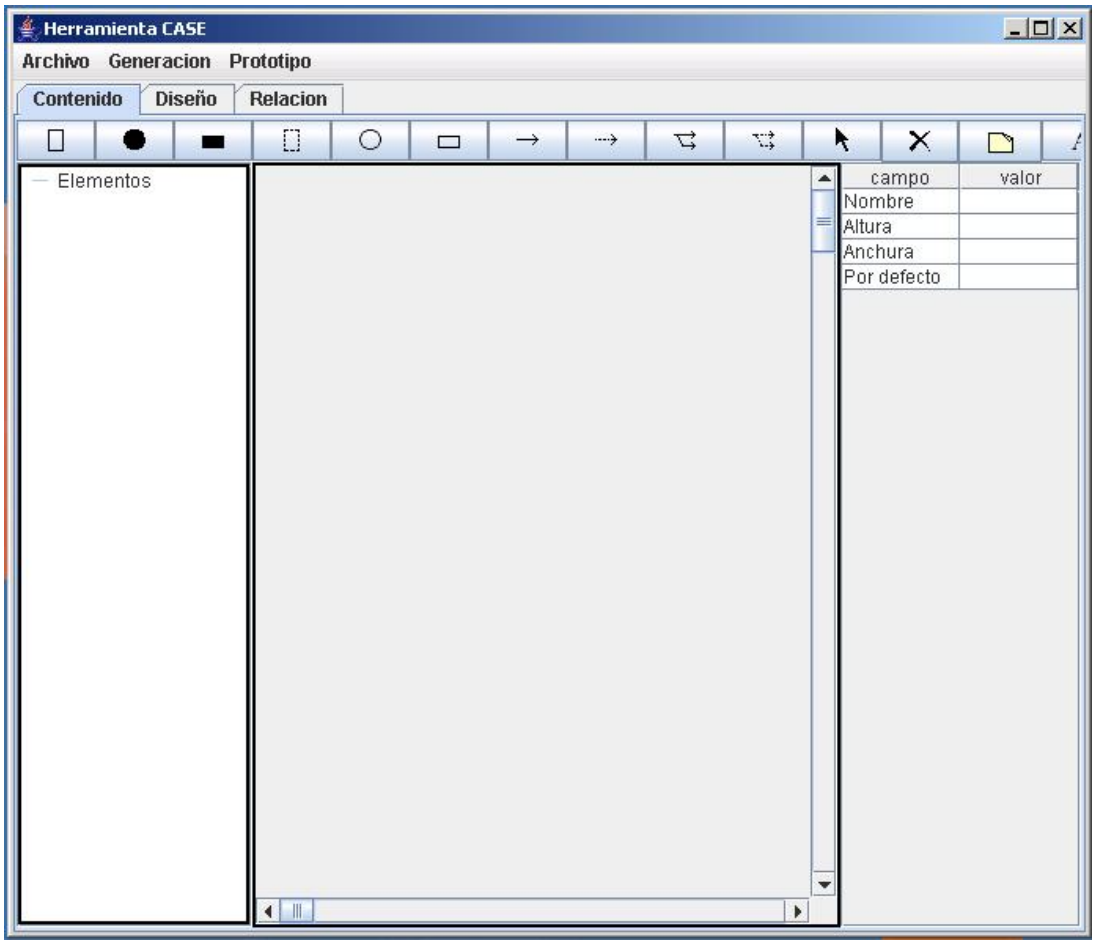
Nos situamos en el fichero donde hayamos copiado el fichero case.jar, en nuestro caso c:\herramientaCase\, para ello escribimos en la ventana de ms-dos que hemos abierto anteriormente:

cd c:\herramientaCase

Una vez en ese directorio lo que haces es ejecutar el siguiente comando:

java -jar case.jar

Y ya tendremos la herramienta case funcionando.



- Figura B.1 Pantalla inicial tras el arranque de la aplicación -

NOTA: Para poder ejecutar el comando java desde cualquier punto en la ventana de ms-dos abierta, tenemos que añadir, en caso de que no tener declarado en nuestro sistema, la variable PATH para lo cual tendremos que poner la ruta completa hasta el fichero bin de la jdk.

GUíA RÁPIDA DE USO

Ahora nos vamos a centrar en ver como funciona nuestra herramienta CASE. Primero vamos a ver como se llevaría a cabo un interfaz, el segundo paso es ver como se desarrollaría el contenido de ese interfaz y el tercero paso es relacionar el interfaz con el contenido.

Una de las características gráficas de esta herramienta es la posibilidad de trabajar en modo interfaz, en modo contenido o con ambas vistas, mediante un sistema muy sencillo de pestañas que permiten al usuario pasar rápidamente de una vista a otra, sin mayor complicación.

DESARROLLO DE UN INTERFAZ

En la pantalla de interfaz, al usuario se le presenta una barra de herramientas con todas las posibles funcionalidades a realizar, como se muestra en la siguiente figura.

- Figura B.2 Barra de herramientas de la pestaña interfaz -

SELECCIÓN

Pulsamos el botón correspondiente a la selección  posteriormente marcamos la figura que queramos con tan solo llevar el cursor con el ratón a algún punto que este contenido por el elemento y presionar el botón izquierdo del ratón. Observaremos que los bordes de la figura cambiarán a color rojo, mientras que las demás figuras mantienen su color correspondiente.

SELECCIÓN MÚLTIPLE

Actuamos de forma similar a la anterior sólo que en este caso tenemos la tecla *Ctrl* presionada mientras vamos seleccionando todos los elementos que queremos selección.

DIBUJANDO NODO NEXO

Para poder dibujar un nodo nexo nos basta con pulsar el botón correspondiente a dicha acción  posteriormente hacemos click sobre cualquier punto de la pizarra de interfaz.

DIBUJANDO NODO CONTENEDOR

Pulsamos el botón correspondiente a dicha acción  posteriormente hacemos click sobre cualquier punto de la pizarra de interfaz.

DIBUJANDO NODO ACTIVADOR DE NEXO

Pulsamos el botón correspondiente a dicha acción  posteriormente hacemos click sobre cualquier punto de la pizarra de interfaz.

DIBUJANDO ENLACE

Pulsamos el botón correspondiente a dicha acción  posteriormente tenemos que hacer click sobre el nodo que será el origen de la relación. Una vez elegido el origen de forma análoga tenemos que hacer click en el nodo destino. Pero con hacer esto no nos vale, existe una tabla de restricciones con respecto a que relaciones puede existir entre los distintos elementos de interfaz (Véase tabla 1).

DIBUJANDO ENLACE SINCRO

Pulsamos el botón correspondiente a dicha acción  posteriormente tenemos que hacer click sobre el nodo que será el origen de la relación. Una vez elegido el origen de forma análoga tenemos que hacer click en el nodo destino. Pero con hacer esto no nos vale, existe una tabla de restricciones con respecto a que relaciones puede existir entre los distintos elementos de interfaz (Véase tabla 1). La diferencia de este enlace con el anterior es que se nos pedirá una unidad de tiempo.

DIBUJANDO CONEXIÓN

Pulsamos el botón correspondiente a dicha acción  posteriormente tenemos que hacer click sobre el nodo que será el origen de la relación. Una vez elegido el origen de forma análoga tenemos que hacer click en el nodo destino. Pero con hacer esto no nos vale, existe una tabla de restricciones con respecto a que relaciones puede existir entre los distintos elementos de interfaz (Véase tabla 1).

DIBUJANDO CONEXIÓN SINCRO

Pulsamos el botón correspondiente a dicha acción  posteriormente tenemos que hacer click sobre el nodo que será el origen de la relación. Una vez elegido el origen de forma análoga tenemos que hacer click en el nodo destino. Pero con hacer esto no nos vale, existe una tabla de restricciones con respecto a que relaciones puede existir entre los distintos elementos de interfaz (Véase tabla 1). La diferencia de este enlace con el anterior es que se nos pedirá una unidad de tiempo.

Origen\Destino	Nexo	Contenido	Activador Nexo
Nexo	Enlace sincro	Conexión, Conexión sincro	Conexión
Contenido	-	Enlace	-
Activador Nexo	Enlace	-	-

- Figura B.3 Tabla de restricciones de relación entre elementos interfaz -

DIBUJANDO AREA DE TEXTO

Si queremos poner en el interfaz una aclaración o una observación podemos dibujar una especie de posit.

Para ello solamente tenemos que pulsar el botón  Una vez pulsado el botón lo que tenemos que hacer es hacer click sobre cualquier punto de la pizarra interfaz.

DIBUJANDO LÍNEA DE TEXTO

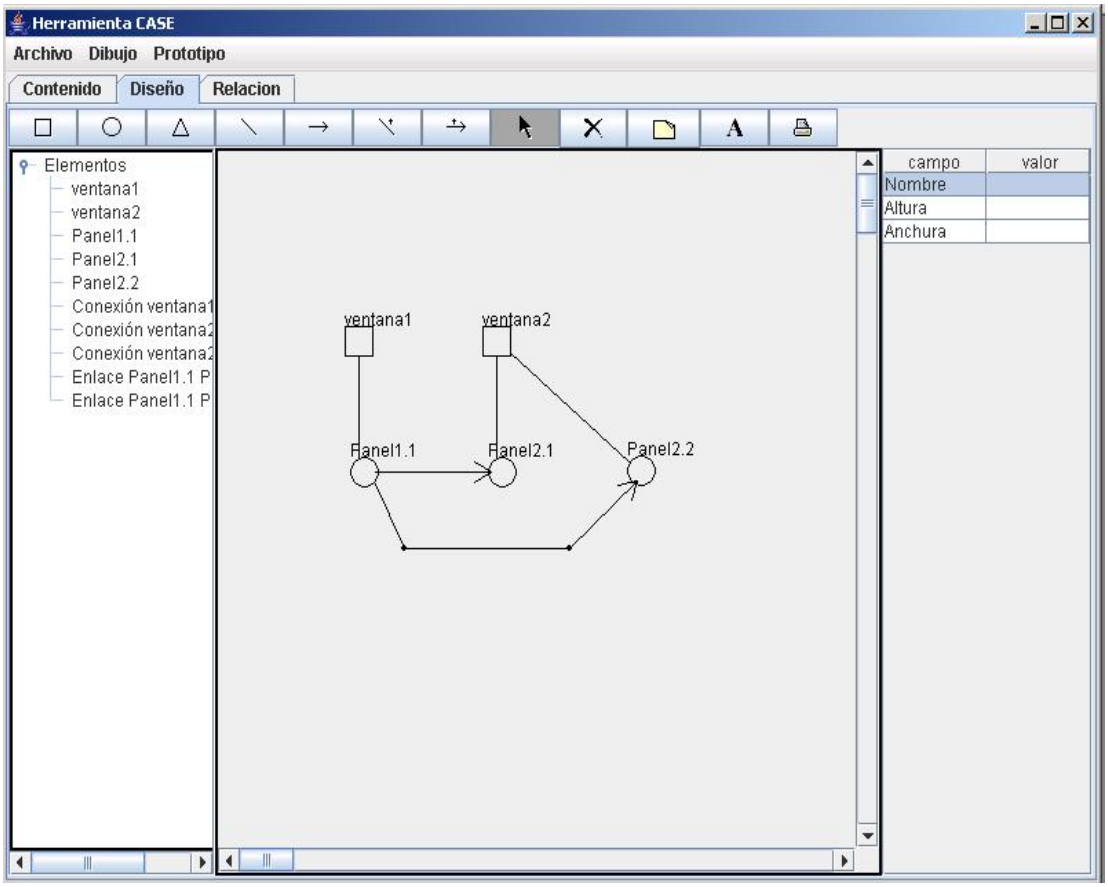
Si queremos poner en el interfaz una etiquete podemos dibujar una línea de texto sin borde ni nada.

Para ello solamente tenemos que pulsar el botón  Una vez pulsado el botón lo que tenemos que hacer es hacer click sobre cualquier punto de la pizarra interfaz.

IMPRESIÓN DE UN INTERFAZ

Para ello simplemente tenemos que pulsar el botón el siguiente botón 

EJEMPLO DE DESARROLLO DE UN INTERFAZ



-Figura B.4 Ejemplo completo de interfaz -

DESARROLLO DE UN CONTENIDO

En este apartado vamos a intentar explicar como se llevaría a cabo el desarrollo de un contenido que posteriormente relacionaremos con el diseño desarrollado anteriormente. En la pantalla de contenido, al usuario se le presenta una barra de herramientas con todas las posibles funcionalidades a realizar, como se muestra en la siguiente figura.

- Figura B.5 Barra de herramientas de la pestaña contenido -

SELECCIÓN

Pulsamos el botón correspondiente a la selección  posteriormente marcamos la figura que queramos con tan solo llevar el cursor con el ratón a algún punto que este contenido por el elemento y presionar el botón izquierdo del ratón. Observaremos que los bordes de la figura cambiarán a color rojo, mientras que las demás figuras mantienen su color correspondiente. Decir también que se pueden seleccionar tramos de un enlace n-ario.

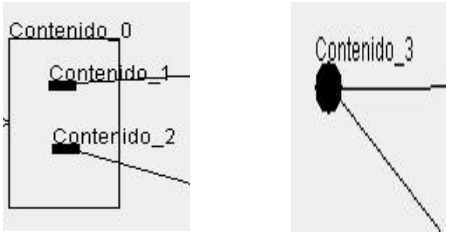
SELECCIÓN MÚLTIPLE

Actuamos de forma similar a la anterior sólo que en este caso tenemos la tecla *Ctrl* presionada mientras vamos seleccionando todos los elementos que queremos selección.

DIBUJANDO CONTENIDO ESTÁTICO

Pulsamos uno de los dos botones  ó  posteriormente lo que debemos hacer es click en cualquier punto de la pizarra contenido que queramos.

Decir que un contenido estático tiene dos representaciones una es un rectángulo en posición vertical y la otra es un circulo coloreado de negro. Esta última representación la podríamos ver como una representación comprimida de la anterior, y ¿por qué decimos esto?, a esta pregunta responderemos con un ejemplo.



- Figura B.6 Ejemplo de formas alternativas de contenido Estático -

Si nos damos cuenta en este ejemplo tenemos dos notaciones alternativas para el mismo significado, es decir la figura de la izquierda consiste en un rectángulo en vertical que posee anclas tanto estáticas como dinámicas y la figura de la derecha igualmente vemos que tiene dos enlaces que salen de él, pero que no se ven porque es como si hubiéramos hecho una compresión de la figura de la izquierda.

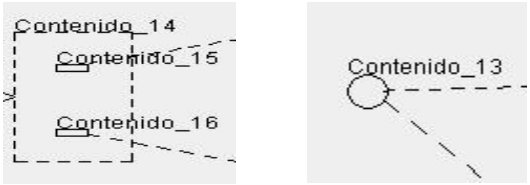
DIBUJANDO ANCLA ESTÁTICO

Para ello pulsamos el botón  , y posteriormente hacemos click en algún punto que se encuentre dentro de un contenido estático (representación rectangular) o un contenido dinámico (representación rectangular). Un ancla no puede aparecer aislada, es decir tiene que formar parte o bien de un contenido estático o bien de un contenido dinámico.

DIBUJANDO CONTENIDO DINÁMICO

Tenemos dos alternativa para pintarlo  ó  posteriormente lo que debemos hacer es click en cualquier punto de la pizarra contenido que queramos.

Decir que un contenido dinámico tiene dos representaciones una es un rectángulo en posición vertical con línea discontinua y la otra es un círculo sin rellenar. Esta última representación la podríamos ver como una representación comprimida de la anterior, y ¿por qué decimos esto?, a esta pregunta responderemos con un ejemplo.



- Figura B.7 Ejemplo de formas alternativas de contenido Dinámico -

Si nos damos cuenta en este ejemplo tenemos dos notaciones alternativas para el mismo significado, es decir la figura de la izquierda consiste en un rectángulo en vertical de línea discontinua que posee anclas tanto estáticas como dinámicas y la figura de la derecha igualmente vemos que tiene dos enlaces que salen de él, pero que no se ven porque es como si hubiéramos hecho una compresión de la figura de la izquierda.

DIBUJANDO ANCLA DINÁMICA

Para dibujar un ancla dinámica pulsamos el botón  posteriormente hacemos click en algún punto que se encuentre dentro de un contenido estático (representación rectangular) o un contenido dinámico (representación rectangular). Un ancla no puede aparecer aislada, es decir tiene que formar parte o bien de un contenido estático o bien de un contenido dinámico.

DIBUJANDO UN ENLACE ESTÁTICO

Para dibujar un ancla estática pulsamos el botón  posteriormente tenemos que hacer click en el nodo origen de dicha relación que debe ser un ancla estática, un contenido estático (representación circular), o bien un contenido dinámico (representación circular). Una vez hecho click en la figura origen debemos hacer click en la figura que queramos que sea destino, la cual podrá ser un contenido estático (ambas representaciones). (Véase tabla 2)

DIBUJANDO UN ENLACE DINÁMICO

Para ello lo que tenemos que hacer es pulsar el  posteriormente tenemos que hacer click en el nodo origen de dicha relación que debe ser un ancla dinámica, un contenido estático (representación circular), o bien un contenido dinámico (representación circular). Una vez hecho click en la figura origen debemos hacer click en la figura que queramos que sea destino, la cual podrá ser un contenido dinámico (ambas representaciones).

DIBUJANDO UN ENLACE ESTÁTICO N-ARIO

Esto es análogo al enlace estático, solo que en este caso tenemos que ir pulsando los N nodos destino de la relación. Una vez marcados todos los nodos destino lo que tenemos que hacer es pinchar en cualquier punto de la pizarra contenido que no tenga ningún nodo y se nos mostrará el enlace. El botón para realizar esta op 

DIBUJANDO UN ENLACE DINÁMICO N-ARIO

Esto es análogo al enlace dinámico, solo que en este caso tenemos que ir pulsando los N nodos destino de la relación. Una vez marcados todos los nodos destino lo que tenemos que hacer es pinchar en cualquier punto de la pizarra contenido que no tenga ningún nodo y se nos mostrará el enlace. El botón para realizar esta operación es 

DIBUJANDO AREA DE TEXTO

Si queremos poner en el contenido una aclaración o una observación podemos dibujar una especie de posit. Para ello solamente tenemos que pulsa el botón 

Una vez pulsado el botón lo que tenemos que hacer es hacer click sobre cualquier punto de la pizarra contenido.

DIBUJANDO LÍNEA DE TEXTO

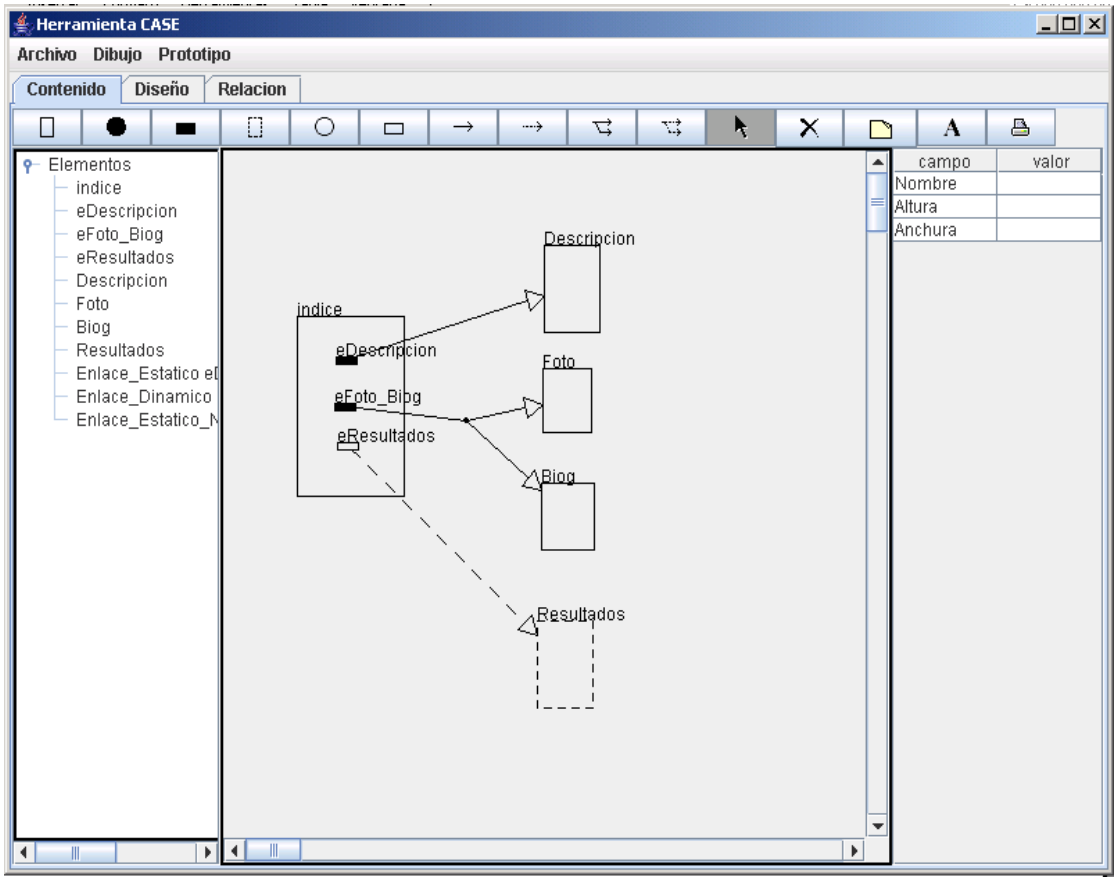
Si queremos poner en el contenido una etiquete podemos dibujar una línea de texto sin borde ni nada. Para ello solamente tenemos que pulsar el botón 

Una vez pulsado el botón lo que tenemos que hacer es hacer click sobre cualquier punto de la pizarra contenido.

IMPRESIÓN DE UN CONTENIDO

Para ello simplemente tenemos que pulsar el botón el siguiente botón 

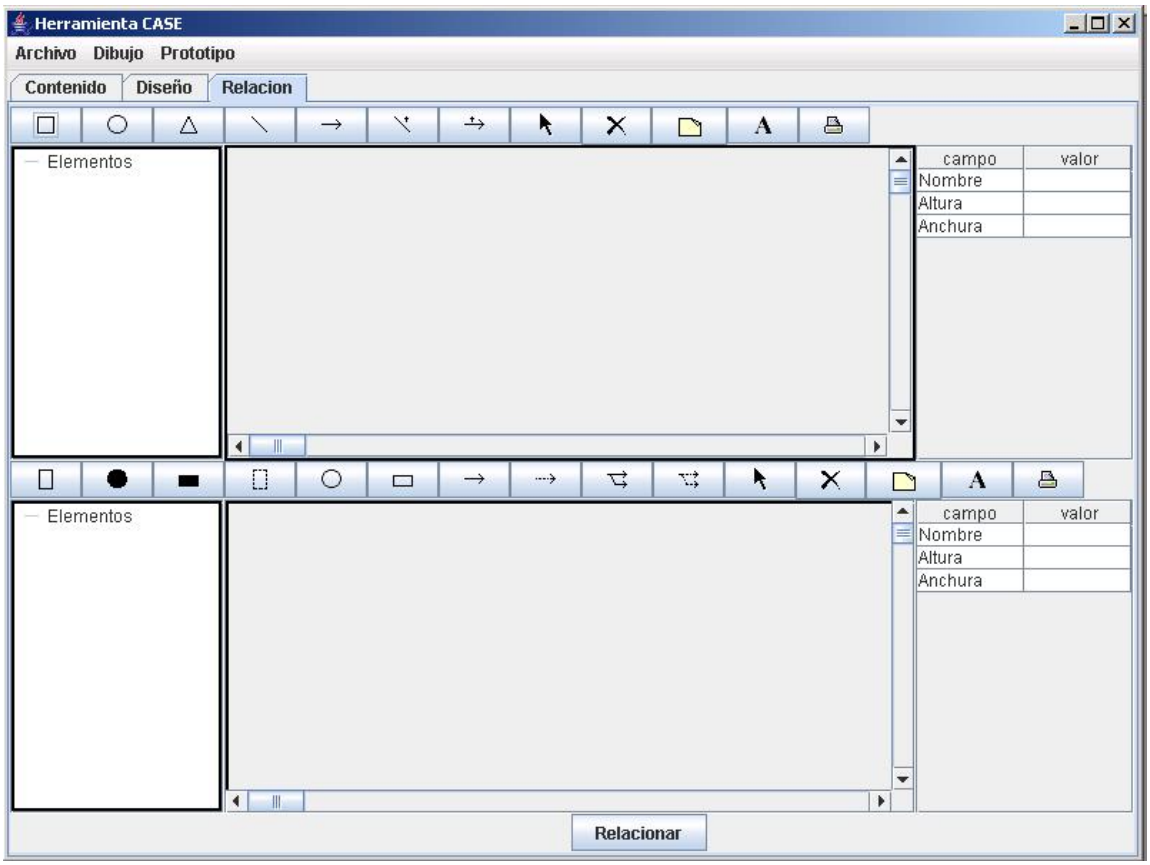
EJEMPLO DE DESARROLLO DE UN CONTENIDO



- Figura B.9 Ejemplo completo del desarrollo de contenido -

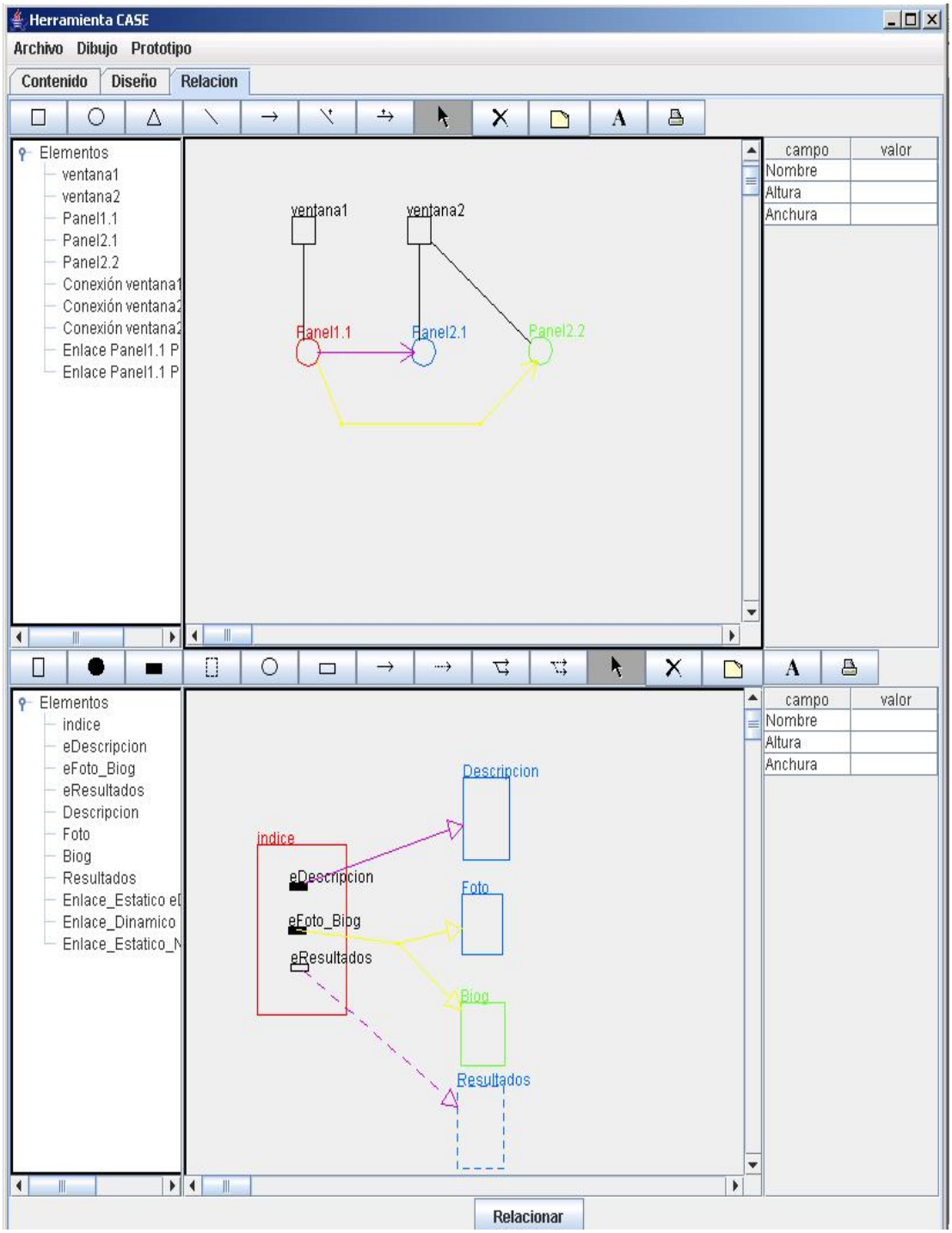
Pues ya hemos llegado a la parte final del desarrollo de una aplicación hipermedia. Tenemos por un lado el interfaz de la misma y por otro tenemos los contenidos de dicho interfaz, nos falta por tanto relacionarlos. Para ello nos vamos a la pestaña *Relación* del interfaz gráfico.

