Universidad Carlos III de Madrid

Escuela Politécnica Superior

Ingeniería Técnica en Informática de Gestión

Proyecto fin de carrera

SISTEMA GRÁFICO DE VISUALIZACIÓN

DEL MOTOR DE REGLAS CLIPS

DE UN SIMULADOR EMPRESARIAL

Autor: Javier Muelas Serrano

Tutor: Fernando Fernández Rebollo

A mi madre y hermanos,
y a la memoria de mi padre

Agradecimientos

Con la finalización del proyecto fin de carrera se cierra un ciclo más de la vida del que han tomado parte muchas personas. Mi entorno durante estos años se pueden dividir en 5 grandes grupos, mis *amigos del barrio*, *mi familia* y *amigos del pueblo*, mis *amigos de la* universidad, mi novia y mi *familia*.

Antes de nada quiero agradecer este proyecto a mi tutor, *Fernando Fernández*, por su paciencia conmigo durante el desarrollo del mismo, estando siempre disponible ante cualquier duda o problema que han ido apareciendo, y marcándome la senda a seguir para la correcta evolución de este trabajo.

De mis *amigos del barrio* quiero acordarme en primer lugar de todos aquellos que forman parte de "*Los Sotillo*". Más que un equipo de fútbol es una religión, es la representación de la amistad desde la infancia y la adolescencia, y que todavía goza, y espero que siga gozando, de una excelente salud.

En el otro lado del barrio, con la carrera ya empezada, aparecieron en mi vida a través de mi hermana, los que siento como si fueran amigos de siempre. Todos los pertenecientes y allegados a esa gran asociación juvenil llamada "*Junior Yedra*" donde encontré grandes amigos, y donde se encendió en mí la llama del *voluntariado,* que tanto disfruto realizando.

Rebuscando en las raíces de mis padres, que son las mías también, allá en un pueblecito de la alcarria conquense llamado *Villaconejos de Trabaque*, tengo una gran familia y algunos de mis mejores amigos, que desde que era pequeño me han acompañado en mi camino y que siempre me han apoyado a pesar de la distancia. Han pasado los años y me sigue emocionando pasar tiempo con vosotros cada vez que la agenda me lo permite.

Ya en la universidad, han sido muchos los compañeros con los que he compartido horas de clase, prácticas y exámenes. Con algunos de ellos se fraguó una amistad más allá del campus. Esta vez diré algunos de los nombres que marcan mi paso por la universidad.

Primero mencionar a aquellos con los que empecé este viaje en mi primer año, *Carlos, Rocío y Jorge*. Fuimos muchos los que empezamos, pero solo unos pocos elegidos los que llegamos ☺

De mi segundo año me quiero acordar de gente como el señor *"Crusta", Berna, Gabi y Quique* (vital en el último momento, enorme por dentro y por fuera) que me acompañaron durante toda la carrera, a veces en las aulas, y en otras ocasiones en el campo de fútbol("Pollastre" → Campeones de la Copa Rector de Leganés en 2008, y "Los Benjamines de Niño Mora").

Luego más adelante, aunque hubo algunos otros, se produjo un caso excepcional, en el que la casualidad o la causalidad quisieron que conociera a *Sergio Coronado*, un grande entre los grandes que, cosas de la vida, ha pasado a ser uno de mis mejores amigos del barrio dentro del sanedrín "sotillista".

Me gustaría mencionar especialmente también a un amigo dentro y fuera de la universidad, con el que ya llevo muchos años, y con el que he echado muchas horas de biblioteca para sacar nuestras ingenierías adelante (te debo una cena) y he tenido grandes conversaciones en nuestras idas y venidas a la universidad. El gran *Pablo Díaz*. Gracias por "darme la vara" (en el buen sentido) para que terminara este proyecto.

Ya en la parte más personal, como no puede ser de otra forma, quiero dedicar el final de este ciclo a mi novia *Lidia*. Apareciste un 3 de febrero de 2008, y desde entonces he contado con todo tu apoyo para sacar esto adelante, haciendo que los momentos difíciles hayan sido más llevaderos, y endulzando cada paso para llegar finalmente a la meta. Gracias.

Como colofón a estos agradecimientos, y como lo mejor se deja para el final, quiero dedicar la finalización de mis estudios, con letras mayúsculas y de forma muy especial, a mi *familia*.

En primer lugar dar las gracias a mi *madre*, doña *Pilar Serrano*, por darme una educación sublime, que me hace ser quien soy en la actualidad. Por todo lo que has luchado conmigo para que no deje mis proyectos a medias, desde el colegio hasta la empresa, como en cualquier faceta de la vida, siempre animándome y convenciéndome

de que puedo con todo, y no hay limites para alcanzar todos los objetivos que me proponga. Tú misma has sido siempre un ejemplo de ello sacándonos a mis hermanos y a mí adelante cuando las cosas se pusieron difíciles. Mejor ejemplo no se puede tener. Es un privilegio tenerte como madre.

De mis hermanos no puedo decir menos. Mi hermano *Juan* que desde que era bien pequeño, hasta en la misma universidad, siempre se ha esmerado en ayudarme a aprender la lección, dedicándome todo el tiempo que he necesitado hasta que comprendía las cosas. Mis logros académicos nunca se hubieran materializado sin ti. Luego en