RELACIONANDO INTERFAZ CON CONTENIDO



- Figura B.10 Pantalla de relación -

Como podemos observar en esta imagen, en la pantalla de relación se nos muestra las dos visiones juntas, es decir el diseño y el contenido. Ahora simplemente vamos seleccionando los distintos elementos del interfaz y del contenido que están relacionados. Una vez seleccionados todos los que queremos lo que hacemos el pulsar el botón *Relacionar*, el cual nos despliega un diálogo donde podemos elegir el color que queramos para esa relación. Posteriormente a hacer esto, los elementos relacionados se mostrarán de dicho color.



- Figura B.11 Ejemplo completo -

SALVAR Y GUARDAR

En este apartado nos vamos a centrar en la persistencia de tanto las aplicaciones hipermedia diseñadas como los prototipos que se puedan generar.

SALVAR DISEÑO

Para poder guardar o salvar el diseño de una aplicación hipermedia diseñada, como puede ser el ejemplo que hemos estado desarrollando paralelamente a la explicación del manejo de dicha herramienta, lo único que tenemos que hacer es pulsar el ítem correspondiente del menú persistencia **"*SALVAR DISEÑO*"**, . En este momento se nos abrirá un diálogo donde se nos permitirá navegar por el árbol de ficheros de nuestro ordenador. Simplemente tendremos que dar un nombre al fichero donde se guardará el diseño.

CARGAR DISEÑO

Para poder cargar el diseño de una aplicación hipermedia desarrollado anteriormente, se trabaja de forma similar al anterior caso, es decir se pulsa el ítem correspondiente del menú persistencia **"*GUARDAR DISEÑO*"**, y se navega por la estructura de ficheros de nuestro ordenador hasta encontrar el fichero que anteriormente guardamos.

SALVAR PROTOTIPO

Análogo al salvar diseño, pero en este caso pinchamos en el ítem correspondiente del menú, **"*SALVAR PROTOTIPO*".**

CARGAR PROTOTIPO

Análogo al cargar diseño, pero en este caso pinchamos en el ítem correspondiente del menú, **"*CARGAR PROTOTIPO*".**

PROTOTIPADO

GENERAR PROTOTIPO

Para poder generar un prototipo a partir de un diseño anteriormente realizado, simplemente tenemos que pulsar el ítem correspondiente en el menú prototipado, **"*GENERAR PROTOTIPO*".**

En este punto se nos irán abriendo una serie de diálogos donde deberemos ir eligiendo los archivos xml generados con anterioridad para poder generar ahora nuestro prototipo. Los archivos que hay que generar son tres, uno para interfaz, otro para contenido y otro para canalización o relación. Estos xml se van a transformar en unos archivos xml entendibles por el motor de prototipado, por ello se debe especificar el nombre de los res nuevos documentos, también mediante diálogos.

UTILIDADES DE FUNCIONAMIENTO

MOVER ELEMENTO

Para poder mover un elemento, ya sea de la parte de interfaz o de la parte de contenido, tan solo tenemos que seleccionar el elemento como se ha explicado en los puntos anteriores y posteriormente arrastrar el ratón manteniendo pulsado el botón izquierdo del mismo y soltar cuando se quiera.

CAMBIAR EL TAMAÑO DE UN ELEMENTO

Para poder mover un elemento, ya sea de la parte de interfaz o de la parte de contenido, tan solo tenemos que seleccionar el elemento como se ha visto ya y posteriormente arrastrar el ratón manteniendo pulsado el botón derecho del mismo y soltar cuando se quiera.

El tamaño de un elemento también se puede cambiar desde la tabla, donde se pueden editar las celdas de altura y anchura, así como la del nombre.

BORRAR ELEMENTOS

Si lo que queremos es borrar un elemento de alguna de las pizarras tenemos dos formas de realizar esta acción. Una de ellas es, una vez seleccionado el elemento que deseamos borrar pulsar la tecla *Supr***.** La otra opción es mediante la barra de herramientas pulsando el botón 

NOTA: Se pueden borrar y seleccionar de forma parcial enlaces pertenecientes a un enlace estático o dinámico n-ario. Solo tenemos que pinchar en la trama del enlace que queramos.

3. Casos de uso y diagramas de actividades

3.1 Introducción

En este punto de la memoria nos vamos a centrar en mostrar la distinta funcionalidad que el usuario puede llevar a cabo cuando se encuentre manejando nuestra aplicación. Se puede decir que vamos a hacer dos visiones de la funcionalidad, una simplemente centrándonos en qué es lo que ofrece nuestra aplicación (**diagramas de casos de uso**), que se puede ver como una enumeración de acciones que puede llevar a cabo nuestro usuario. El otro punto de vista es cómo se lleva a cabo esa funcionalidad, los pasos que se dan para realizar la funcionalidad, simplemente es un esquema de las fases que se deberían ir pasando, es un esquema muy de alto nivel sin meternos en ningún caso en la implementación real de la funcionalidad, eso lo veremos en otro punto más adelante.

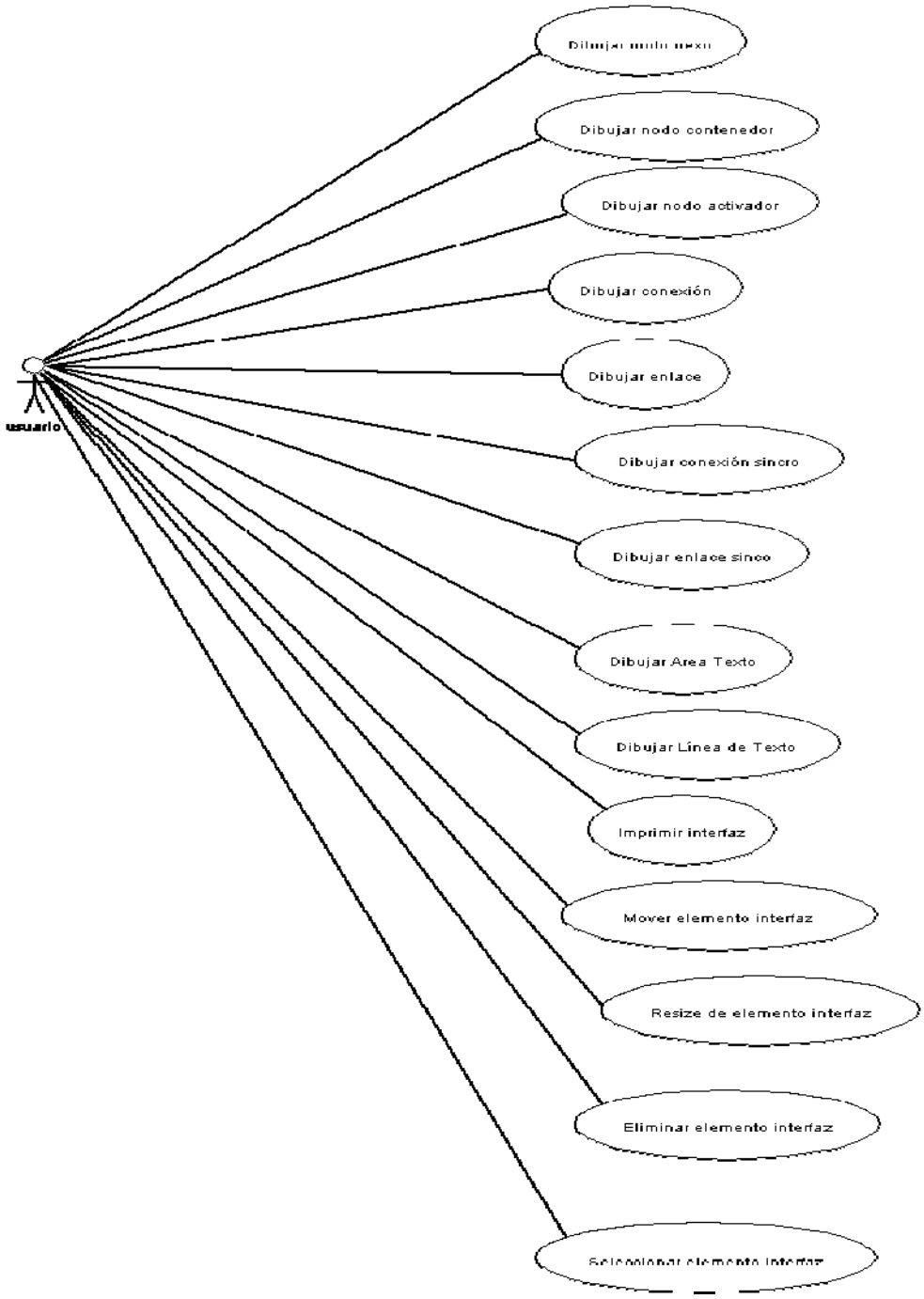
Como se mencionará más adelante, por motivos de similitud entre distintas funcionalidades y por no cargar en exceso de diagramas que básicamente van a ser iguales, lo que hemos hecho es representar los pasos que se deben llevar a cabo para realizar dicha funcionalidad, indicando después que para un cierto conjunto de funcionalidad se trabaja de forma análoga.

La estructura de este punto consiste en mostrar por cada módulo los dos tipos de diagramas, el de casos de uso y el de actividades.

3.2 Módulo gráfico

- **Nivel interfaz**
  - **Diagrama de casos de uso**

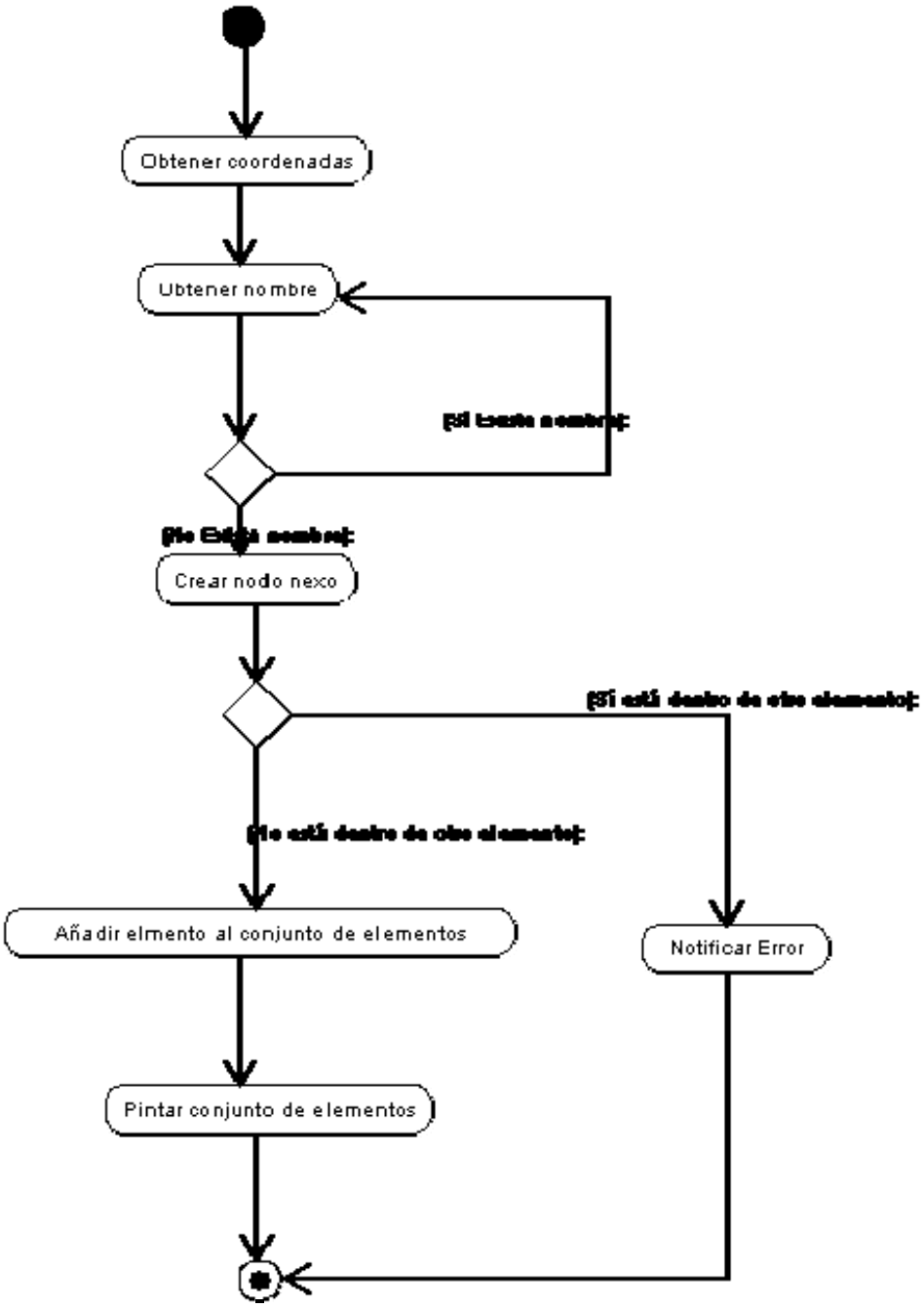
En este apartado se muestra la funcionalidad que se puede encontrar el usuario en el módulo de interfaz.



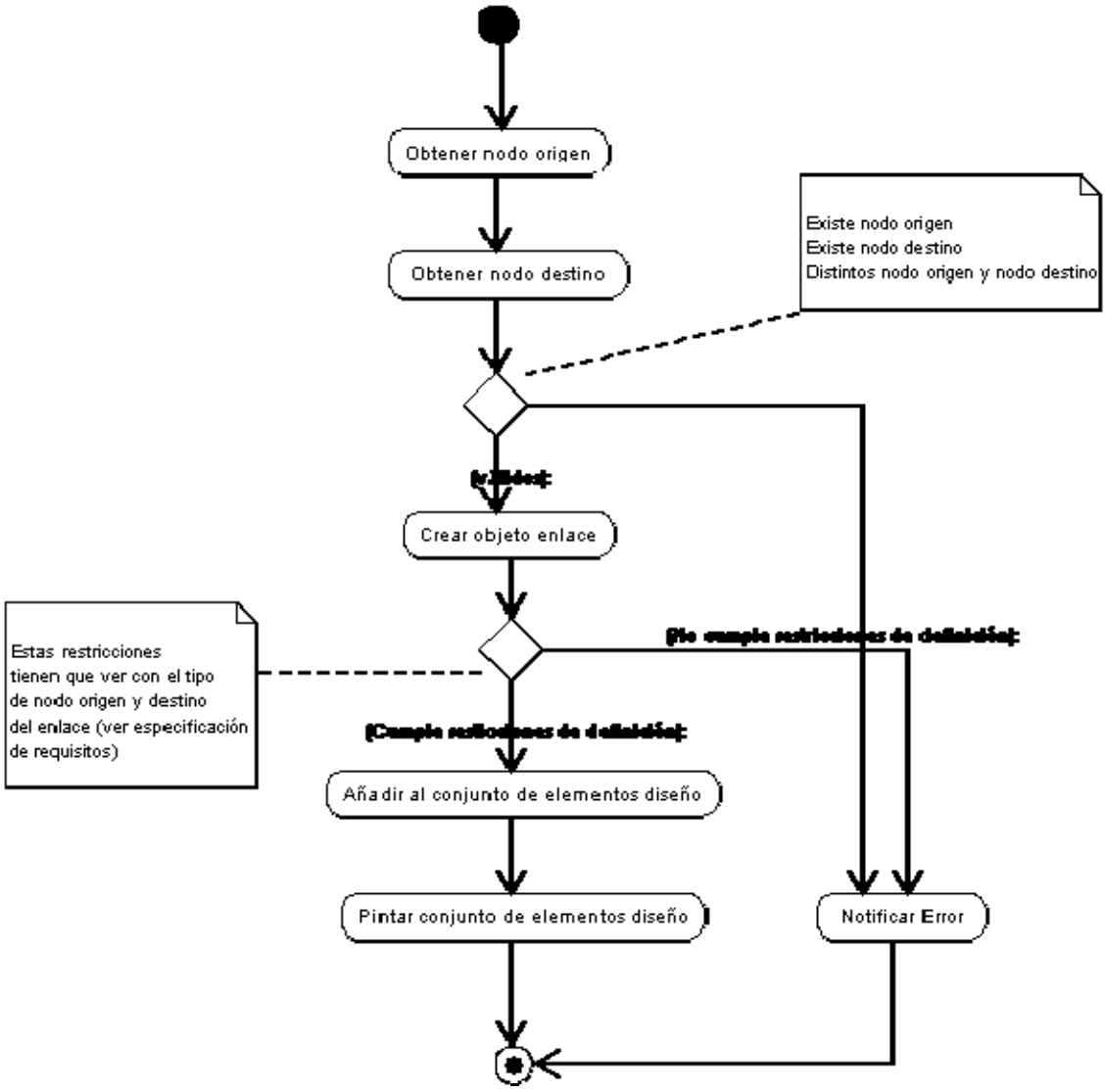
- **Figura 3.1. Diagrama de casos de uso del módulo interfaz -**

o **Diagramas de actividades**

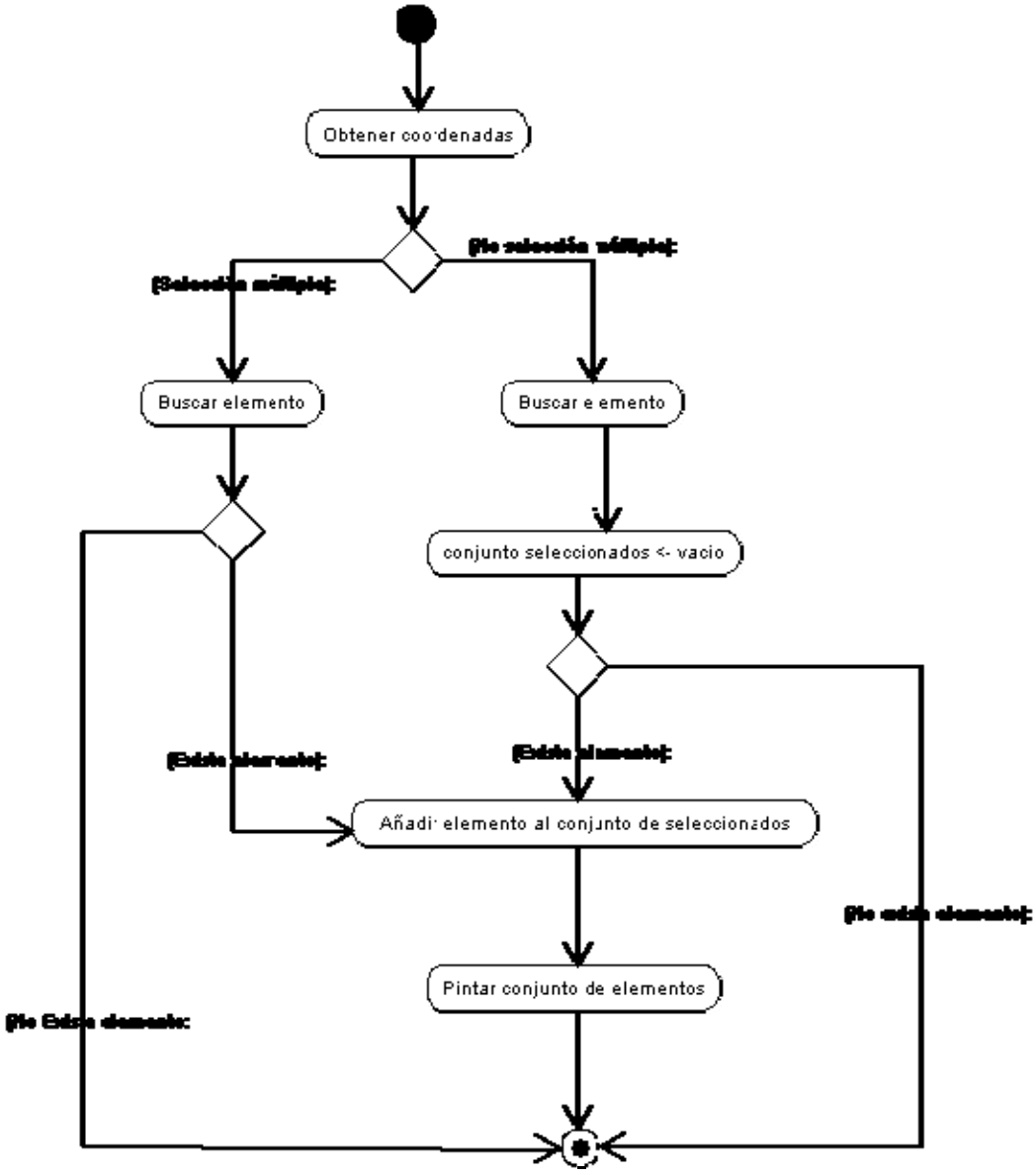
Ahora nos centramos en el cómo se llevan a cabo las distintas funcionalidades que nos presenta este módulo. Por motivos de similitud, en funcionalidad como dibujar nodo nexo, dibujar nodo contenedor y dibujar nodo activador, hemos decidido mostrar como sería dicho diagrama para uno de ellos ya que el de los siguientes sería análogo, así de forma similar se hace con dibujar enlace, dibujar conexión, dibujar enlace cincho, etc..



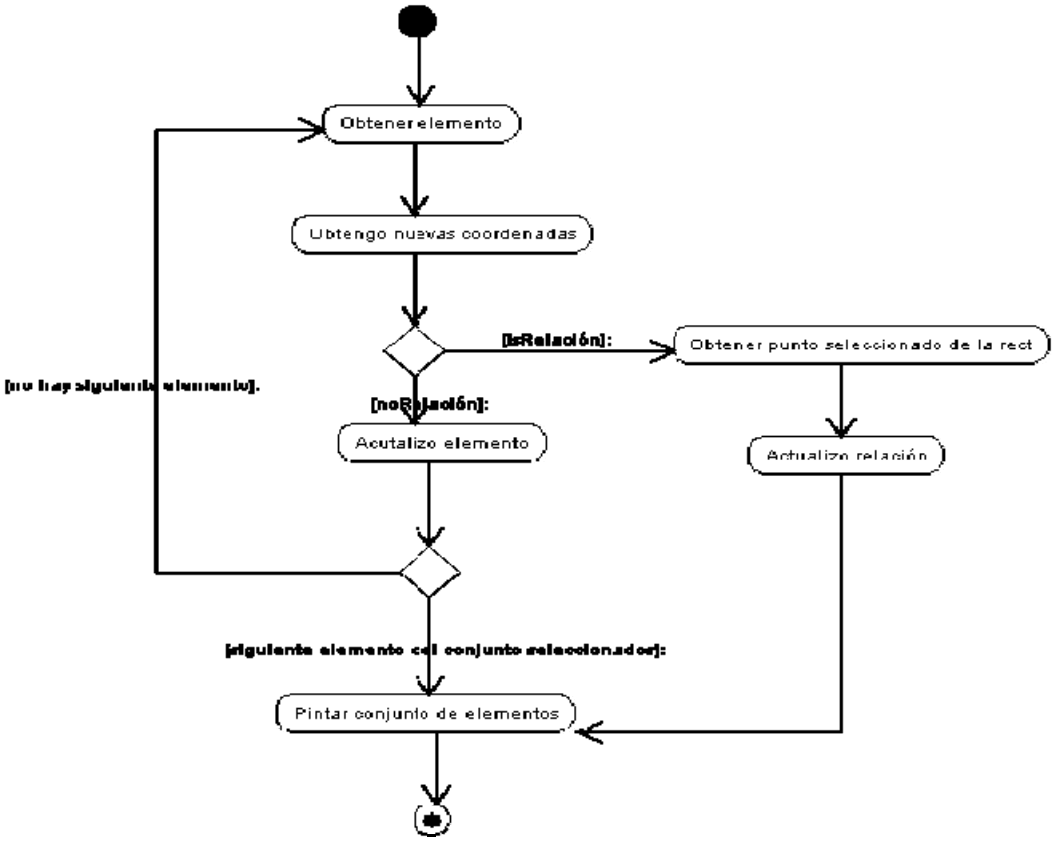
- Figura 3.2. Dibujar nodo nexo -



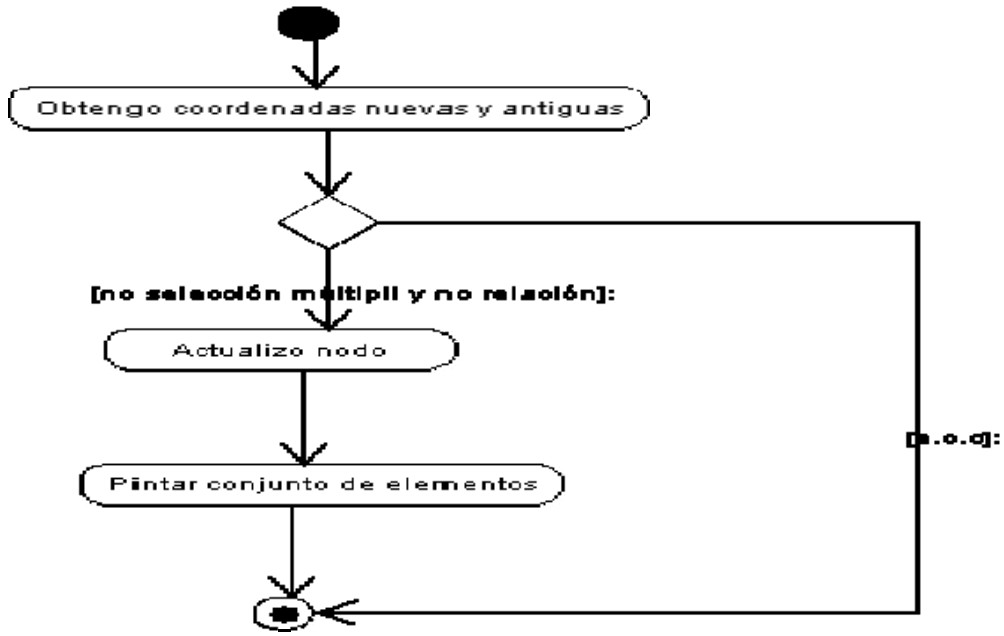
- Figura 3.3. Dibujar relación (enlace, conexión, enlace cincho, …) -



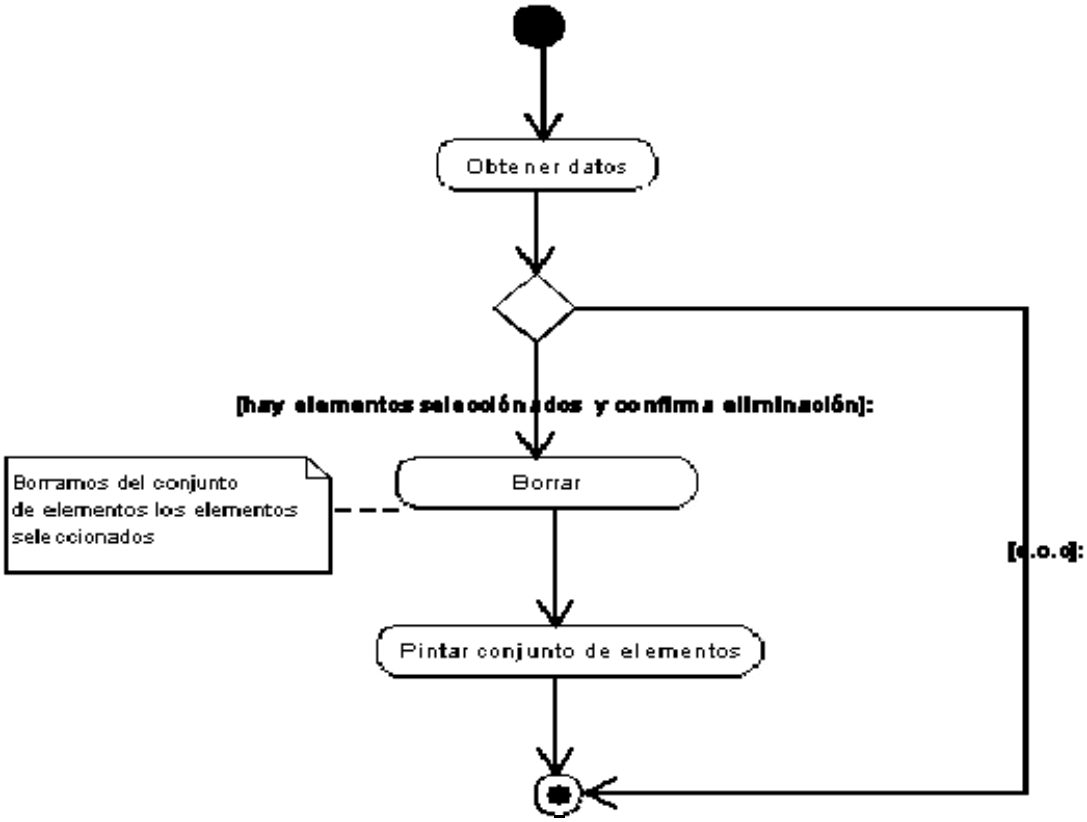
- **Figura 3.4. Seleccionar Elemento –**



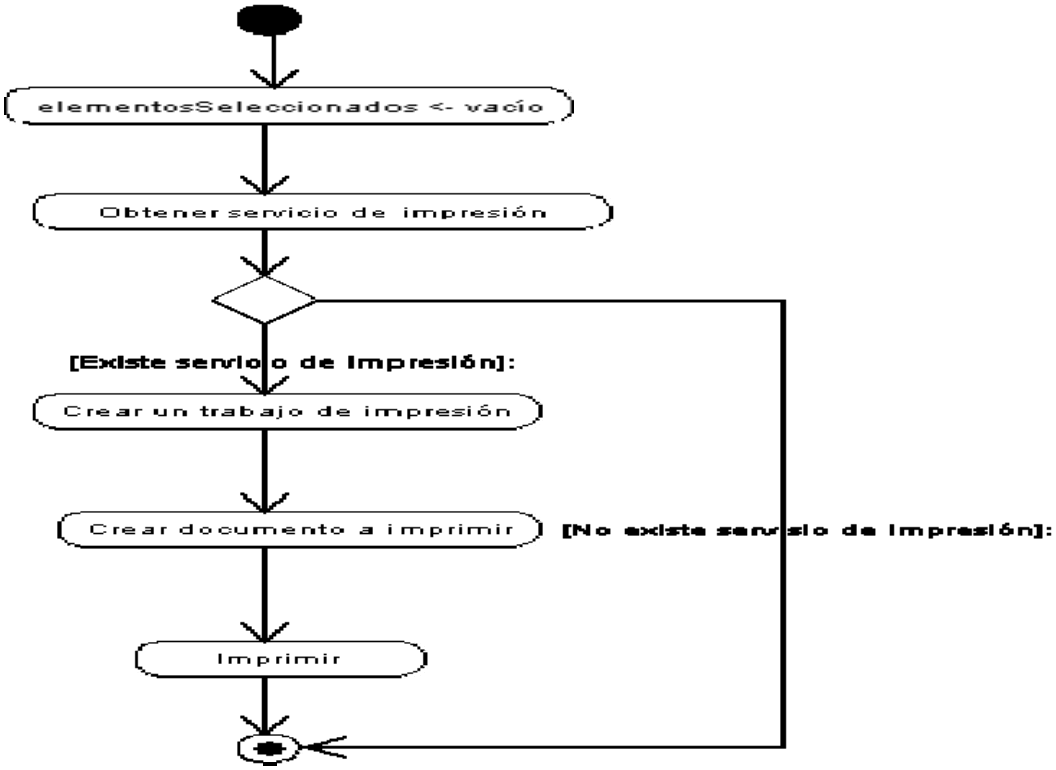
- **Figura 3-5. Mover Elemento –**



- **Figura 3.6. Resise Elemento –**



- **Figura 3.7. Eliminar elemento –**

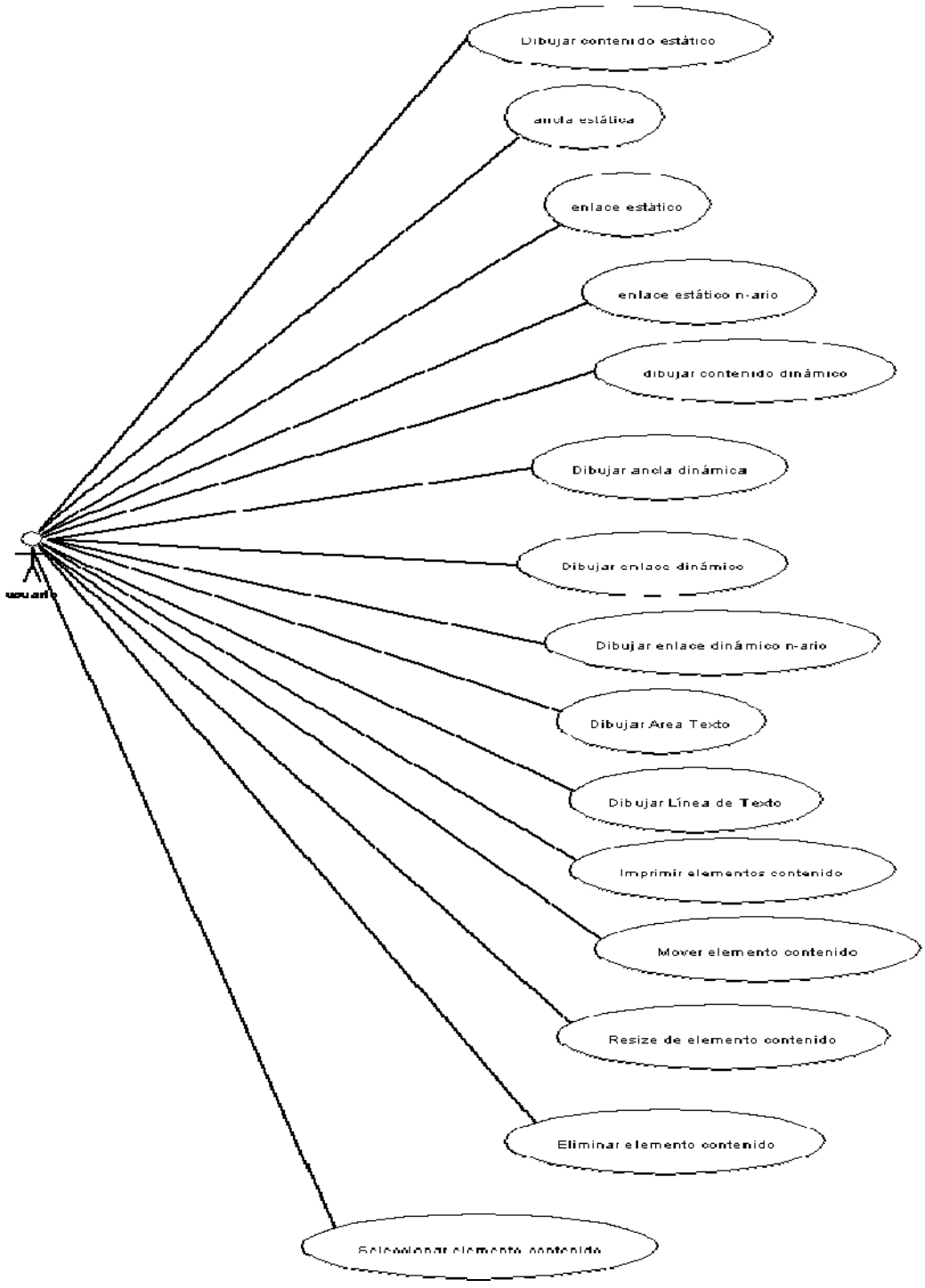


- Figura 3.8. Imprimir Interfaz -

- **Nivel contenidos**

  - **Diagrama de casos de uso**

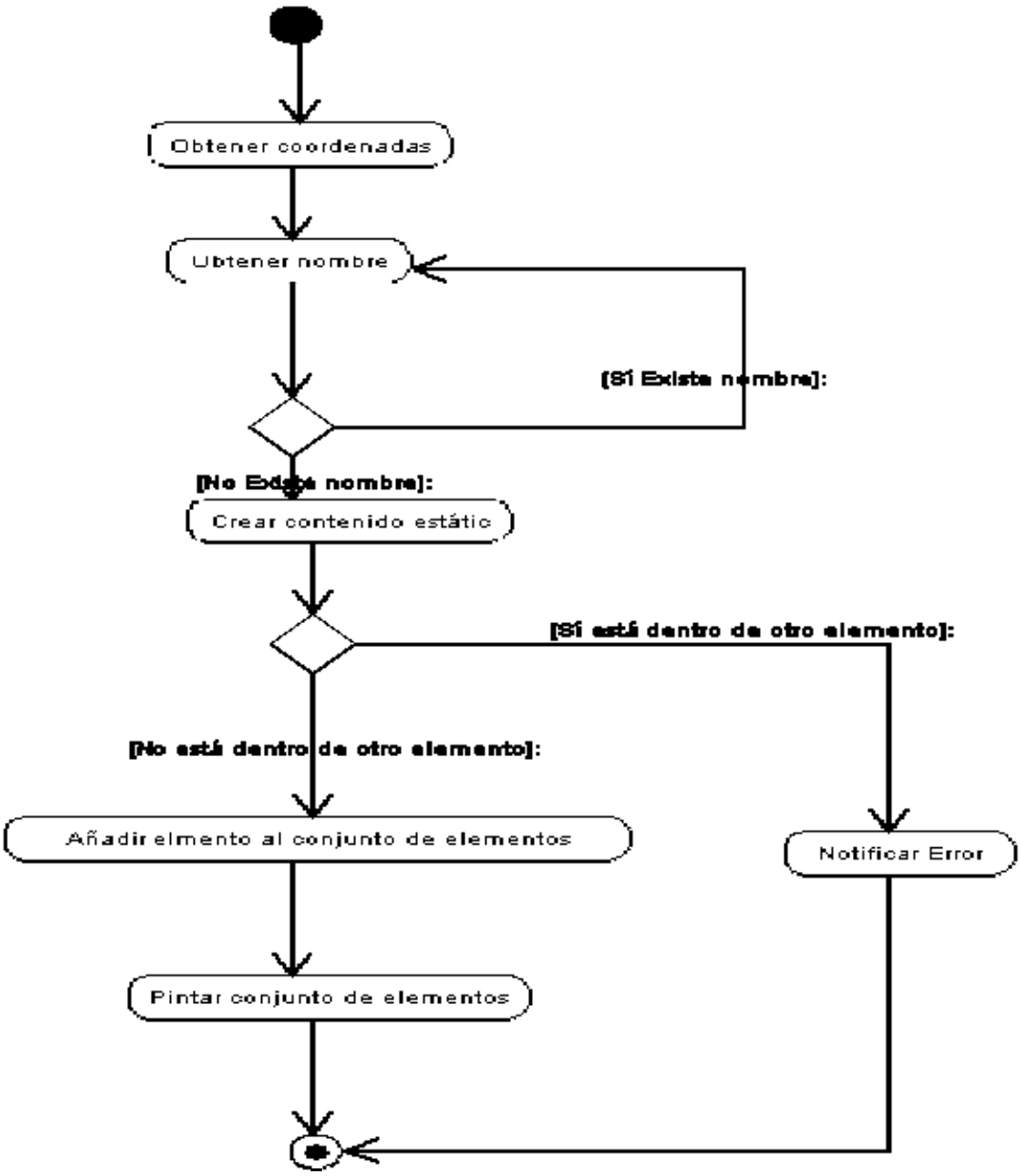
    En este apartado se muestra la funcionalidad que se puede encontrar el usuario en el módulo de contenido.



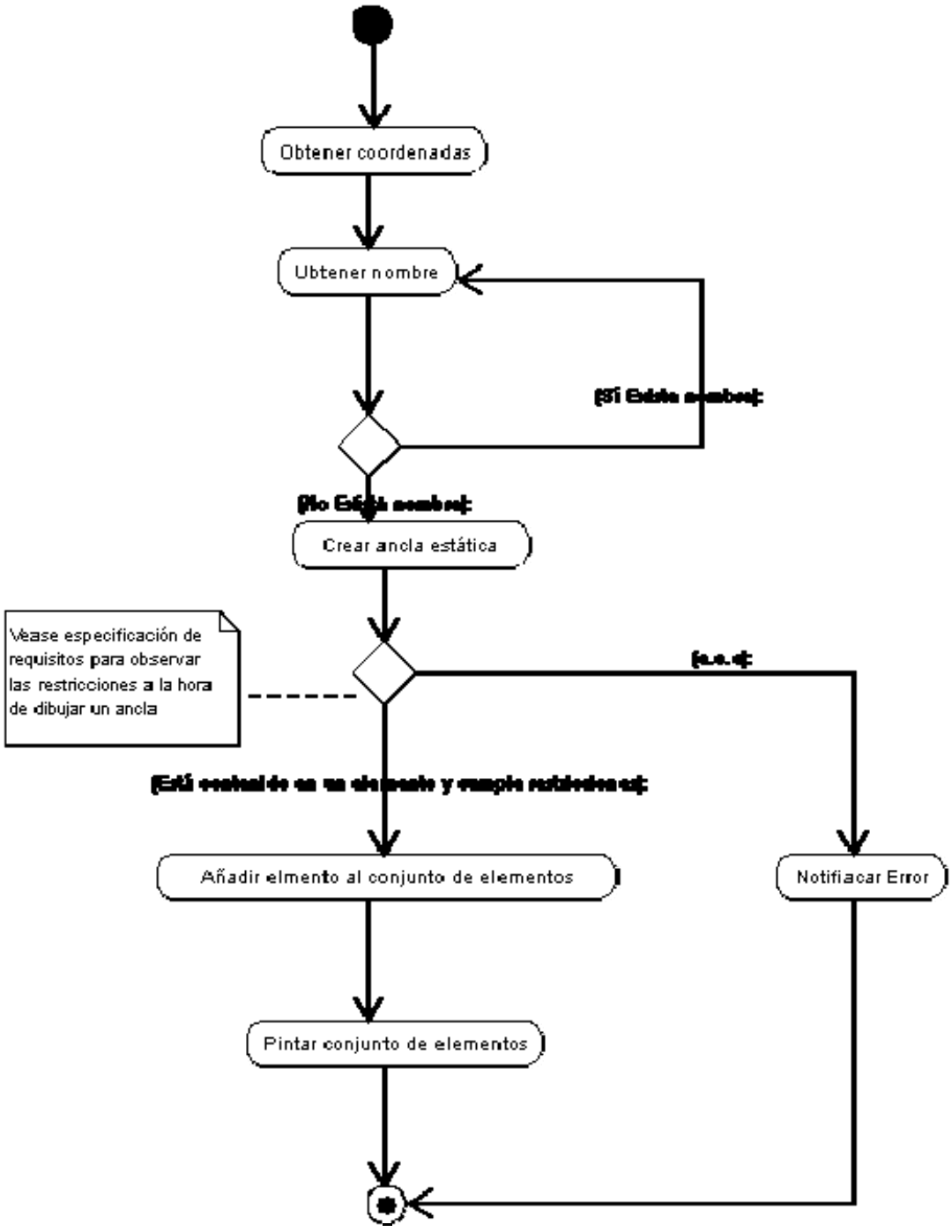
- **Figura 3.9. Diagrama de casos de uso del módulo contenido -**

- **Diagramas de actividades**

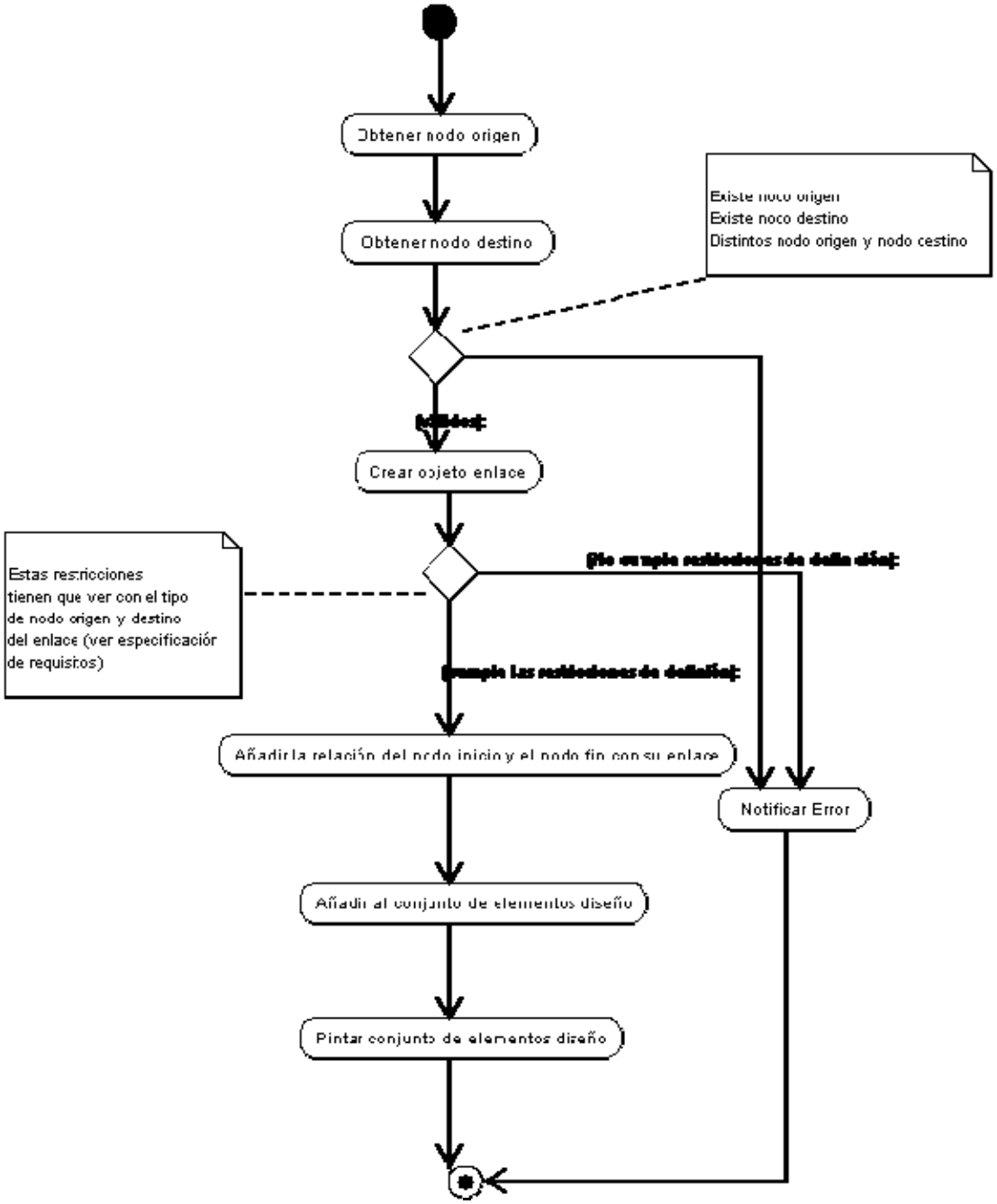
Ahora nos centramos en el cómo se llevan a cabo las distintas funcionalidades que nos presenta este módulo. Por motivos de similitud, al igual que en el módulo anterior, en la funcionalidad de dibujar contenido estático pequeño, contenido estático grande, contenido estático grande y contenido dinámico grande, hemos decidido mostrar como sería dicho diagrama para uno de ellos ya que el de los siguientes serían análogos, así de forma similar se hace con ancla estática, ancla dinámica, enlace estático, enlace estático n-ario, enlace dinámico, enlace dinámico n-ario.



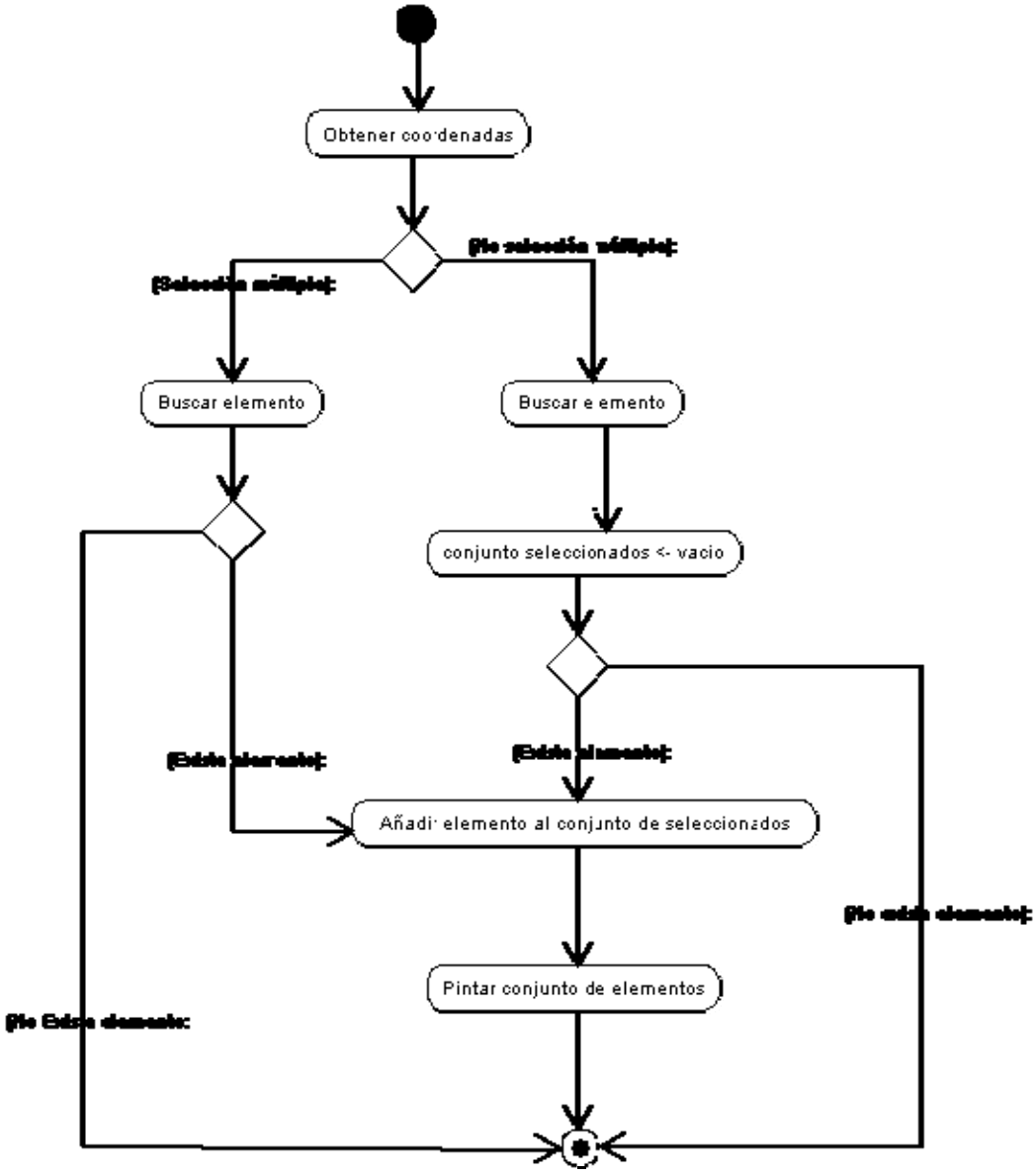
- Figura 3.10. Dibujar contenido estático pequeño -



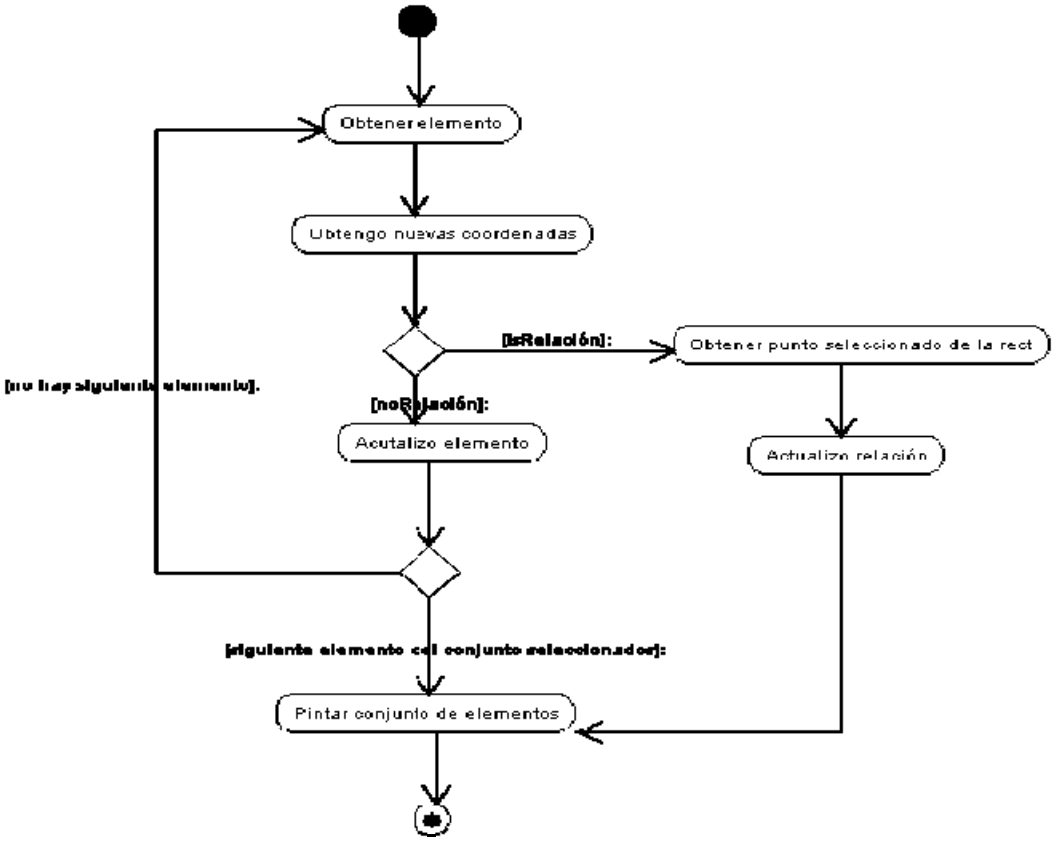
- Figura 3.11. Dibujar ancla estática -



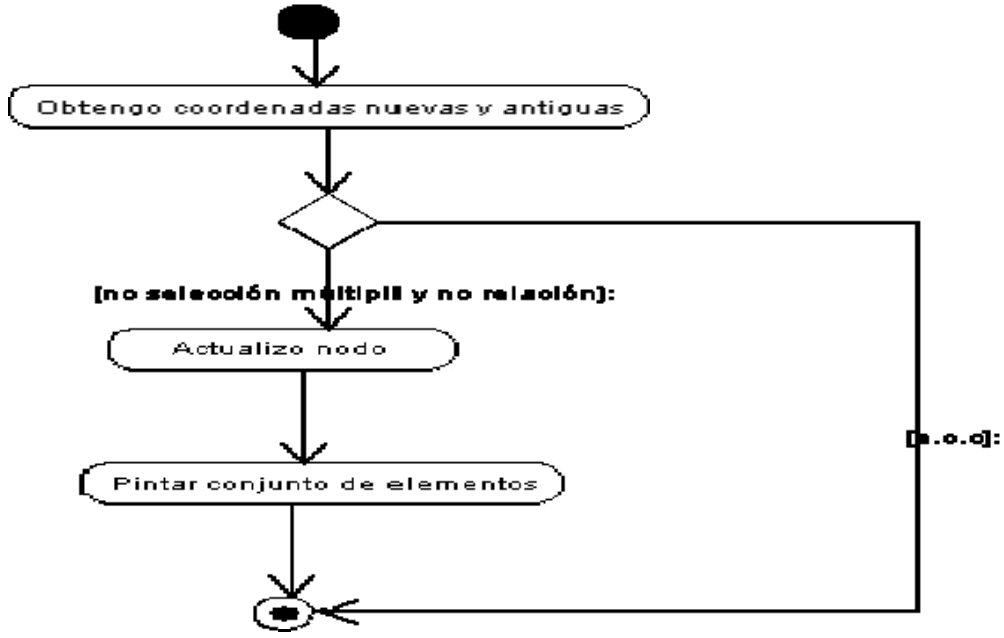
- Figura 3.12. Dibujar enlace estático -



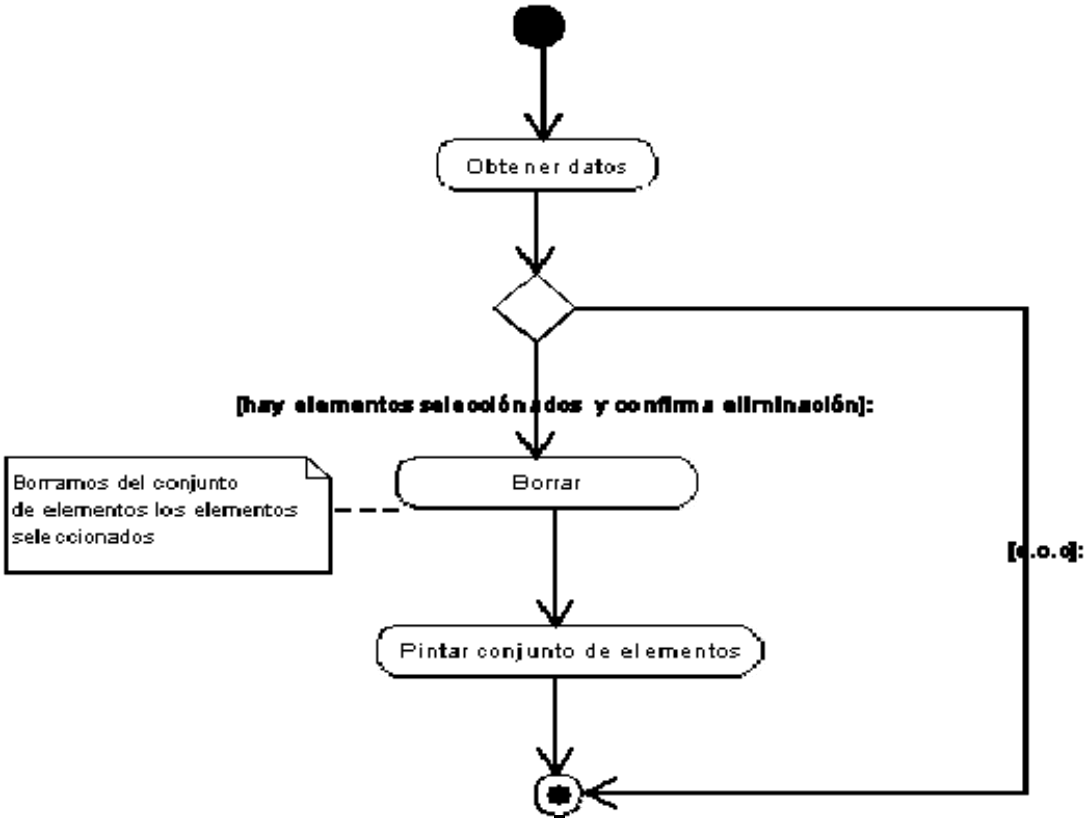
- Figura 3.13. Seleccionar elementos del contenido -



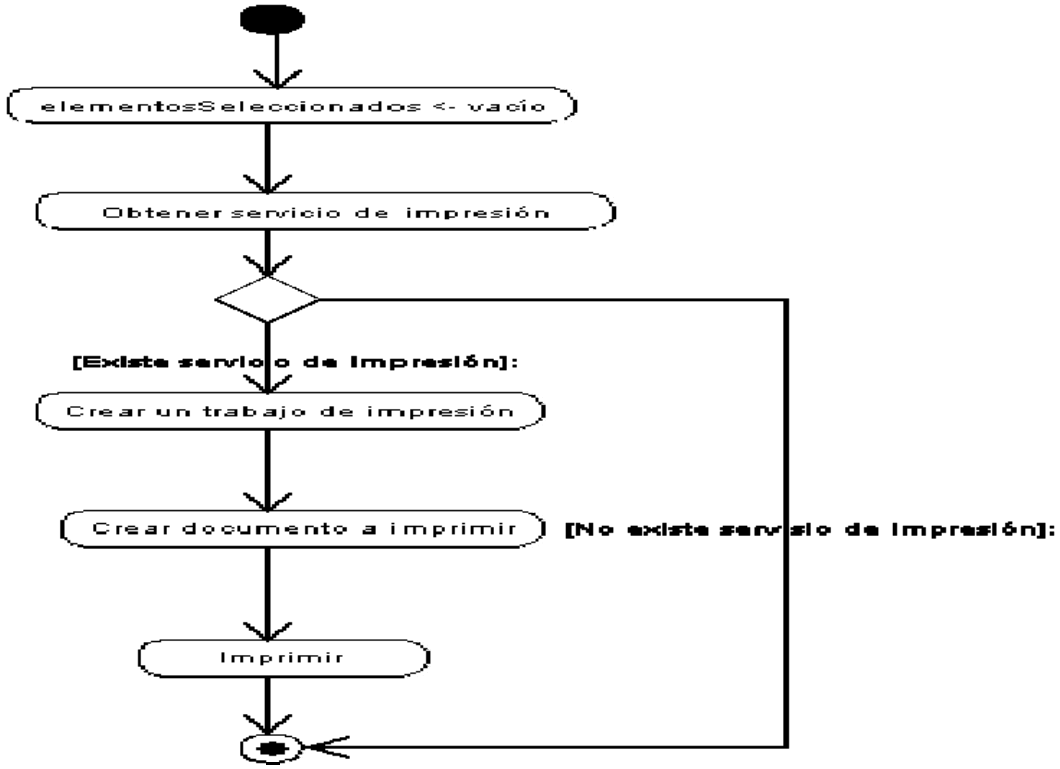
- Figura 3.14. Mover elemento contenido -



- Figura 3.15. Resise elemento contenido -

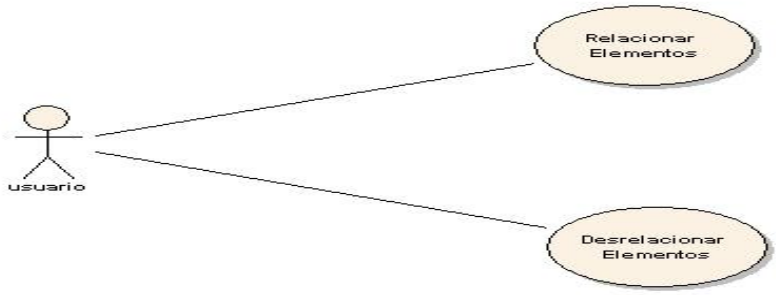


- **Figura 3.16. Eliminar elemento contenido –**



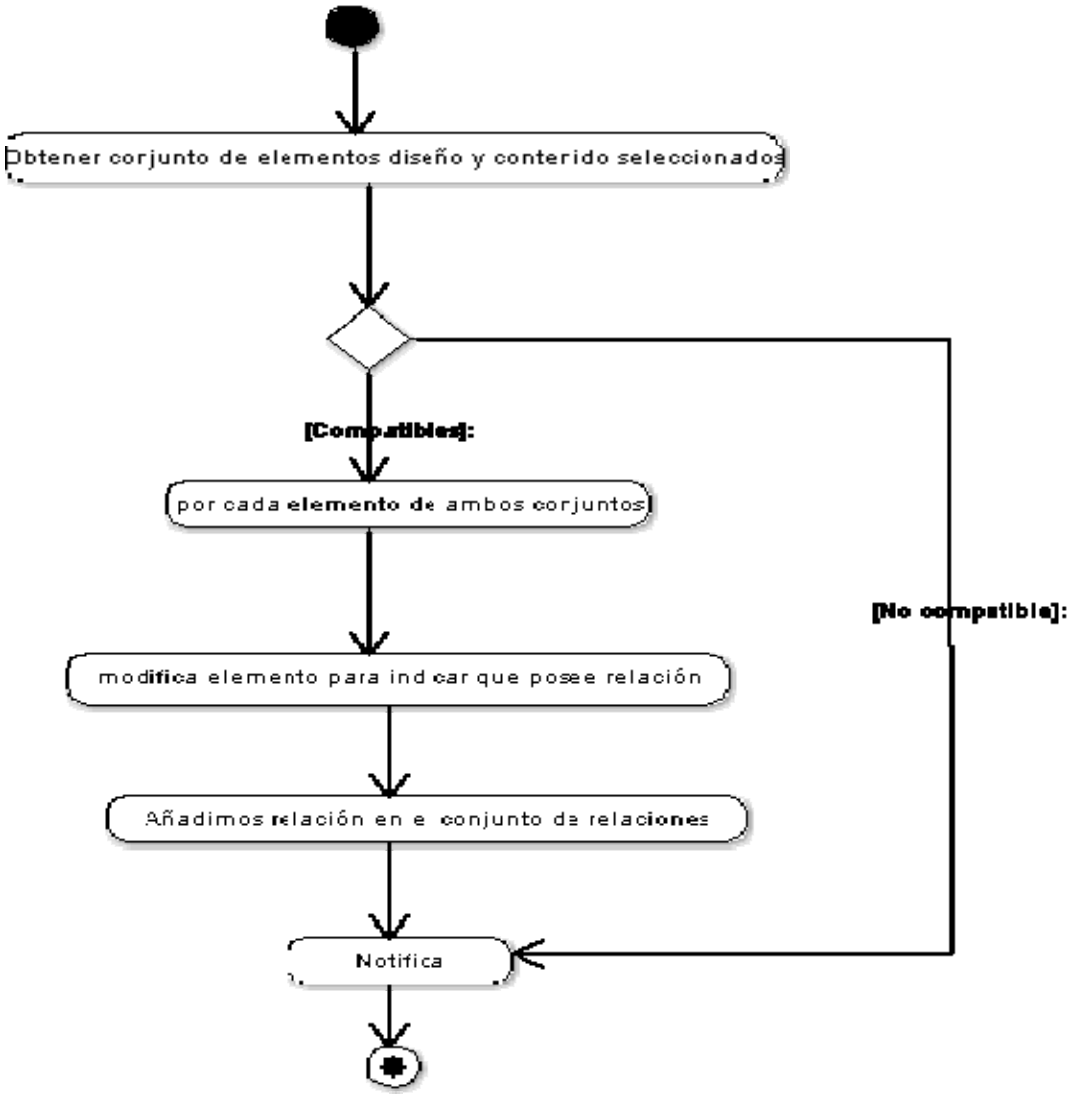
- Figura 3.17. Imprimir contenido -

- **Nivel Relación**
  - **Diagrama de casos**

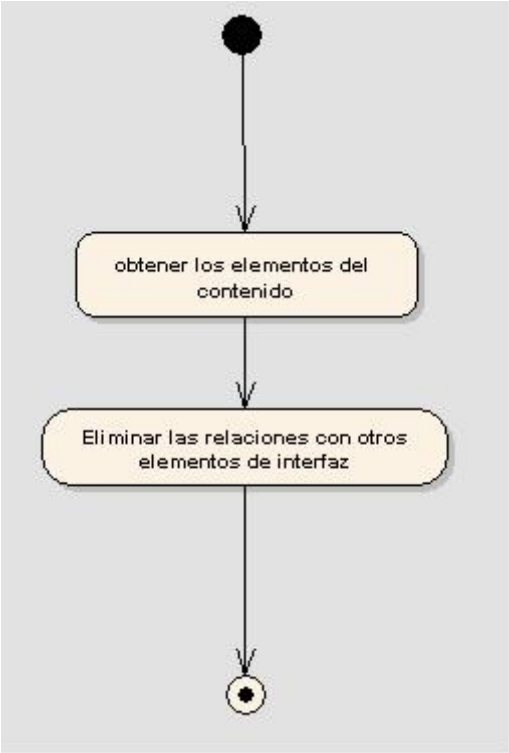


- **Figura 3.18. Diagrama de casos de uso del módulo relación -**

  - **Diagrama de actividades**



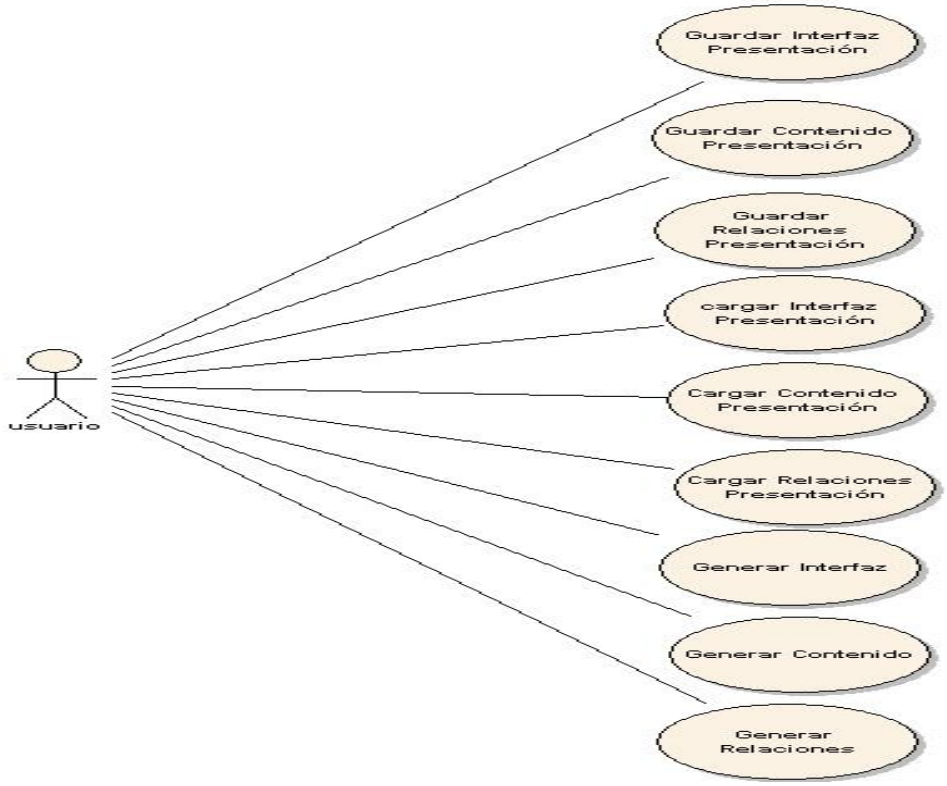
- Figura 3.19. Relacionar elementos diseño con elementos contenido -



- Figura 3.20. Eliminar relación entre elementos diseño y contenidos -

3.3 Módulo Persistencia

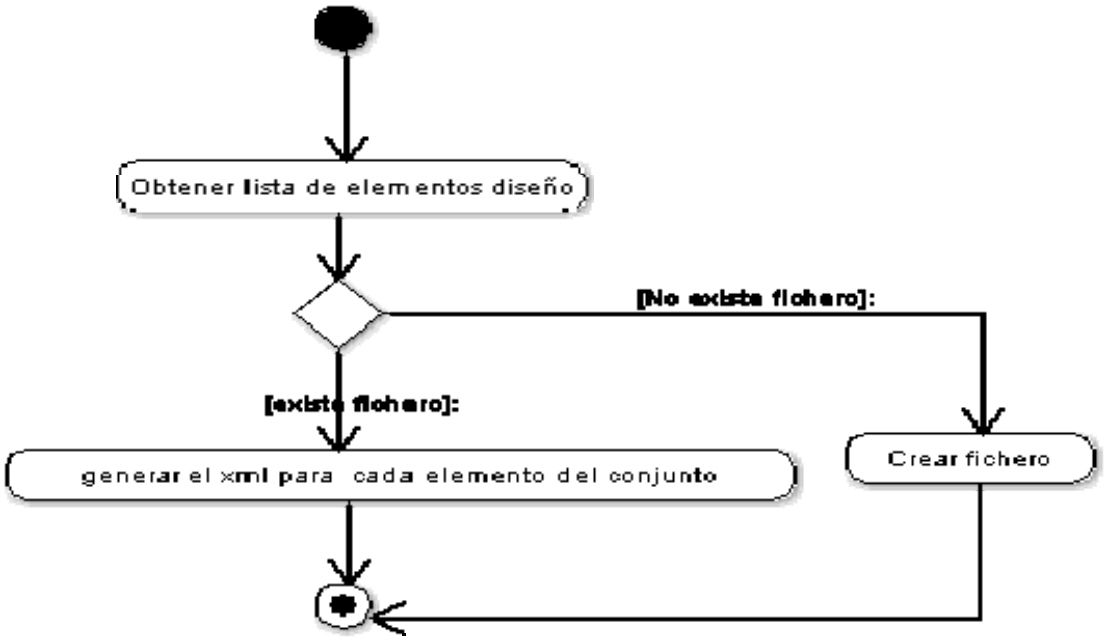
- **Diagrama de casos de uso**



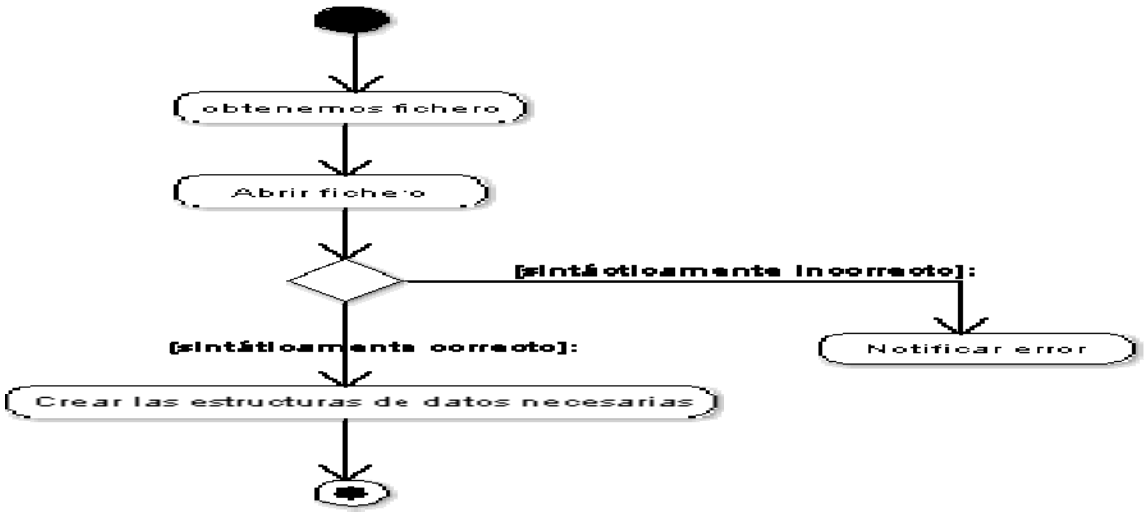
- **Figura 3.21. Diagrama de casos de uso del módulo persistencia -**

- **Diagramas de actividades**

De forma similar a los apartados se nos presenta una situación de similitud en los diagramas de diferentes funcionalidades que nos llevarían a escribir dibujos semejantes, por ello y para no llenar esta memoria de dibujos que no aporten nada nuevo a la compresión del trabajo, nos limitamos a poner un diagrama para cada caso. La funcionalidad de este modulo es básicamente guardar, se nos permite guardar desde el diseño hasta el prototipo, pasando por contenido y relación. De igual modo pasa lo mismo con cargar.



- **Figura 3.22. Diagrama de actividades de guardar diseño –**

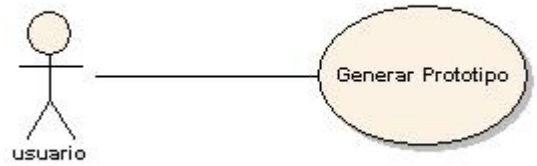


- Figura 3.23. Diagrama de actividades de cargar diseño -

Todos los demás cargar seguirán un diagrama similar, únicamente se diferenciarán en la dtd que marca si un fichero es válido o no. Además decir que el cargar relación, necesita que previamente se hayan cargado diseño y contenido. Análogamente todos los guardar son iguales, solo se diferencian en el xml necesario para generar cada uno de los elementos tanto de la lista de elementos diseño como contenido.

3.4 Módulo Prototipado

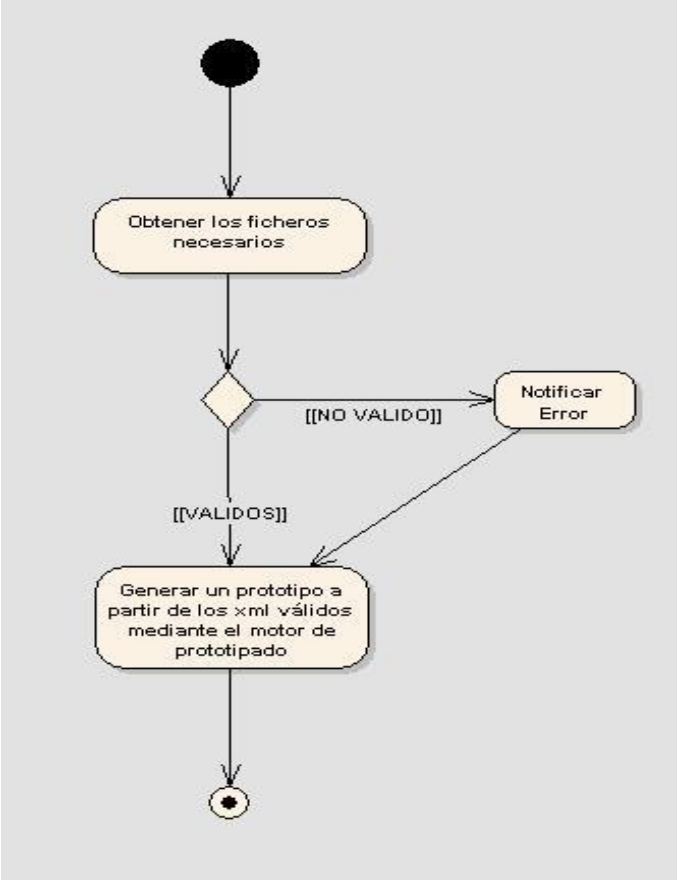
- **Diagrama de casos de uso**



- **Figura 3.24. Diagrama de casos de uso del módulo prototipado -**

Esta funcionalidad es implementada por un aplicación desarrollada con anterioridad a la implementación de esta herramienta case. Es de esperar que tras la implementación de esta herramienta se consiga una integración con dicho motor de prototipos. Dicho motor lo que hace es basándose en lo que se ha diseñado anteriormente tanto desde el punto de vista del interfaz como del contenido, así como sus relaciones, genera un ejemplo físico de aplicación.

- **Diagrama de actividades**



- Figura 3.25. Diagrama de actividades para generar un prototipo –

4. Diagramas de clases e interacción

4.1 Introducción

En este apartado de la memoria nos vamos a centrar en ver cuál es la arquitectura del sistema, es decir que clases, interfaces y como se van a relacionar entre ellos para llegar a cabo la solución, esto lo veremos con el diagrama de clases, en el cual se explicará cuales han sido los mecanismos a nivel de patrones de desarrollo se han utilizado para resolver los distintos problemas que se nos han planteado a lo largo del desarrollo de la herramienta case.

Por otro lado vamos a mostrar los diagramas de interacción, más concretamente, los diagramas de interacción que se muestran son del tipo de secuencia. Estos diagramas lo que nos van a permitir ver es cómo interaccionan los objetos, que pasos de mensajes hacen para llevar a cabo la funcionalidad requerida.

Nos vamos a centrar en mostrar los diagramas de secuencia de las funcionalidades de las que se han indicado los diagramas de actividad anteriormente. Estos diagramas pueden ser lo más refinado posible, ya que de los propios diagramas de secuencia se puede obtener código ejecutable, lo que ocurre es que esto implica una serie de dibujo muy grandes que pueden llegar a ser muy complicados de incluir en las dimensiones que una página de DINA-4 nos ofrece. Es por esto último que hemos llegado a un compromiso entre el grado de refinamiento de la funcionalidad que se muestra para la compresión de la misma y las limitaciones de impresión.

4.2 Arquitectura de la solución

Nuestra aplicación tiene tres patrones que entendemos que deben aparecer como mínimo en cualquier aplicación con cierta envergadura, estos patrones son *Observers, Factoría Abstracta y Fachada.*

El patrón observes es implementado para tener perfectamente separado lo que sería el modelo de sus presentaciones y de los controladores, en Java podríamos a hablar por un lado de modelo y por otro lado de controladores/Observer, ya que los controladores pueden aparecer como clases privadas de las clases que implementan la interfaz gráfica o incluso ser la propia interfaz gráfica la que implemente los controladores. Sin embargo nosotros hacemos una clara distinción entre los Observers, los controladores y las interfaces gráficas que se encuentran en paquetes distintos y que por tanto no se ven si no a través de interfaces, de esta forma se puede cambiar fácilmente la presentación de la aplicación sin que esto afecte en demasía al modelo y los controladores.

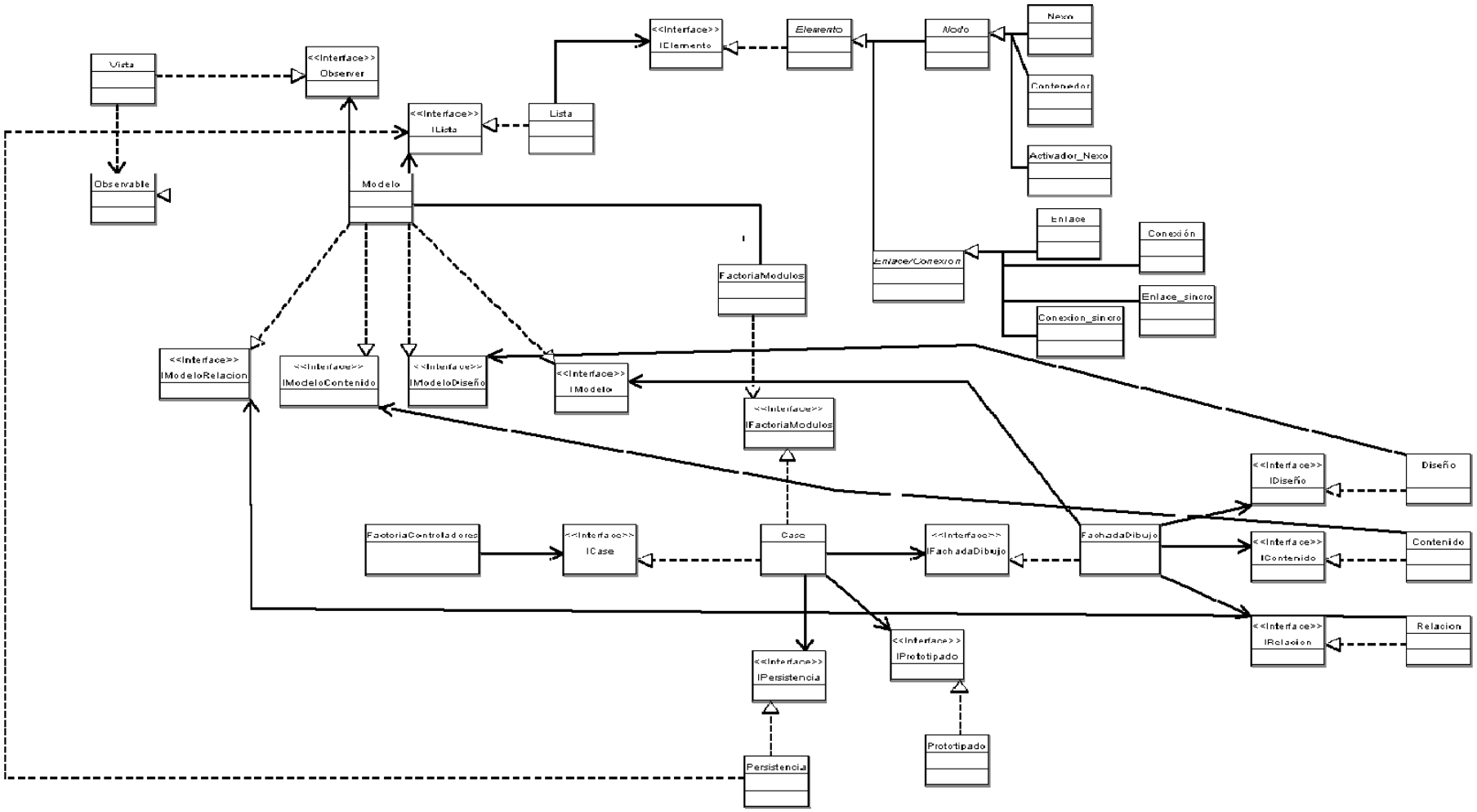
Tenemos una factoría Abstracta en dos puntos de nuestro diseño, una es *Factoría Módulos*, el cuál se encargara de crear los objetos que implementan la funcionalidad de la fachada, de la cuál hablaremos en el siguiente punto. La otra factoría es *Factoría Dibujo*, el cual se encarga de crear los distintos objetos que se encargan de la funcionalidad de la segunda fachada que divide el modulo dibujo en tres partes que se encargan cada una de ellas de diseño, contenido y relación.

El patrón de fachada, pues ya lo hemos ido explicando por el camino, consiste en modularizar la funcionalidad, de forma que se agrupan las funcionalidades que conceptualmente son similares en un modulo y se define un interfaz para que la fachada sea totalmente independiente de quien sea el que implemente dicha funcionalidad pedida.

Para la persistencia hemos utilizado *fabricación pura*, es decir el propio objeto no sabe como guardarse, quién sabe como guardar los objetos son clases especializadas en esa labor.

Modelo de Datos

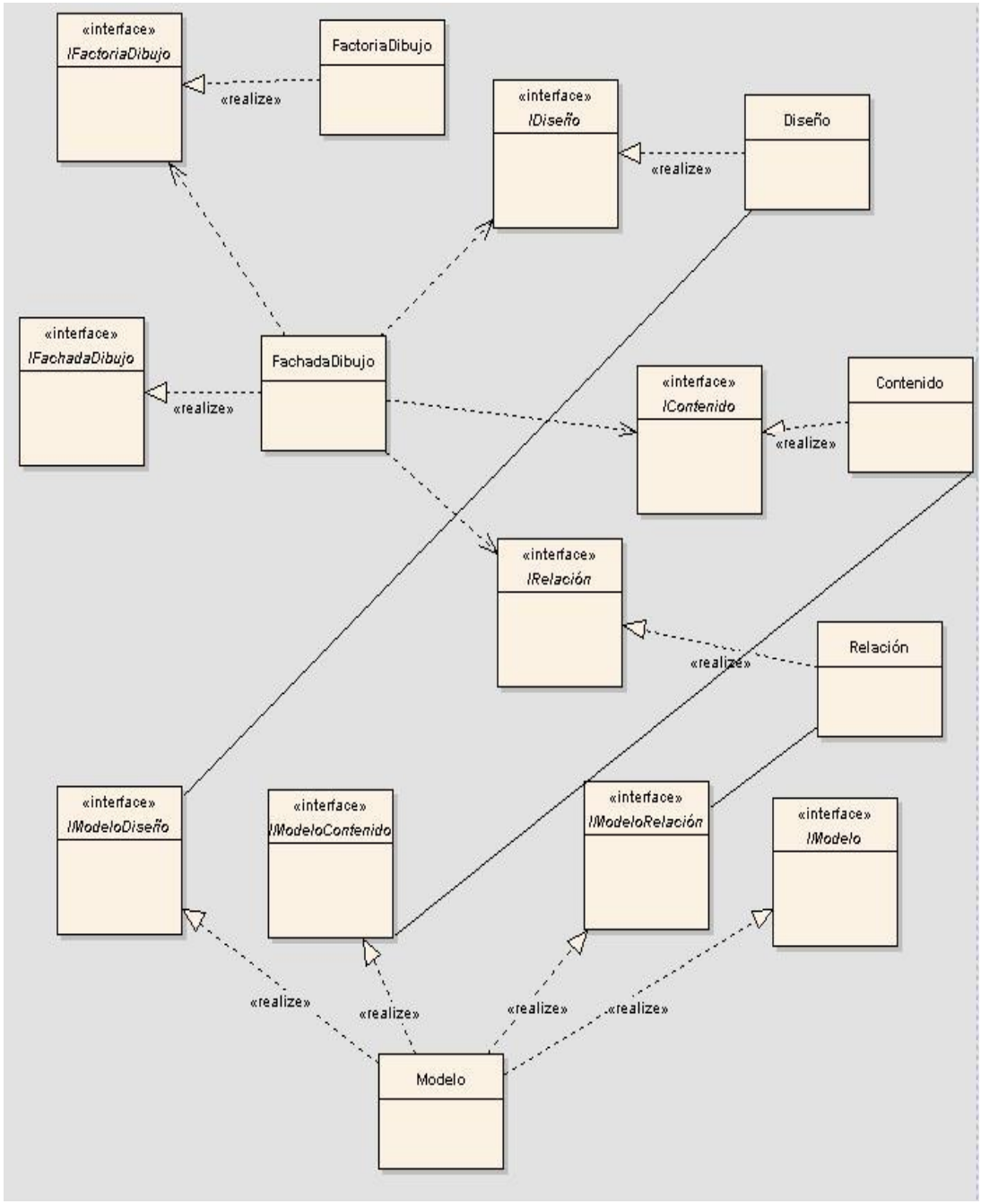
La jerarquía de clases utilizada para clasificar los elementos a dibujar por nuestra aplicación es la típica que se puede encontrar en los libros de introducción a la programación orientada objetos. Tenemos un objeto superior que *Elemento*, es cuál es abstracto y que básicamente define lo que es común a todos los objetos gráficos de la aplicación. Este a su vez es padre de otras dos clases que son *Nodo* y *ConexionEnlace*, ésta es una clasificación un poco más fina, que trata de separar, digamos las figuras polígonos, de las relaciones entre nodos. Todos los objetos se tiene que pintar pero realmente quien sabe como se debe pintar es el propio objeto. Con esta clasificación conseguimos tratar los objetos mediante listas de *Elementos,* es decir, polimórfimica. Posteriormente cada módulo depende de una o varias listas de elementos.



- Figura 4.1 Diagrama de clases completo -

4.3 Módulo Dibujo

- **Diagrama de Clases**



- Figura 4.2. Diagrama de clases del módulo dibujo -

Ahora nos centramos en hacer una descripción de cuál es el papel que juega cada clase en el diseño de este módulo.

IFachadaDibujo, este interfaz junto con la clase que lo realiza, constituye la representación del patrón fachada. Se utiliza este patrón porque si nos damos cuenta el módulo dibujo es fácilmente divisible en tres grandes módulos independientes, a saber: Diseño ó Interfaz, Contenido y Relación. Por tanto parece bastante interesante hacer que el módulo dibujo en realidad sea una fachada la cuál aísla al módulo anterior, que solicita la funcionalidad de este módulo, de cómo está implementado.

FachadaDibujo**,** esta clase implementa la funcionalidad que ofrece el interfaz a sus clientes. Sin embargo esta clase es una mera distribuidora, ya que lo único que va a hacer es invocar al método correspondiente de aquel módulo que realice la funcionalidad que le solicitan. Para un menor acoplamiento lo que hemos hecho ha sido que esta clase solo conoce a los interfaces y no a los módulos que realmente llevan a cabo la funcionalidad, de esta forma podemos algún día cambiar uno de esos módulos sin que la fachada se de cuenta de dicho cambio, obviamente ese módulo debe implementar la interfaz que conoce la fachada.

Diseño, Contenido, Relación, estas clases obviamente desempeñan el mismo papel, pero cada una de ellas se encarga de diferente funcionalidad. Son producto de dividir la funcionalidad, que debe ofrecer la fachada, desde el punto de vista de acercamiento conceptual de la misma. Como hemos mencionado anteriormente la fachada no conoce a estas clases si no a los interfaces ***IDiseño, IContenido, IRelación***, los cuales implementan respectivamente.

IFactoriaDibujo y FactoriaDibujo, este interfaz y esta clase, representan la implementación del patrón factoría abstracta. Igual que antes, para evitar que la fachada conozca las clases que implementan realmente la funcionalidad que se solicita, la fachada no crea directamente los objetos si no que se los pide a la factoría la cuál si que conoce como se deben construir dichos objetos, pero le devuelve a la fachada un Interface por el motivo que se ha explicado anteriormente.

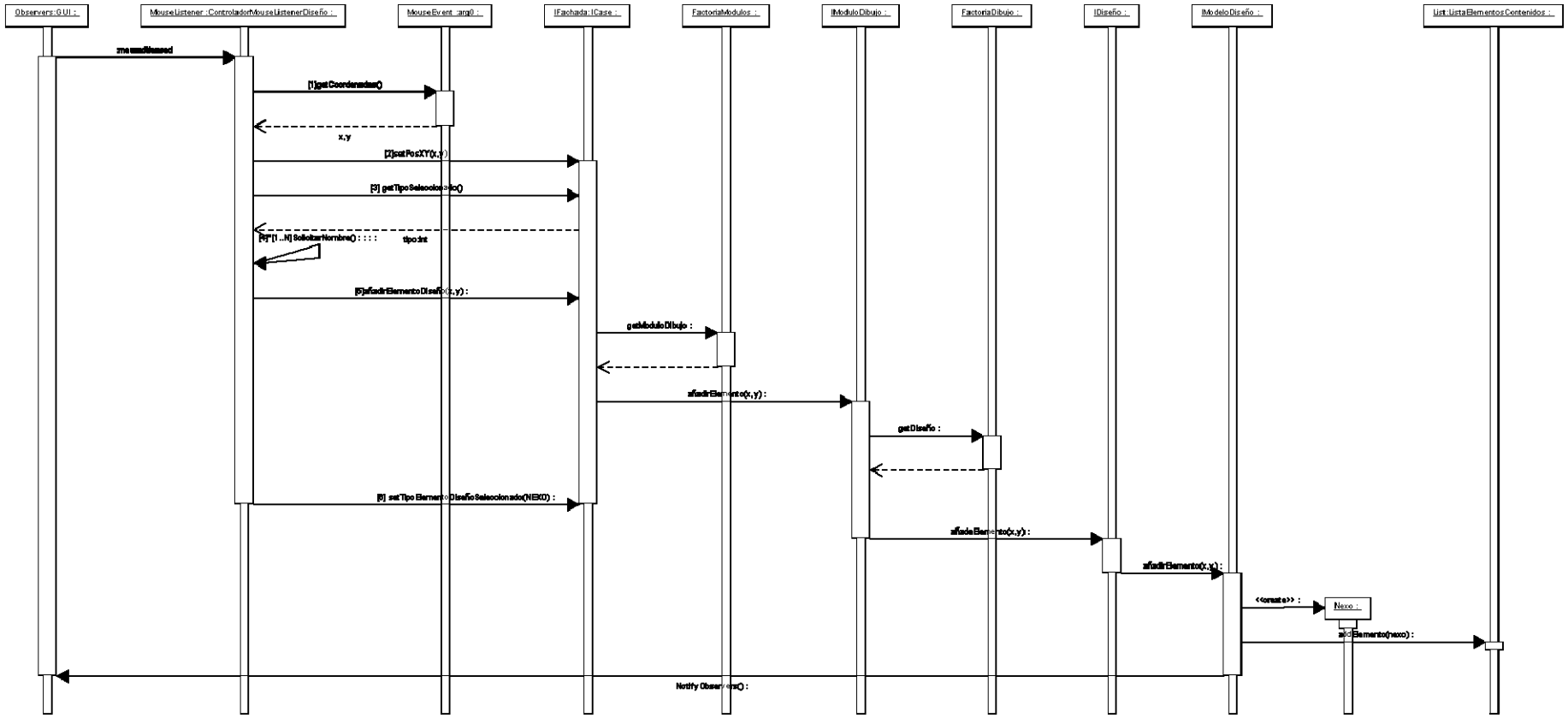
El modelo de datos que se maneja fue explicado en un punto anterior ya que es común a todos los módulos y así evitamos repetir lo mismo en varios puntos, lo que llevaría a que la memoria se extendiera sin aportar cosas nuevas.

Ahora nos vamos a centrar en ver los distintos diagramas de interacción, obviamente no vamos a poder mostrar todos, ya que son muchos y algunos no son de gran importancia. Por ello solo nos vamos a centrar en aquellas funcionalidades que sean de cierta importancia.

Como se explico en el punto anterior de los diagramas de actividades, los diagramas de secuencia de las funcionalidades de dibujar nexo, dibujar contenedor, etc. (es decir, dibujar los elementos nodo del diseño) son análogos. Por ello, y para no llenar la memoria de dibujos prácticamente iguales, hemos decidido desarrollar sólo uno de ellos, en este caso dibujar nodo nexo.

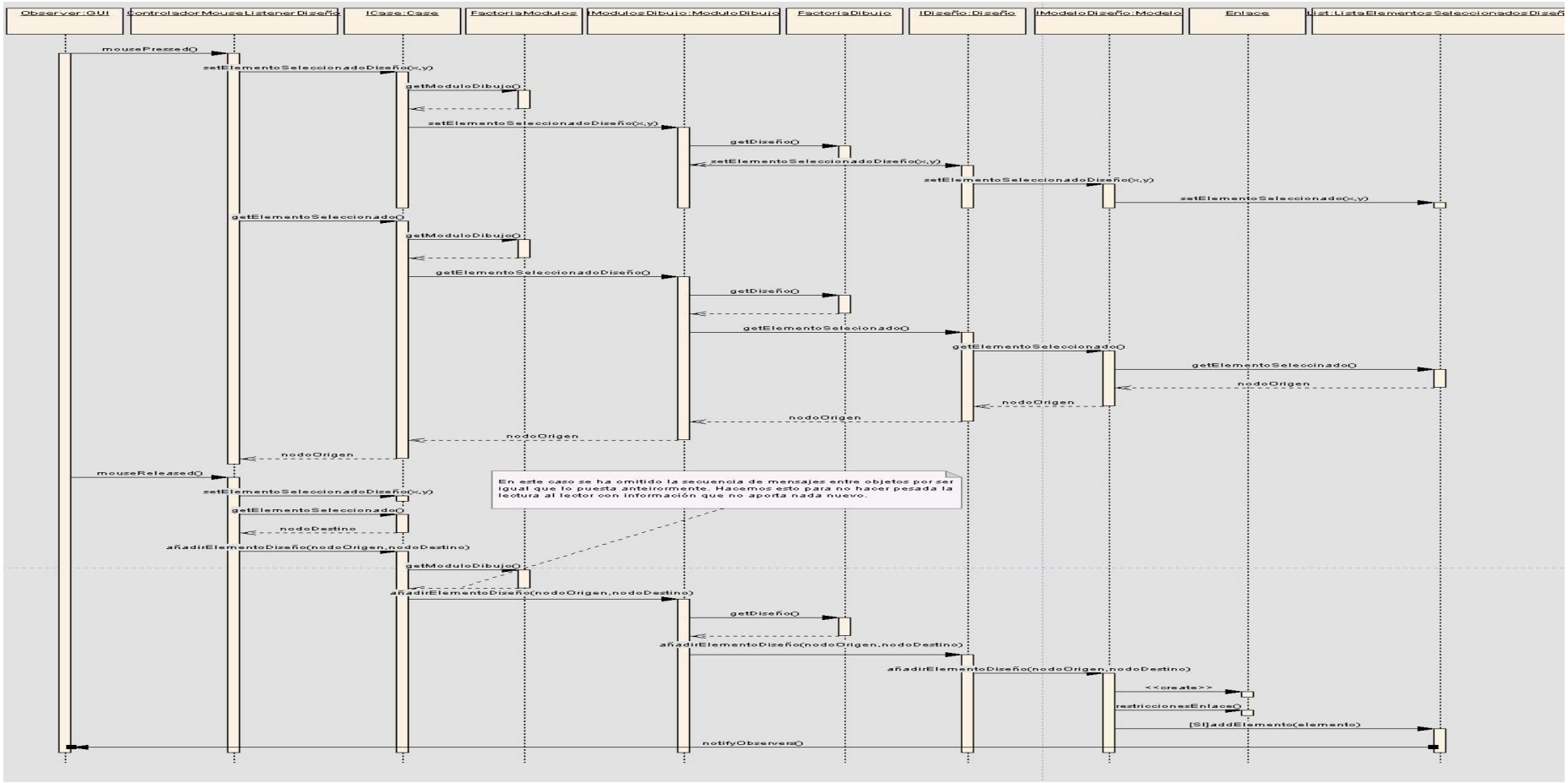
- **Diagramas de interacción**
  - **NIVEL INTERFAZ**

DIBUJAR NEXO



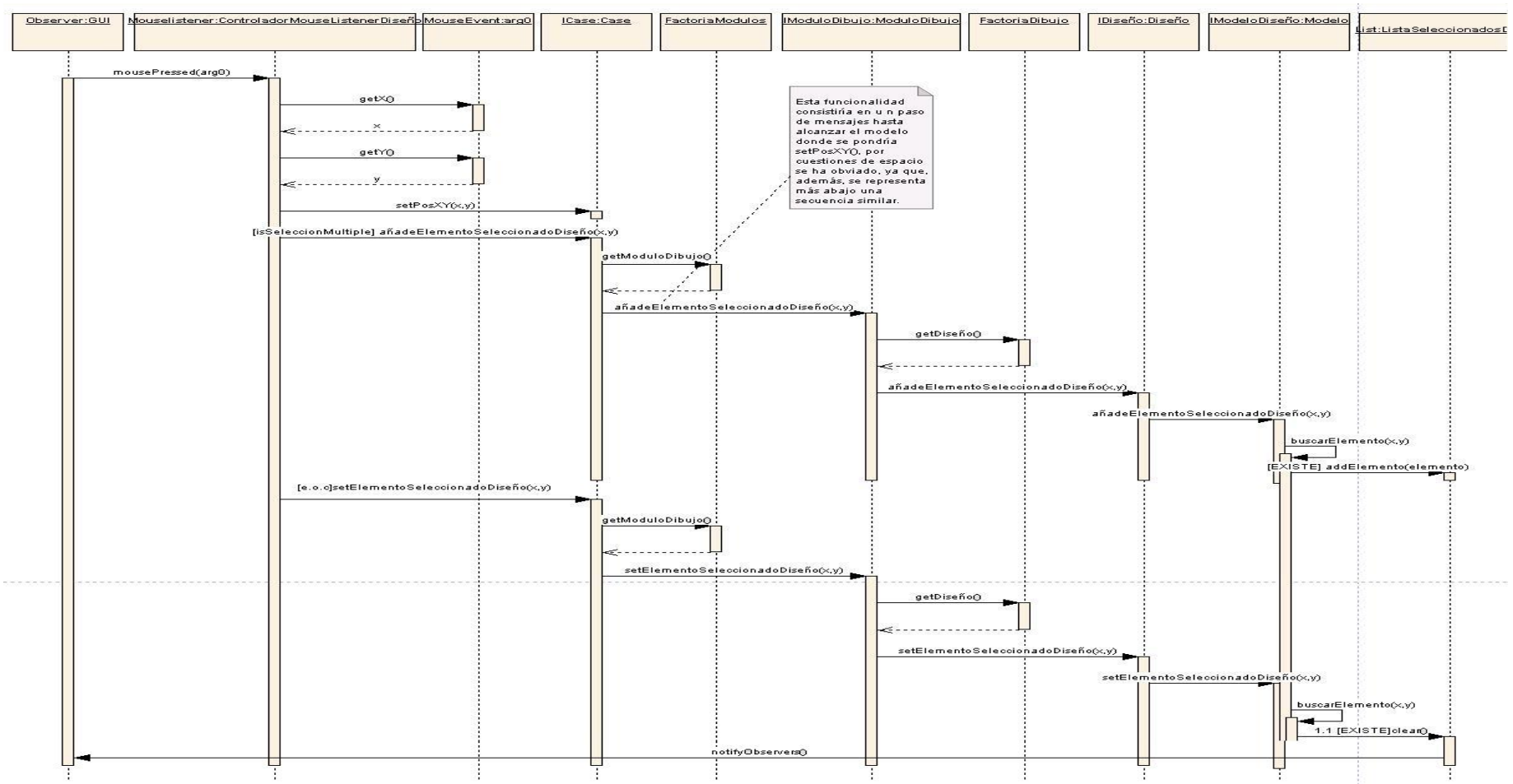
- Figura 4.3. Diagrama de interacción para dibujar un nodo nexo -

DIBUJAR ENLACE



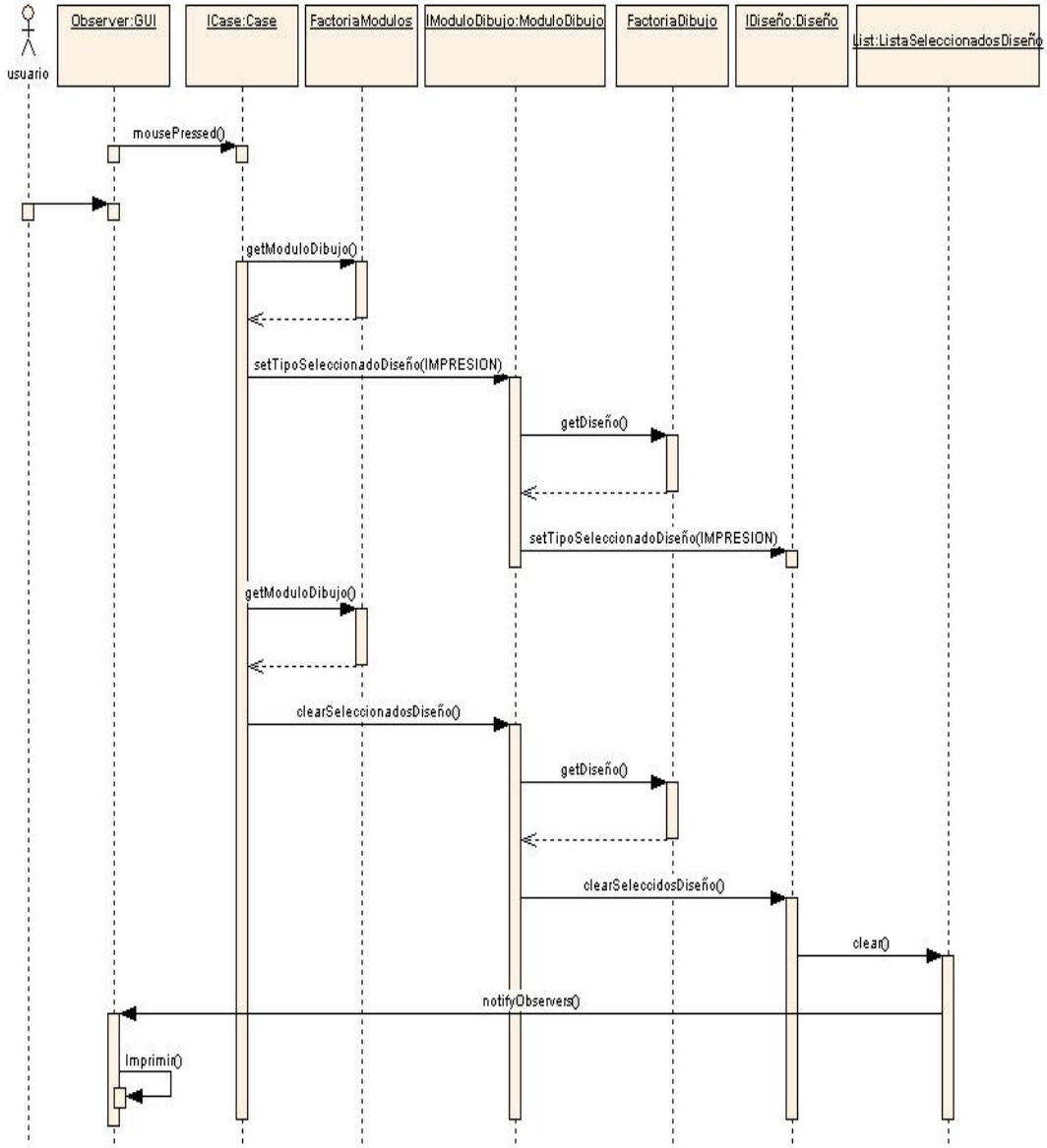
- Figura 4.4. Diagrama de interacción para dibujar un enlace -

SELECCIONAR ELEMENTO DE INTERFAZ



- Figura 4.5. Diagrama de interacción para seleccionar un elemento del interfaz -

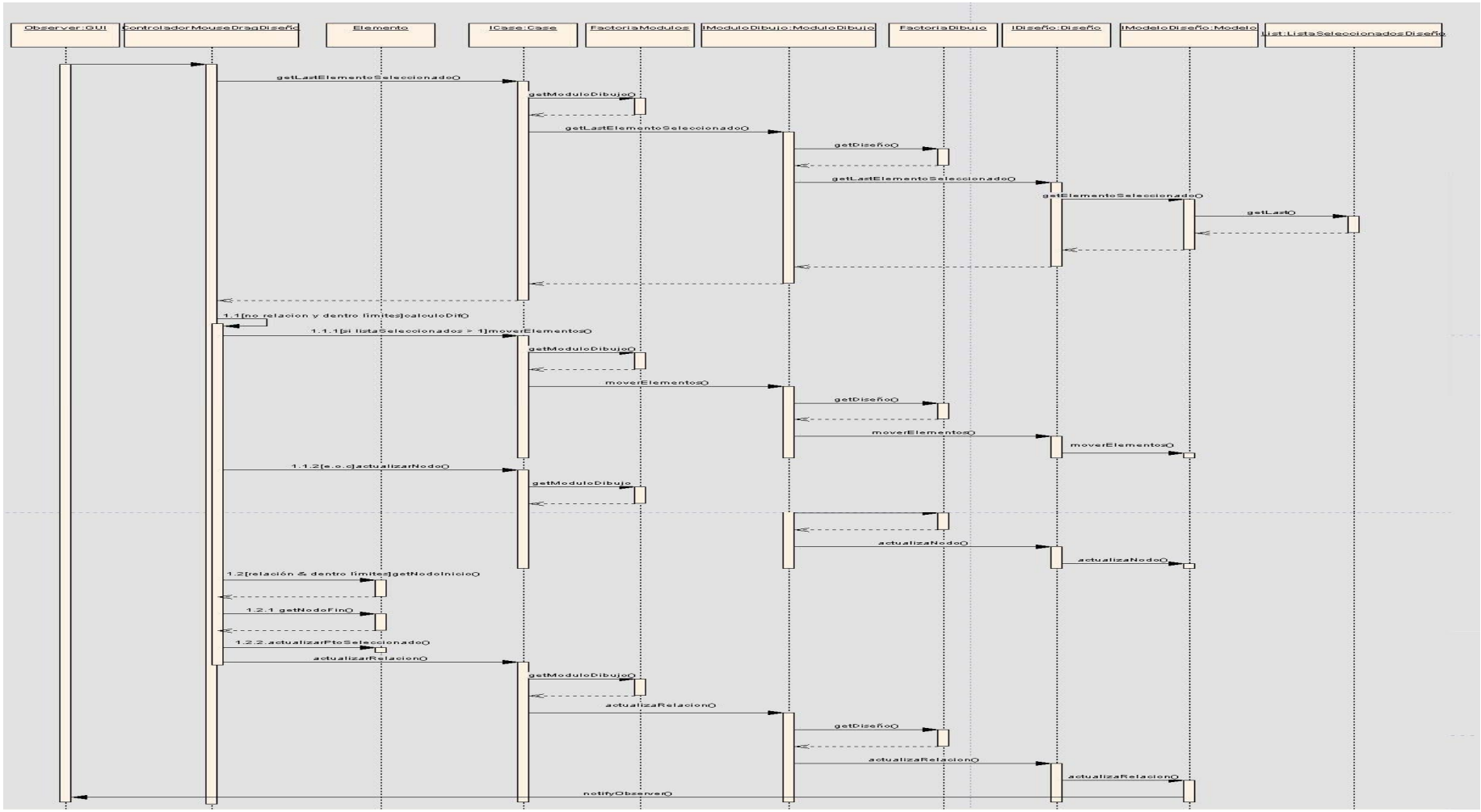
IMPRIMIR PIZARRA INTERFAZ



-Figura 4.6. Imprimir pizarra Interfaz –

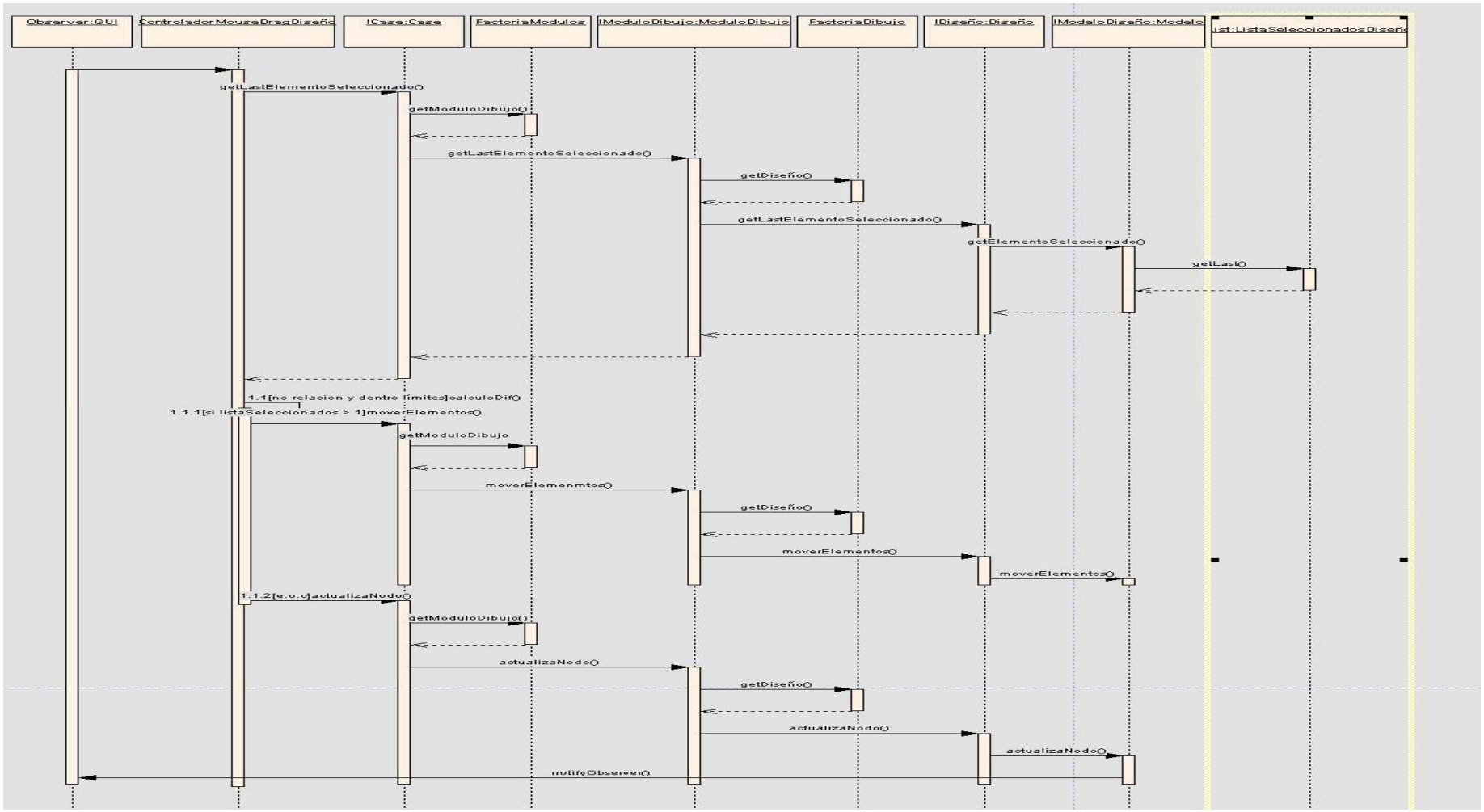
De forma análoga al diagrama anterior, este diagrama es igual al imprimir pizarra contenido, por ello y por no llenar la memoria de diagramas que no apartan nada nuevo y que solo ocupan espacio y tiempo al lector, estos diagramas junto a los de mover, resize y eliminar elemento se plasmaran con diagramas solo a nivel de interfaz, dejando constancia de que son similares.

Mover Elemento



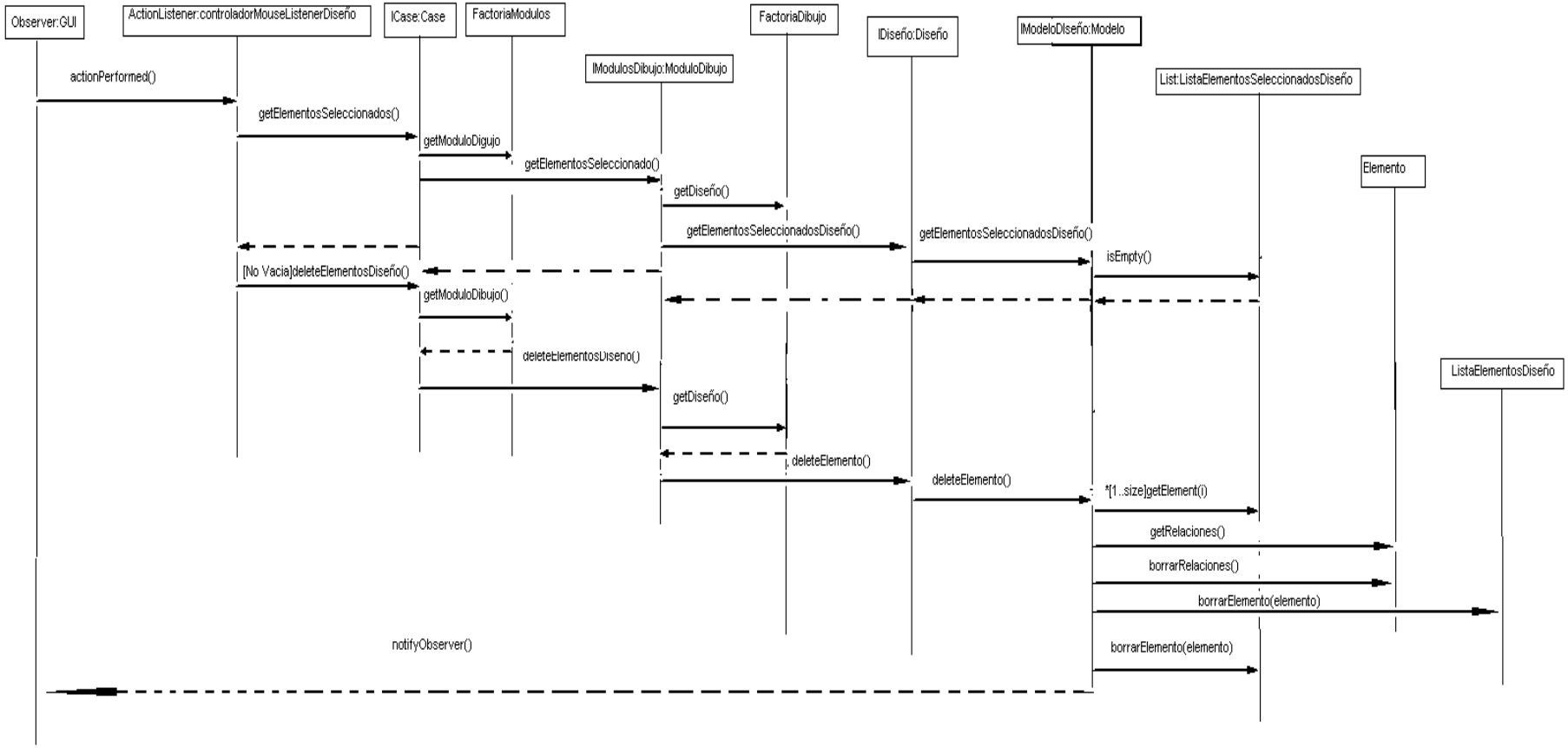
- Figura 4.7. Diagrama de interacción para mover elemento -

RESIZE ELEMENTO



- Figura 4.8. Diagrama de interacción para cambiar el tamaño de un elemento -

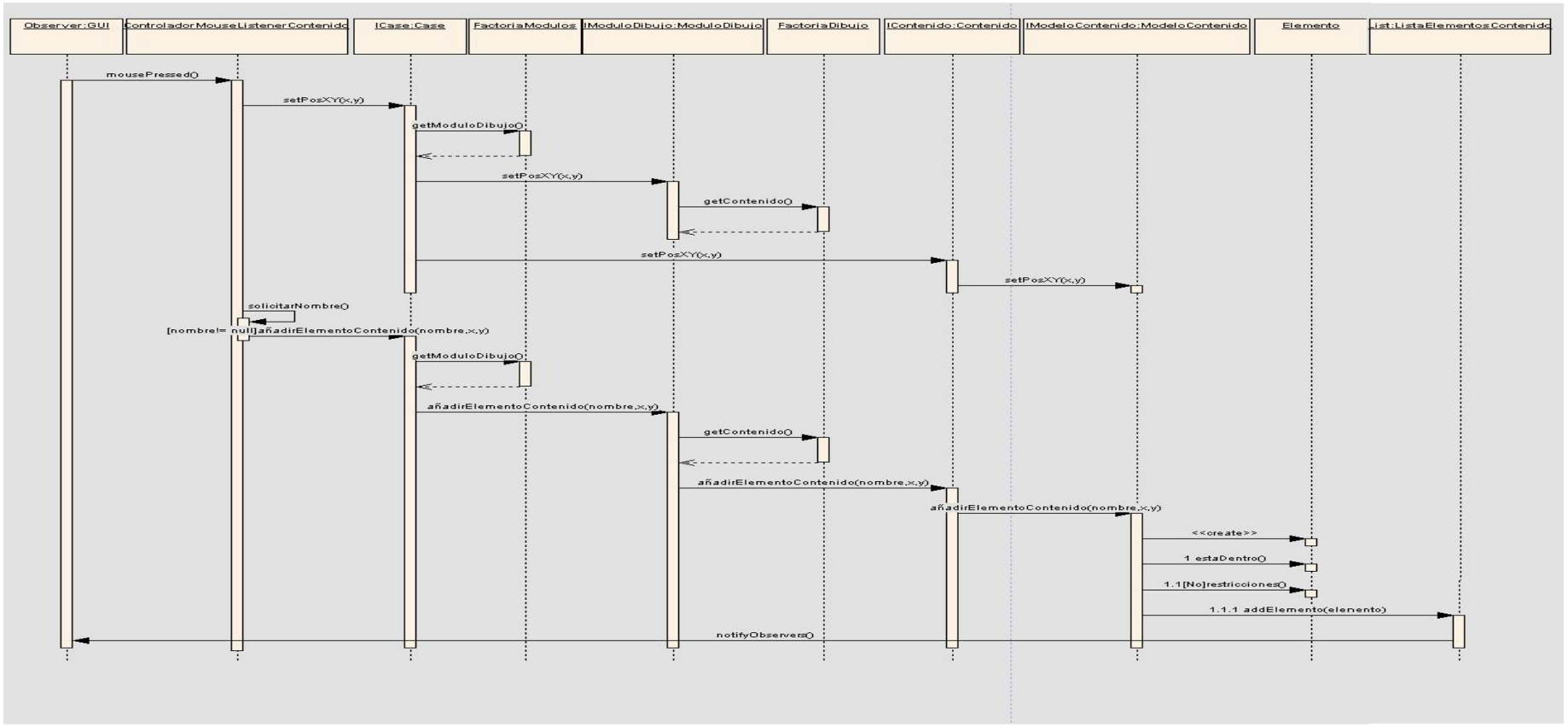
ELIMINAR ELEMENTO



- Figura 4.9. Diagrama de interacción para eliminar un elemento -

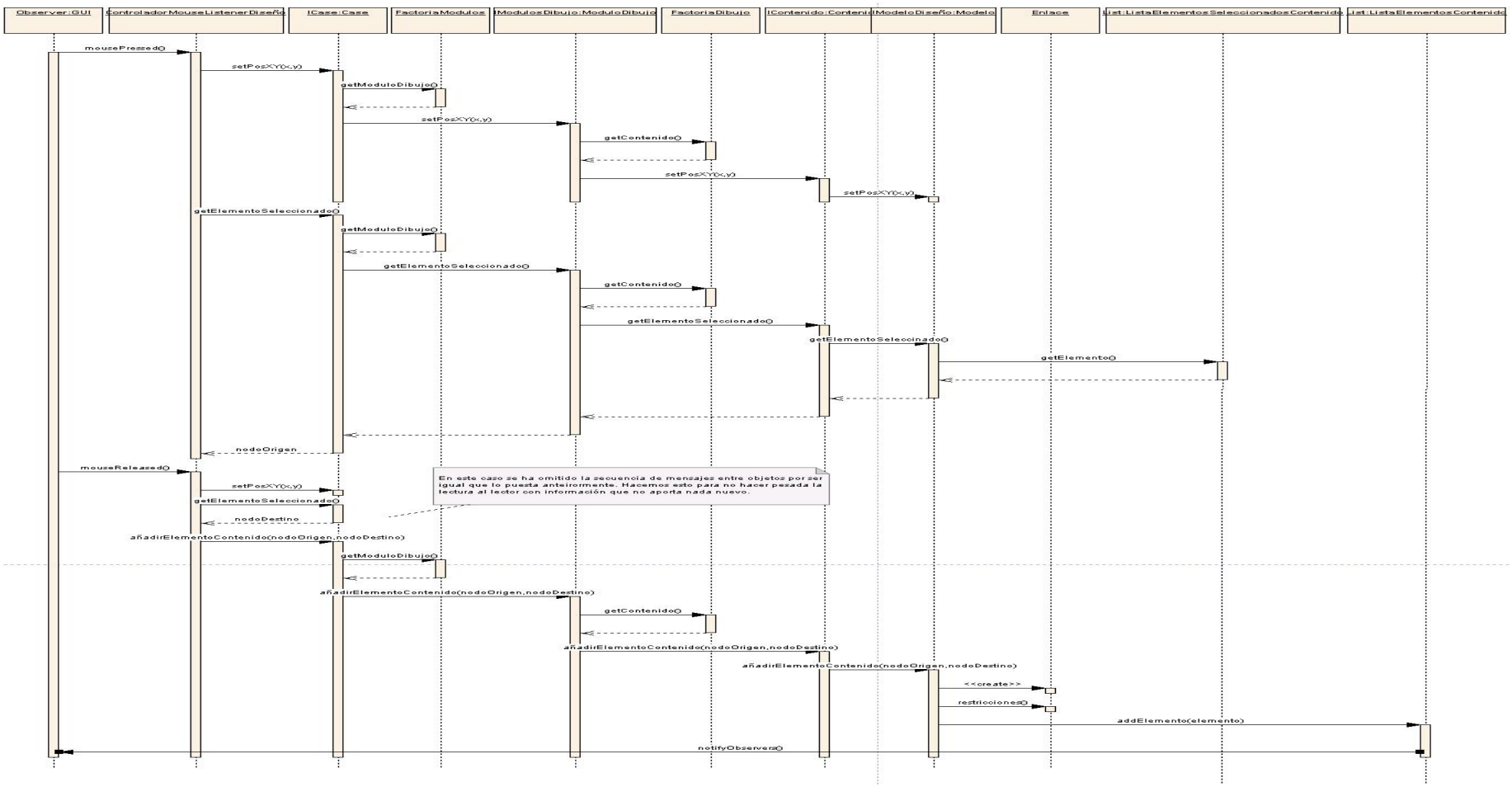
- **Nivel Contenido**

DIBUJAR CONTENIDO ESTÁTICO



-Figura 4.10. Diagrama de interacción para dibujar un contenido estático -

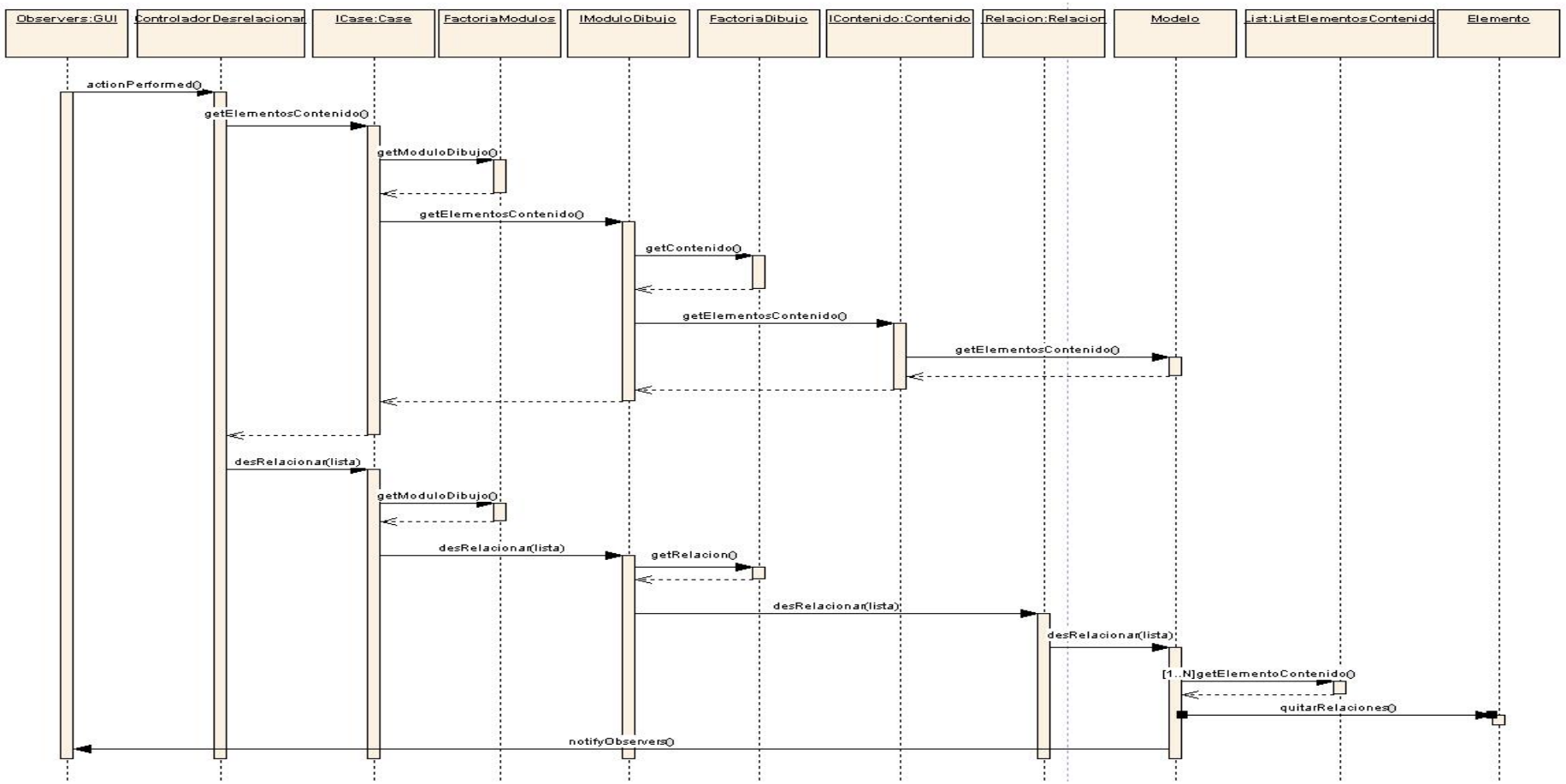
DIBUJAR ENLACE ESTÁTICO



-Figura 4.11. Diagrama de interacción para dibujar un enlace estático-

- **Nivel Relación**

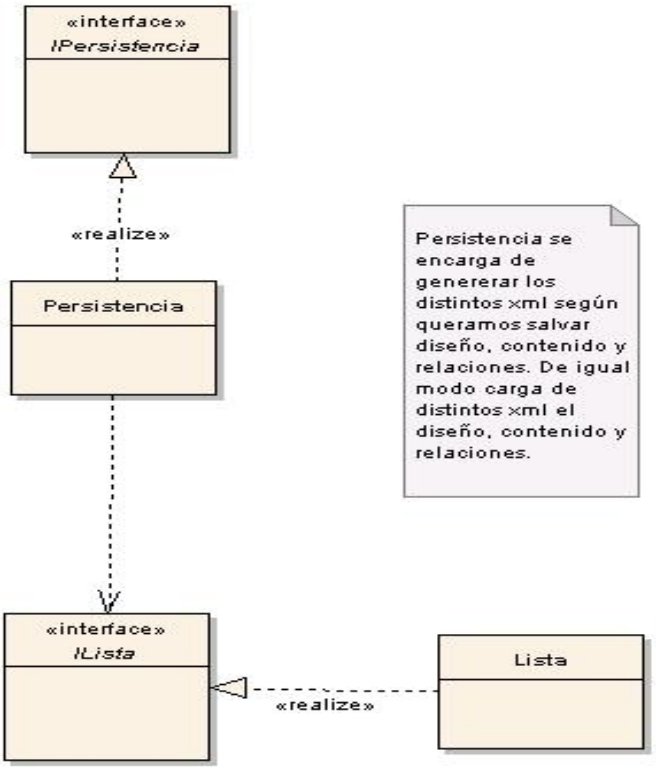
DESRELACIONAR ELEMENTOS



- Figura 4.12. Diagrama de interacción para la eliminación de las relaciones entre elementos -

4.2 Módulo Persistencia

- **Diagrama de clases**



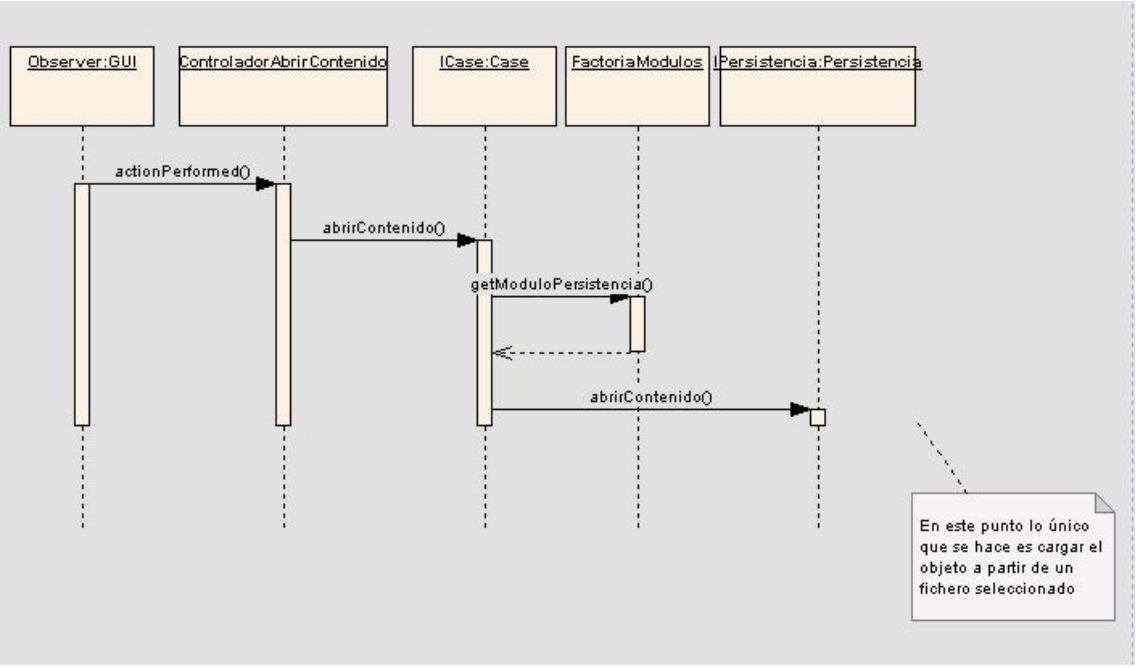
- Figura 4.13. Diagrama de clases para el módulo Persistencia -

Igual que en el módulo anterior ahora nos centramos en ver que papel juega cada una de las clases.

Nos basamos sencillamente en una fachada, al igual que se hizo con el módulo dibujo, en este caso *Persistencia* hace de Fachada, para evitar un acoplamiento de la fachada general con este módulo lo que se hace es que *Persistencia* implementa *IPersitencia* el cuál tiene como cliente la Fachada general del sistema. Este módulo se encarga de él mismo de tanto generar los distintos documentos xml para guardar el diseño, contenido y relación en la notación “formal” que necesita el motor de prototipado para poder a su vez generar prototipos.

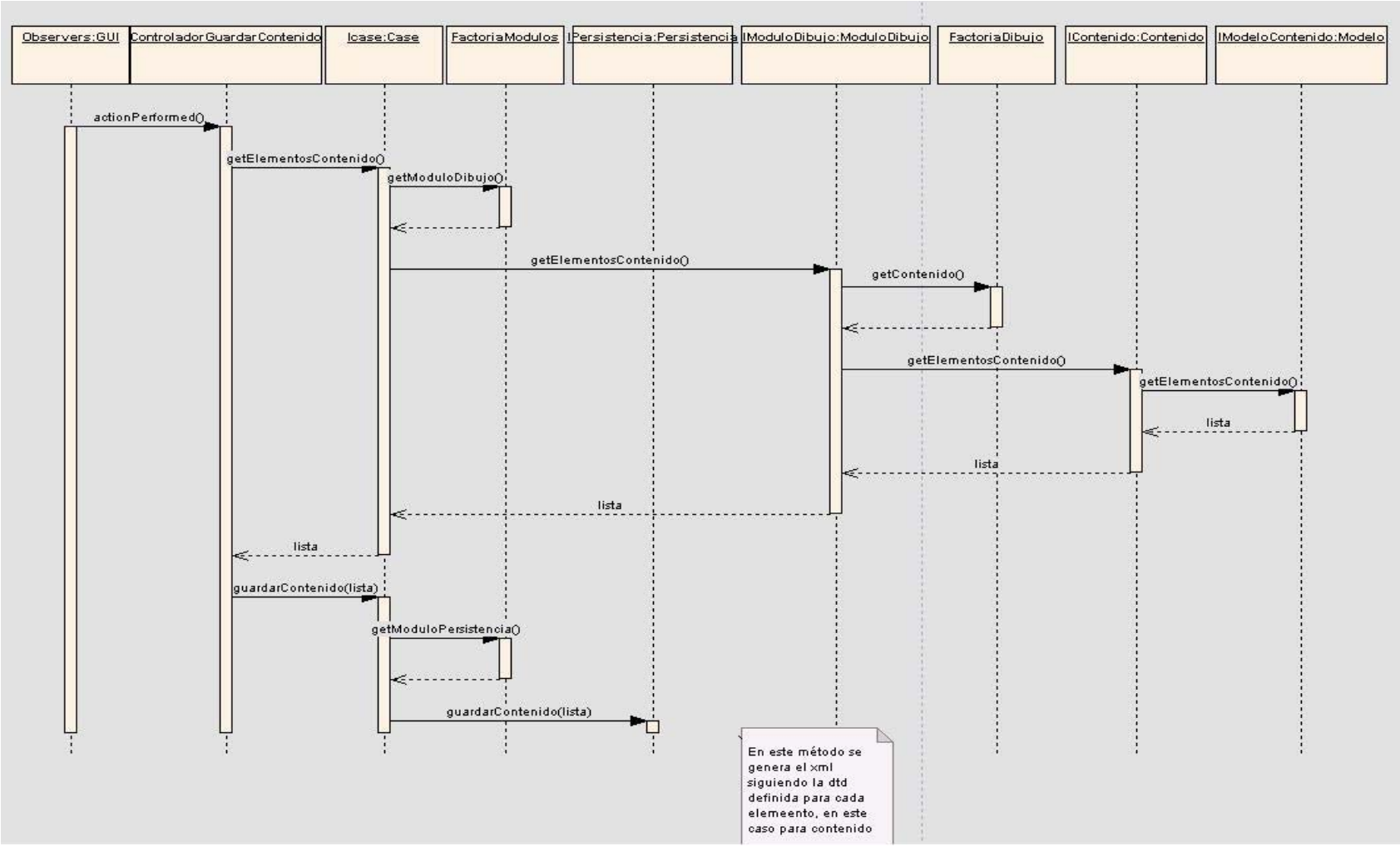
- **Diagramas de interacción**

ABRIR CONTENIDO



- Figura 4.14. Diagrama de interacción para abrir un contenido -

GUARDAR CONTENIDO



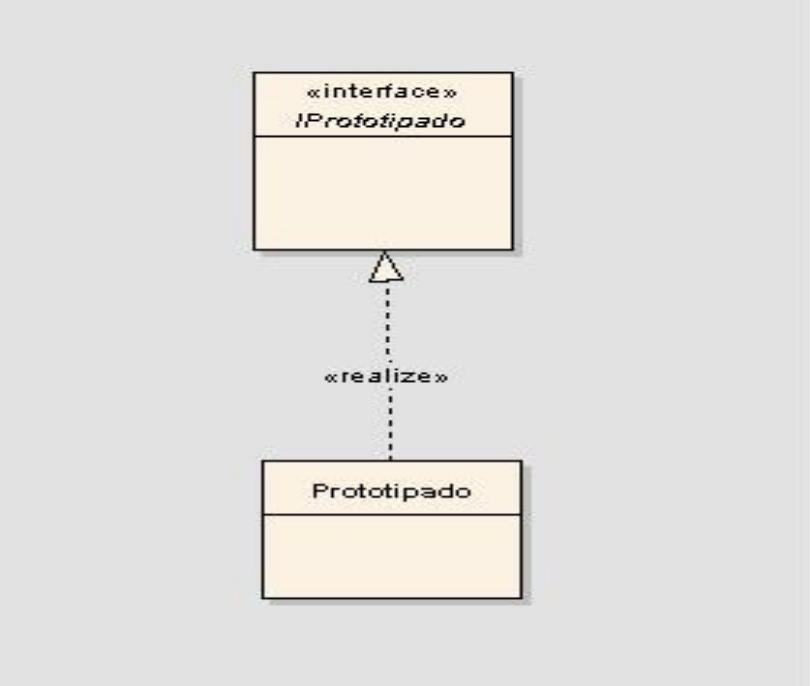
- Figura 4.15 Diagrama de interacción para guardar el contenido -

4.3 Módulo Prototipado

- **Diagrama de Clases**

Este módulo se encarga de utilizar el motor de prototipos que como hemos comentado a lo largo de la memoria estaba previamente implementado al comienzo de este proyecto.

Por tanto esta parte del código consiste en un módulo que implementa un interfaz, el cuál ofrece una serie de servicios invocables desde la fachada principal de la aplicación. Esta clase se llama *Prototipazo* y el Interface que implementa se llama *IPrototipado*.

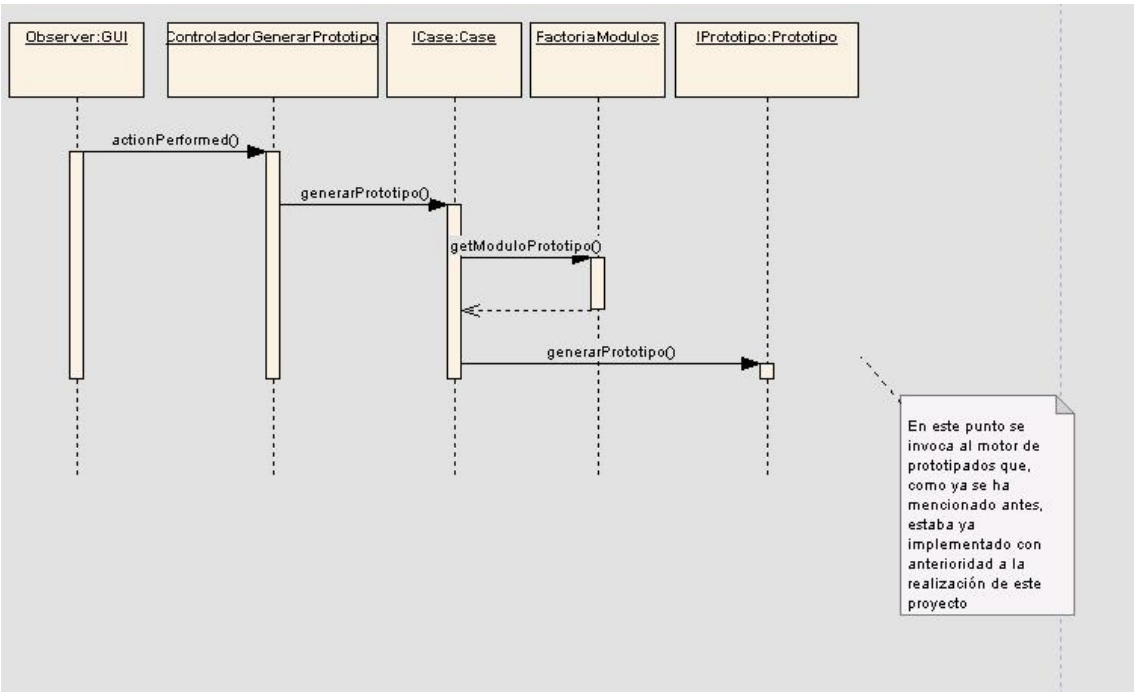


- Figura 4.16. Diagrama de clases del módulo prototipado -

Ahora nos centraremos en los diagramas de interacción para las funcionalidades que desempeña este módulo.

- **Diagramas de interacción**

GENERAR PROTOTIPOS



-Figura 4.17. Diagrama de interacción para generar prototipos –

5. Conclusiones

El desarrollo de este proyecto ha permitido demostrar que las ideas sobre la herramienta CASE PlumbingMatic descritos en la tesis Conceptualización, Prototipado y Diseño de Aplicaciones Hipermedia eran totalmente correctas. Además, su integración con el motor GAP ha permitido desarrollar una herramienta que facilita el prototipado rápido de aplicaciones hipermedia basadas en un modelo.

Desde el punto de vista del diseño cabe destacar la generación de diseños que van más allá de lo que es capaz de entender el motor de prototipado para la generación de prototipos, sin embargo es de esperar que en un futuro dicho motor sea capaz de interpretarlas. Podemos decir que nuestra aplicación está fundamentada en un diseño estructurado, con alta cohesión y bajo acoplamiento, siguiendo las pautas vistas en ingeniería del software.

Desde el punto de vista docente, este proyecto ha servido para consolidar los conocimientos de programación adquiridos a lo largo de la carrera. También se puede destacar la profundización de conocimientos en la implementación de interfaces gráficas de usuario con la tecnología de swing de Java. Además hemos manejado notación XML y herramientas para analizar dichos documentos XML. Una nota a reseñar es la constatación de que una buena planificación es fundamental para que un proyecto desemboque con buen fin.

6. Glosario

Herramienta Case, De acuerdo con Kendall y Kendall la ingeniería de sistemas asistida por ordenador es la aplicación de tecnología informática a las actividades, las técnicas y las metodologías propias de desarrollo, su objetivo es acelerar el proceso para el que han sido diseñadas, en el caso de CASE para automatizar o apoyar una o mas fases del ciclo de vida del desarrollo de sistemas.

Hipermedia, El mismo W3C define *hipermedia* como: hipertexto que no está restringido a ser texto. Puede incluir gráficos, video y/o sonido. En la actualidad tienden a utilizarse ambos términos indistintamente. El concepto de hipermedia surgió en 1945 de la mano de Vannevar Bush.

Prototipo, es una aplicación hipermedia que existe físicamente y que refleja lo diseñado como interfaz, contenido y sus relaciones. Es decir, el diseño me especifica la estructura e integración de los diferentes componentes gráficos (ventanas, paneles, etc.), el contenido me indica la información que se muestra y las relaciones como y donde se muestra esa información.

Tecnología Case, La tecnología CASE supone la automatización del desarrollo del software contribuyendo a mejorar la calidad y la productividad en el desarrollo de sistemas de información y se plantean los siguientes objetivos:

- Permitir la aplicación práctica de metodologías estructuradas, las cuales al ser realizadas con una herramienta se consigue agilizar el trabajo.
- Facilitar la realización de prototipos y el desarrollo conjunto de aplicaciones.
- Simplificar el mantenimiento de los programas.
- Mejorar y estandarizar la documentación.
- Aumentar la portabilidad de las aplicaciones.
- Facilitar la reutilización de componentes software.
- Permitir un desarrollo y un refinamiento visual de las aplicaciones, mediante la utilización de gráficos.

7. Bibliografía

[Bush 45] Vannevar Bush: As We May Think. The Atlantic Monthly 176(1): 101-108 (1945)

[Conallen 99] Jim Conallen: Modeling Web Application Architectures with UML. Communications of the ACM 42(10): 63-70 (1999)

[Fraternali 99] Piero Fraternali: Tools and Approaches for Developing Data-Intensive Web Applications: A Survey. ACM Comput. Surv. 31(3): 227-263 (1999)

[HT 05] Website of the Sixteen ACM Conference on Hypertext and Hypermedia http://www.ht05.org/ (2005)

[IEEE 830] IEEE 830-1998 Recommended Practice for Software Requirements Specifications (1998)

[Jacobson 99] Ivar Jacobson, Grady Booch, James Rumbaugh. The Unified Software Development Process. Addison-Wesley (1999)

[Nanard 95] Jocelyne Nanard, Marc Nanard: Hypertext Design Environments and the Hypertext Design Process. Commun. ACM 38(8): 49-56 (1995)

[Navarro 02a] Antonio Navarro, Baltasar Fernández-Manjón, Alfredo Fernández-Valmayor, José Luis Sierra: Formal-Driven Conceptualization and Prototyping of Hypermedia Applications. Ralf-Detlef Kutsche, Herbert Weber (Eds.): Fundamental Approaches to Software Engineering, 5th International Conference, FASE 2002, held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002, Grenoble, France, April 8-12, 2002, Proceedings. Lecture Notes in Computer Science 2306 Springer 2002, ISBN 3-540-43353-8, pp 308-322

[Navarro 02b] Antonio Navarro. Conceptualización, Prototipado y Proceso de Aplicaciones Hipermedia. Tesis Doctoral. http://tesis.sim.ucm.es:2004/mat/ucm-t25910.pdf

[Navarro 04a] Antonio Navarro, Baltasar Fernández-Manjón, Alfredo Fernández-Valmayor, José Luis Sierra: The PlumbingXJ Approach for Fast Prototyping of Web Applications. Journal of Digital Information 5(2): (2004)

[Navarro 04b] Antonio Navarro, Alfredo Fernández-Valmayor, Baltasar Fernández-Manjón, José Luis Sierra: Conceptualization, Prototyping And Process Of Hypermedia Applications. International Journal of Software Engineering and Knowledge Engineering 14(6): 565-602 (2004)

[OMG 04] Object Management Group. Unified Modeling Language (UML), Version 1.5. http://www.omg.org/technology/documents/formal/uml.htm (2004)

[Sun 04] JavaTM 2 Platform Standard Edition 5.0 API Specification, http://java.sun.com/j2se/1.5.0/docs/api/ (2004)

[W3C 95] World Wide Web Consortium, Hypertext Terms, http://www.w3.org/Terms.html (1995)

Apéndice A. Especificación de requisitos software

A.1 Introducción

En este apéndice nos vamos a centrar en mostrar los requisitos que se concretaron con el profesor al iniciarse el proyecto, allá por el mes de octubre. Hay que dejar constancia que los requisitos no han cambiado en exceso, si bien tan solo se han incluido unas pequeñas ampliaciones a los iniciales, como que se pudieran dibujar notas y etiquetas, así como que los enlaces se pintarán hasta la superficie de las imágenes.

El desarrollo del proyecto básicamente ha consistido en desarrollar una herramienta case que cumpliera los requisitos analizados y concretados entre el profesor y los alumnos.

A.2 Descripción General

Como se ha mencionado ya a lo largo de toda la memoria, este proyecto consiste una herramienta case para el desarrollo de aplicaciones hipermedia.

Digamos que esta aplicación tiene tres puntos de gran interés, el primero es la notación gráfica, es decir los mecanismos que se le ofrece al usuario para poder diseñar desde un interfaz hasta el contenido de una aplicación hipermedia, además por supuesto de poderlos relacionar. El segundo punto de interés de la herramienta es que se permite guardar y cargar interfaces, contenidos y relaciones previamente diseñadas por dicho usuario. En este punto es importante indicar que el formato de los ficheros que se guardan es xml, lo que implica que no vale cualquiera ya que tiene que seguir una dtd definida. Además se ofrece la posibilidad de guardar en dos formatos cada apartado diseñado en la aplicación, uno es para guardarlo en formato que entiende el motor de prototipado, del cual hablaremos ahora y el otro simplemente para cargarlo en la herramienta. El tercer punto de interés de esta herramienta case es la capacidad de generar prototipos bien desde un fichero xml que se ha guardado previamente como hemos dicho antes o bien cogiendo los datos desde el diseño actual que contenga la aplicación. Esta generación de prototipos consiste únicamente en invocar a un motor de prototipos que ya estaba previamente implementado, el cual se encarga de generar físicamente el diseño que se ha estado haciendo con la herramienta case anteriormente.

A.4 Interfaces externos

La aplicación necesita la máquina virtual de Java para poder ejecutarse, así como la aplicación antes desarrollada y que nos servirá como motor para la generación de prototipos

A.4 Funciones

A.4.1 Módulo gráfico

A.4.1.1 Nivel interfaz

A.4.1.1.1 Dibujar nodo nexo

Descripción: Dibuja sobre el lienzo del interfaz un cuadrado, que es la representación gráfica del nodo nexo
Entrada: Dos puntos del lienzo del interfaz
Salida: Dibujo de un rectángulo
Origen: Lienzo del interfaz
Destino: Lienzo del interfaz
Requisito: La opción de dibujar nodo nexo debe de estar activada, y se debe cumplir que no este dentro de otro elemento del lienzo del interfaz.
Precondición:
Post condición: El nodo nexo queda dibujado
Efectos laterales:

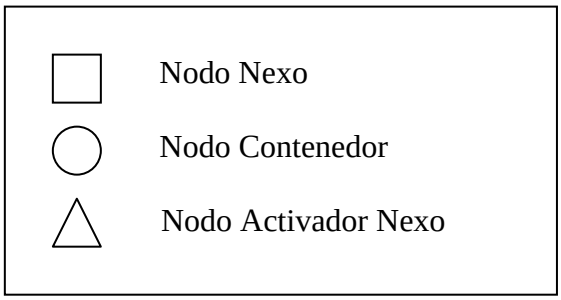
A.4.1.1.2 Dibujar nodo contenedor

Descripción: Dibuja sobre el lienzo del interfaz un círculo, que es la representación gráfica del nodo contenedor
Entrada: Dos puntos del lienzo del interfaz
Salida: Dibujo de un círculo
Origen: Lienzo del interfaz
Destino: Lienzo del interfaz
Requisito: La opción de dibujar nodo contenedor debe de estar activada, y se debe cumplir que no este dentro de otro elemento del lienzo del interfaz
Precondición:
Post condición: El nodo contenedor queda dibujado
Efectos laterales:

A.4.1.1.3 Dibujar nodo activador nexo

Descripción: Dibuja sobre el lienzo del interfaz un triángulo, que es la representación gráfica del nodo activador nexo
Entrada: Dos puntos del lienzo del interfaz
Salida: Dibujo de un triángulo
Origen: Lienzo del interfaz
Destino: Lienzo del interfaz
Requisito: La opción de dibujar nodo activador nexo debe de estar activada, y se debe cumplir que no este dentro de otro elemento del lienzo del interfaz
Precondición:
Post condición: El nodo activador nexo queda dibujado

Efectos laterales:



- Figura A.1. Representación visual de los nodos del modelo -

A.4.1.1.4 Dibujar conexión

Descripción: Dibuja sobre el lienzo del interfaz una recta, que es la representación gráfica de la conexión
Entrada: Dos nodos del lienzo del interfaz
Salida: Dibujo de una recta
Origen: Lienzo del interfaz
Destino: Lienzo del interfaz
Requisito: La opción de dibujar conexión debe de estar activada, además se debe cumplir que dicha conexión se realice entre un nodo nexo y un nodo contenedor o bien un nodo activador de nexo.
Precondición:
Post condición: La conexión queda dibujada. Se respetan las restricciones existentes entre elementos (ejemplo, no se pueden conectar dos nodos nexos con una relación de agregación)
Efectos laterales:

A.4.1.1.5 Dibujar enlace

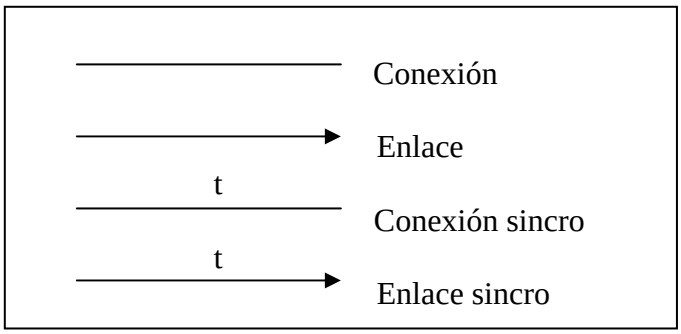
Descripción: Dibuja sobre el lienzo del interfaz una flecha, que es la representación gráfica del enlace.
Entrada: Dos nodos del lienzo del interfaz
Salida: Dibujo de una flecha
Origen: Lienzo del interfaz
Destino: Lienzo del interfaz
Requisito: La opción de dibujar enlace debe de estar activada, y además se debe cumplir la restricción de que una conexión solo se puede realizar entre dos nodos contenedores o bien un nodo nexo y un nodo activador de nexo.
Precondición:
Post condición: El enlace queda dibujado
Efectos laterales:

A.4.1.1.6 Dibujar conexión sincro

Descripción: dibuja sobre el lienzo del interfaz una recta con una marca de tiempo, que nos va a dar información sobre ese tiempo, que es la representación gráfica de la conexión sincro.
Entrada: Dos nodos del lienzo del interfaz
Salida: Dibujo de una recta con la marca de tiempo.
Origen: Lienzo del interfaz
Destino: Lienzo del interfaz
Requisito: La opción de dibujar conexión sincro debe de estar activada, y además se debe cumplir las restricciones de que dicha conexión sincro solo se puede llevar a cabo entre un nodo nexo y un nodo contenedor y un nodo nexo y un nodo activador de nexo.
Precondición:
Post condición: La conexión sincro queda dibujada
Efectos laterales:

A.4.1.1.7 Dibujar enlace sincro

Descripción: dibuja sobre el lienzo del interfaz una flecha con una marca “t”, que es la representación gráfica del enlace sincro, análoga a la anterior.
Entrada: Dos nodos del lienzo
Salida: Dibujo de una flecha con la marca “t” de tiempo
Origen: Lienzo del interfaz
Destino: Lienzo del interfaz
Requisito: La opción de dibujar enlace sincro debe de estar activada, y además se debe cumplir las restricciones de que dicha conexión sincro solo se puede llevar a cabo entre dos nodo nexo.
Precondición:
Post condición: El enlace sincro queda dibujado
Efectos laterales:



- Figura A.2. Representación visual de los arcos del interfaz –

A.4.1.2 Nivel contenidos

A.4.1.2.1 Dibujar contenido estático

Descripción: Dibuja sobre el lienzo del contenido un rectángulo de línea continua o un círculo relleno, que es la representación del contenido estático
Entrada: Dos puntos del lienzo del contenido
Salida: Dibujo de un rectángulo de línea continua o de un círculo relleno
Origen: Lienzo del contenido
Destino: Lienzo del contenido
Requisito: La opción de dibujar contenido estático debe de estar activada, y además se debe cumplir que no este dentro de otro elemento contenido.
Precondición:
Post condición: El contenido estático queda dibujado
Efectos laterales:

A.4.1.2.2 Dibujar ancla estática

Descripción: Dibuja sobre el lienzo del contenido un rectángulo relleno, que es la representación del ancla estática
Entrada: Dos puntos del lienzo del contenido
Salida: Dibujo de un rectángulo relleno
Origen: Lienzo del contenido
Destino: Lienzo del contenido
Requisito: La opción de dibujar ancla estática debe de estar activada, además se debe cumplir que no este dentro de ningún elemento del contenido que no sea un contenido estático (rectángulo) o un contenido dinámico (rectángulo).
Precondición:
Post condición: El ancla estática queda dibujada
Efectos laterales:

A.4.1.2.3 Dibujar enlace estático

Descripción: Dibuja sobre el lienzo del contenido una flecha continua de una sola punta, que es la representación del enlace estático
Entrada: Dos nodos del lienzo del contenido
Salida: Dibujo de una flecha continua de una sola punta
Origen: Lienzo del contenido
Destino: Lienzo del contenido
Requisito: La opción de dibujar enlace estático debe de estar activada, además se debe cumplir que el origen puede ser, un ancla estática, un contenido estático(círculo) o un contenido dinámico (círculo) y el destino tiene que ser un contenido estático (cualquiera de sus dos variantes).
Precondición: El destino tiene que ser un contenido estático

Post condición: El enlace estático queda dibujado
Efectos laterales:

A.4.1.2.4 Dibujar enlace estático n-ario

Descripción: Dibuja sobre el lienzo del contenido una flecha continua de n puntas, que es la representación del enlace estático n-ario
Entrada: n+1 nodos del lienzo del contenido
Salida: Dibujo de una flecha continua de n puntas
Origen: Lienzo del contenido
Destino: Lienzo del contenido
Requisito: La opción de dibujar enlace estático n-ario debe de estar activada, y se debe cumplir que el origen, igual que antes, debe ser un ancla estática, un contenido estático(círculo) o un contenido dinámico (círculo) y como destinos contenidos estáticos (en ambas variantes).
Precondición:
Post condición: El enlace estático n-ario queda dibujado
Efectos laterales:

A.4.1.2.5 Dibujar contenido dinámico

Descripción: Dibuja sobre el lienzo del contenido un rectángulo de línea discontinua o un círculo sin relleno, que es la representación del contenido dinámico
Entrada: Dos puntos del lienzo del contenido
Salida: Dibujo de una rectángulo de línea discontinua o de un círculo sin relleno
Origen: Lienzo del contenido
Destino: Lienzo del contenido
Requisito: La opción de dibujar contenido dinámico debe de estar activada , y además se debe cumplir que no este dentro de otro elemento contenido.
Precondición:
Post condición: El contenido dinámico queda dibujado
Efectos laterales:

A.4.1.2.6 Dibujar ancla dinámica

Descripción: Dibuja sobre el lienzo del contenido un rectángulo sin relleno, que es la representación del ancla dinámica
Entrada: Dos puntos del lienzo del contenido
Salida: Dibujo de una triángulo sin relleno
Origen: Lienzo del contenido
Destino: Lienzo del contenido
Requisito: La opción de dibujar ancla dinámica debe de estar activada. Sólo puede dibujarse sobre nodos de contenidos, ya sean estáticos o dinámicos
Precondición:

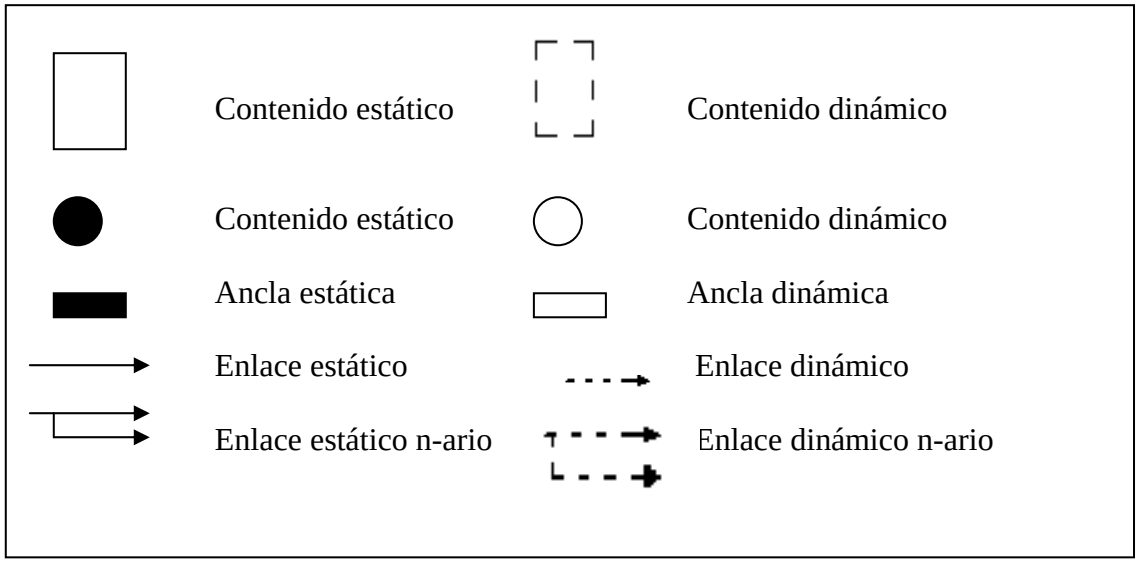
Post condición: El ancla dinámica queda dibujada
***Efectos laterales**:*

A.4.1.2.7 Dibujar enlace dinámico

Descripción: Dibuja sobre el lienzo del contenido una flecha discontinua de una sola punta, que es la representación del enlace estático
Entrada: Dos nodos del lienzo del contenido
Salida: Dibujo de una flecha discontinua de una sola punta
Origen: Lienzo del contenido
Destino: Lienzo del contenido
Requisito: La opción de dibujar enlace dinámico debe de estar activada, y además se debe cumplir que el origen puede ser, un ancla dinámica, un contenido estático(círculo) o un contenido dinámico (círculo) y el destino tiene que ser un contenido dinámico (cualquiera de sus dos variantes).
Precondición: El destino debe de ser un contenido dinámico
Post condición: El enlace dinámico queda dibujado
Efectos laterales:

A.4.1.2.8 Dibujar enlace dinámico n-ario

Descripción: Dibuja sobre el lienzo del contenido una flecha discontinua de n puntas, que es la representación del enlace dinámico n-ario
Entrada: n+1 nodos del lienzo del contenido
Salida: Dibujo de una flecha discontinua de n puntas
Origen: Lienzo del contenido
Destino: Lienzo del contenido
Requisito: La opción de dibujar enlace dinámico n-ario debe de estar activada, y se debe cumplir que el origen, igual que antes, debe ser un ancla dinámica, un contenido estático(círculo) o un contenido dinámico (círculo) y como destinos contenidos dinámicos (en ambas variantes).
Precondición: Los n-destinos tienen que ser todos contenidos dinámicos
Post condición: El enlace dinámico n-ario queda dibujado
Efectos laterales:



- Figura A.3 Representación gráfica de contenidos y enlaces -

A.4.1.3 Nivel mixto:

A.4.1.3.1 Establecer relaciones

Descripción: Relaciona los elementos del contenido con los de la interfaz
Entrada: Selección de elementos y un color
Salida: Relaciones
Origen: Lienzo del contenido y del interfaz
Destino: Lienzo del contenido y del interfaz
Requisito: Los únicos elementos de interfaz que se pueden relacionar con algunos de contenido son nodos contenedores y relaciones (enlace, enlace sincro, conexión y conexión sincro). Entonces un nodo contenedor se puede relacionar con cualquier nodo de contenido. De igual modo cada relación de interfaz se puede relacionar con cualquier relación de contenido (enlace estático, estático n-ario, dinámico, dinámico n-ario).

NOTA: un contenedor de interfaz puede tener una relación, peor ésta es de N elementos de contenido. Lógicamente dos elementos de interfaz o dos elementos de contenido no se pueden relacionar.
Precondición:
Post condición: El color de los contenidos seleccionados será el mismo y distinto a los ya existentes
Efectos laterales:

A.4.1.4 Modificar elementos

Descripción: Nos permite borrar, girar, modificar propiedades de un elemento de los lienzos
Entrada: Elemento
Salida: Elemento modificado
Origen: Lienzos
Destino: Lienzos
Requisito:
Precondición: El elemento tiene que estar seleccionado
Post condición: El elemento queda modificado conforme a lo que desea el usuario
Efectos laterales: Puede haber cambios en el contenido de los lienzos

A.4.2 Módulo persistencia

A.4.2.1 Cargar diseño

Descripción: Carga un diseño a partir de un archivo XML
Entrada: Archivo XML
Salida: Ambos lienzos quedan rellenos
Origen: Sistemas de archivos
Destino: Lienzo del contenido y del interfaz con sus relaciones
Requisito:
Precondición: El archivo debe de existir y ser válido según la dtd definida
Post condición: El diseño queda establecido
Efectos laterales:

A.4.2.2 Cargar contenido

Descripción: Carga un contenido a partir de un archivo XML
Entrada: Archivo XML
Salida: Prototipo
Origen: Sistemas de archivos
Destino: Módulo dibujo
Requisito:
Precondición: El archivo debe de existir y ser válido según la dtd definida
Post condición: El contenido queda establecido
Efectos laterales:

A.4.2.3 Cargar relaciones

Descripción: Carga un contenido a partir de un archivo XML
Entrada: Archivo XML
Salida: Prototipo
Origen: Sistemas de archivos
Destino: Módulo dibujo
Requisito:
Precondición: El archivo debe de existir y ser válido según la dtd definida
Post condición: Las relaciones entre diseño y contenido quedan establecidas
Efectos laterales:

A.4.2.4 Salvar diseño

Descripción: Salva un diseño a un archivo XML
Entrada: lienzo diseño
Salida: Archivo XML
Origen: Interfaz de usuario
Destino: Sistema de archivos
Requisito:
Precondición:
Post condición: El diseño queda guardado en el archivo seleccionado y si no existe se crea
Efectos laterales: Creación o modificación de archivo

A.4.2.5 Salvar contenido

Descripción: Salva un contenido a un archivo XML
Entrada: lienzo contenido
Salida: Archivo XML
Origen: Interfaz de usuario
Destino: Sistema de archivos
Requisito:
Precondición:
Post condición: El contenido queda guardado en el archivo

A.4.2.6 Salvar relación

Descripción: Salva relación entre diseño y contenido a un archivo XML
Entrada: ambos lienzos
Salida: Archivo XML
Origen: Interfaz de usuario
Destino: Sistema de archivos
Requisito:
Precondición:
Post condición: La relación queda guardada en el archivo

A.4.2.7 Salvar prototipo

Descripción: Salva un prototipo a un archivo XML
Entrada: Prototipo
Salida: Archivo XML
Origen: Módulo prototipado
Destino: Sistema de archivos
Requisito:
Precondición:
Post condición: El prototipo queda guardado en el archivo seleccionado y si no existe se crea
Efectos laterales: Creación o modificación de archivo

A.4.3 Módulo prototipado

Este módulo fue desarrollado antes de la realización de nuestra herramienta case, el objetivo final de este desarrollo es la integración de la herramienta con dicho motor de prototipazo, para como su propio nombre indica la generación de prototipos a partir de un diseño y viceversa.

A.4.1 Generar prototipo

Descripción: Crea un prototipo a partir de la notación gráfica
Entrada: Notación visual
Salida: Prototipo
Origen: Interfaz de usuario
Destino: Módulo prototipado
Requisito:
Precondición:
Post condición: El prototipo es creado
Efectos laterales: Creación de prototipo

A.4.2 Generar diseño

Descripción: Crea un diseño a partir de un prototipo
Entrada: prototipo
Salida: Notación visual
Origen: Módulo prototipado
Destino: Interfaz de usuario
Requisito:
Precondición:
Post condición: El diseño es creado
Efectos laterales: Creación de diseño

A.5 Requisitos de rendimiento

No precisa de requisitos de rendimientos establecidos, la aplicación debe ofrecer una eficiencia que haga razonable el uso de la aplicación por parte del usuario.

Apéndice B. Documentos XML

B.1 Introducción

Utilizamos la tecnología XML para la generación de prototipos, ya que el motor que se encara de ellos partiendo de dos documentos de este tipo es capaz de generar un prototipo concreto.

Para llevar a cabo esta funcionalidad necesitamos crear seis documentos xml. Primero de todo hay que generar interfaz, contenido y canalización, esto de lo que se trata es de generar una serie de XML intermedios para posteriormente a la hora de generar el prototipo se obtengan dos nuevos XML a partir de los anteriores que son lo que utiliza el motor prototipado para, como ya se ha mencionado antes, generar un prototipo. Por último decir que el motor prototipado necesita también un archivo XSLT para transformar los documentos XML que recibe y generar el prototipo real.

B.2 DTD’s y XSLT

B.2.1 DTD’s alternativas

En este punto mostramos las DTD’s de los ficheros XML intermedios que genera para el interfaz, contenido y canalización (relación), necesarios para posteriormente generar los prototipos.

B.2.1.1 DTD contenido

Esta dtd sirve para validar el documento XML intermedio que se obtiene de generar o guardar “formalmente” el diseño del contenido, es decir que volcamos a un documento XML, con ciertas características para después pasar a los que de verdad entiende el motor, todos los elementos que contiene el contenido
.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT contenidos (elemento+)>
<!ELEMENT elemento (#PCDATA | ancla | anclaN)*>
<!ATTLIST elemento
        id ID #REQUIRED
        tipo (est | din) "est"
        forma (circ | rect) "rect">
<!ELEMENT ancla (#PCDATA)>
<!ATTLIST ancla
        id ID #IMPLIED
        href CDATA #REQUIRED>
<!ELEMENT anclaN (texto, destinoN, destinoN+)>
<!ATTLIST anclaN
        id ID #IMPLIED>
<!ELEMENT texto (#PCDATA)>
<!ELEMENT destinoN EMPTY>
<!ATTLIST destinoN
        href CDATA #REQUIRED>
```

- Figura B.1. DTD de contenido (contenidos.dtd)-

B.2.1.2 DTD interfaz

Esta dtd sirve para validar el documento XML intermedio que se obtiene de generar o guardar “formalmente” el diseño de la interfaz, es decir que volcamos a un documento XML, con ciertas características para después pasar a los que de verdad entiende el motor, todos los elementos que contiene el interfaz.

```
<!ELEMENT aplicacion (nombre?, ventana+)>
<!ATTLIST aplicacion id ID #REQUIRED>

<!ELEMENT nombre (#PCDATA)>
<!ELEMENT ventana (nombre?, panel*, activadorV*)>
<!ATTLIST ventana id ID #REQUIRED>

<!ELEMENT panel (contenidoPanel, destinoEnlaces?)>
<!ATTLIST panel id ID #REQUIRED
                tam CDATA "300, 200">

<!ELEMENT contenidoPanel (contenidoDefecto?, contenido*, contenidoGrupo*)>
<!ELEMENT contenidoDefecto (#PCDATA)>
<!ATTLIST contenidoDefecto tipo (XPath|proceso) "XPath">
<!ELEMENT contenido (#PCDATA)>
<!ATTLIST contenido tipo (XPath|proceso) "XPath">
<!ELEMENT contenidoGrupo (#PCDATA)>

<!ELEMENT destinoEnlaces EMPTY>
<!ATTLIST destinoEnlaces panel IDREF #REQUIRED>

<!ELEMENT activadorV (nombre)>
<!ATTLIST activadorV id ID #REQUIRED
          ventana IDREF #REQUIRED>
```

- Figura B.2 DTD del interfaz (aplicación.dtd) -

B.2.1.3 DTD canalización

Esta dtd sirve para validar el documento XML intermedio que se obtiene de generar o guardar “formalmente” las relaciones entre interfaz y contenido, es decir que volcamos a un documento XML, con ciertas características para después pasar a los que de verdad entiende el motor, todas las relaciones establecidas.

```
<!ELEMENT canalizacion (panel+)>

<!ELEMENT panel (contenidoPanel, destinoEnlaces?)>
<!ATTLIST panel id ID #REQUIRED>

<!ELEMENT contenidoPanel (contenidoDefecto?, contenido*, contenidoGrupo*)>
<!ELEMENT contenidoDefecto (#PCDATA)>
<!ATTLIST contenidoDefecto
        tipo (XPath | proceso) "XPath">
<!ELEMENT contenido (#PCDATA)>
<!ATTLIST contenido
        tipo (XPath | proceso) "XPath">
<!ELEMENT contenidoGrupo (#PCDATA)>
<!ELEMENT destinoEnlaces (especifico*)>
<!ATTLIST destinoEnlaces
        panel IDREF #REQUIRED>

<!ELEMENT especifico EMPTY>
<!ATTLIST especifico ancla CDATA #REQUIRED
       panel IDREF #REQUIRED>
```

- Figura B.3 DTD canalización o relación (canalizacion.dtd) -

B.2.2 DTD's y XSLT del motor

En este apartado nos vamos a centrar en los archivo XML que entiende el motor de prototipo y a partir de los cuales y aplicando una transformación XSLT es capaz de generar los prototipos.

B.2.2.1 DTD aplicación

Esta dtd sirve para validar el documento XML que entiende el motor de prototipado que se obtiene a partir del XML generado anteriormente para la interfaz.

```
<!ELEMENT aplicacion (nombre?, ventana+)>
<!ATTLIST aplicacion id ID #REQUIRED>

<!ELEMENT nombre (#PCDATA)>
<!ELEMENT ventana (nombre?, panel*, activadorV*)>
<!ATTLIST ventana id ID #REQUIRED>

<!ELEMENT panel (contenidoPanel, destinoEnlaces?)>
<!ATTLIST panel id ID #REQUIRED
                tam CDATA "300, 200">

<!ELEMENT contenidoPanel (contenidoDefecto?, contenido*, contenidoGrupo*)>
<!ELEMENT contenidoDefecto (#PCDATA)>
<!ATTLIST contenidoDefecto tipo (XPath|proceso) "XPath">
<!ELEMENT contenido (#PCDATA)>
<!ATTLIST contenido tipo (XPath|proceso) "XPath">
<!ELEMENT contenidoGrupo (#PCDATA)>

<!ELEMENT destinoEnlaces EMPTY>
<!ATTLIST destinoEnlaces panel IDREF #REQUIRED>

<!ELEMENT activadorV (nombre)>
<!ATTLIST activadorV id ID #REQUIRED
          ventana IDREF #REQUIRED>
```

- Figura B.4 DTD de aplicación para el motor (aplicacion.dtd) -

B.2.2.2 DTD contenido

Esta dtd sirve para validar el documento XML que entiende el motor de prototipado que se obtiene a partir del XML generado anteriormente para el contenido.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT contenidos (elemento+)>
<!ELEMENT elemento (#PCDATA | a)*>
<!ATTLIST elemento id ID #REQUIRED>
<!ELEMENT a (#PCDATA)>
<!ATTLIST a href CDATA #REQUIRED>
```

- Figura B.5 DTD para el contenido que entiende el motor (contenidosSI.dtd) -

B.2.2.3 XSLT del motor

Esta es la transformación que el documento aplica a los XML que recibe y que utiliza para generar el prototipo final.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">

<xsl:output method="html"
        encoding = "ISO-8859-1"
        omit-xml-declaration="yes"
        standalone="yes"
                doctype-public=""
                doctype-system=""
                cdata-section-elements=""
                indent="no"
                media-type=""/>


<xsl:template match="/">
<html>
  <body>
    <pre style="font-family:'times new roman"'>
      <xsl:apply-templates/>
    </pre>
  </body>
</html>
</xsl:template>

<xsl:template match="*[@href]">
  <a href="{@href}.html" style= "font-style:italic; color:black;
text-decoration:none" >
    <xsl:apply-templates/>
  </a>
</xsl:template>

<xsl:template match="*[@name]">
  <a name="{@name}">
   <xsl:apply-templates/>
  </a>
</xsl:template>
```

```
<xsl:template match="*[@imagen]">
  <xsl:choose>
    <xsl:when test= "./*[@coords]">
    <img src= "file:{@imagen}" usemap= "#{@imagen}"></img>
      <map name="{@imagen}">
        <xsl:apply-templates/>
      </map>
    </xsl:when>
    <xsl:otherwise>
     <img src= "file:{@imagen}"></img>
      <xsl:apply-templates/>
     </xsl:otherwise>
  </xsl:choose>
 </xsl:template>

<xsl:template match="*[@href][@coords]">
  <area shape= "poly" coords= "{@coords}" href="{@href}.html" />
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="*[@hrefP]">
 <a href="{@hrefP}" style= "font-style:italic; color:black;
text-decoration:none">
   <xsl:apply-templates/>
 </a>
</xsl:template>

<xsl:template match="*[@hrefP][@coords]">
  <area shape= "poly" coords= "{@coords}" href="{@hrefP}" />
  <xsl:apply-templates/>
</xsl:template>


<xsl:template match="*[@formulario]">
<form action="función g">
   <xsl:apply-templates/>
</form>
</xsl:template>

<xsl:template match="*[@formInput]">
<input type="{@formTipo}" name="{@formNombre}" size="{@formTam}"
value="{@formValor}">
<xsl:apply-templates/>
</input>
</xsl:template>

</xsl:stylesheet>
```

- Figura B.6 XSLT que utiliza el motor (trans.xsl) -

B.3 Ejemplo de uso

Estos son los ficheros generados por una ejecución de la aplicación.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE contenidos SYSTEM "contenidos.dtd">

<contenidos>
  <elemento id="Contenido_0" forma="circ" tipo="est">
    Este es el contenido 0
    <ancla id="a1_Contenido_0" href="Contenido_1">Ancla 1</ancla>
  </elemento>
  <elemento id="Contenido_1" forma="circ" tipo="est">
    Este es el contenido 1
    <ancla id="a1_Contenido_1" href="Contenido_0">ancla siguiente</ancla>
  </elemento>
  <elemento id="Contenido_2" forma="circ" tipo="est">
    este es el contenido 2
    <ancla id="a1_Contenido_2" href="Contenido_0">ancla ultima</ancla>
  </elemento>
</contenidos>
```

- Figura B.7. Documento XML intermedio para el contenido diseñado -

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE interfaz SYSTEM "interfaz.dtd">

<interfaz id="ejemplo">
  <ventana id="v1">
    <nombre>v1</nombre>
    <panel id="p1.1" tam="300,300">
      <tuberia destino="p2.1" />
    </panel>
    <activadorV id="b1" ventana="v2">
      <nombre>v2</nombre>
    </activadorV>
  </ventana>
  <ventana id="v2">
    <nombre>v2</nombre>
    <panel id="p2.1" tam="300,300">
      <tuberia destino="p1.1" />
    </panel>
    <panel id="p2.2" tam="300,300">
      <tuberia destino="p1.1" />
    </panel>
  </ventana>
</interfaz>
```

- Figura B.8 Documento XML intermedio para el interfaz diseñado -

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE canalizacion SYSTEM "canalizacion.dtd">

<canalizacion>
  <panel id="p2.1">
    <contenidoPanel>
      <contenido
tipo="XPath">/contenidos/elemento[@id="Contenido_1"]</contenido>
    </contenidoPanel>
    <destinoEnlaces panel="p1.1" />
  </panel>
  <panel id="p1.1">
    <contenidoPanel>
      <contenidoDefecto
tipo="XPath">/contenidos/elemento[@id="Contenido_0"]</contenidoDefecto>
          <contenido
tipo="XPath">/contenidos/elemento[@id="Contenido_0"]</contenido>
    </contenidoPanel>
    <destinoEnlaces panel="p2.1" />
  </panel>
  <panel id="p2.2">
    <contenidoPanel>
      <contenidoDefecto
tipo="XPath">/contenidos/elemento[@id="Contenido_2"]</contenidoDefecto>
          <contenido
tipo="XPath">/contenidos/elemento[@id="Contenido_2"]</contenido>
    </contenidoPanel>
    <destinoEnlaces panel="p1.1" />
  </panel>
</canalizacion>
```

- Figura B.9 Documento XML intermedio para la canalización diseñada -

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE contenidos SYSTEM "contenidosSI.dtd">
<contenidos>
  <elemento id="Contenido_0">
    Este es el contenido 0
    <a href="Contenido_1">Ancla 1</a>
  </elemento>
  <elemento id="Contenido_1">
    Este es el contenido 1
    <a href="Contenido_0">ancla siguiente</a>
  </elemento>
  <elemento id="Contenido_2">
    este es el contenido 2
    <a href="Contenido_0">ancla ultima</a>
  </elemento>
</contenidos>
```

- Figura B.10 Documento XML del contenido que utiliza el motor -

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE contenidos SYSTEM "contenidosSI.dtd">

<contenidos>
  <elemento id="Contenido_0">
    Este es el contenido 0
    <a href="Contenido_1">Ancla 1</a>
  </elemento>
  <elemento id="Contenido_1">
    Este es el contenido 1
    <a href="Contenido_0">ancla siguiente</a>
  </elemento>
  <elemento id="Contenido_2">
    este es el contenido 2
    <a href="Contenido_0">ancla ultima</a>
  </elemento>
</contenidos>
```

- Figura B.11 Documento XML de canalizaciones que utiliza el motor -

Se autoriza a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Alejandro Blanco Martín	David Curieses Chamón	Óscar Ortega Mejías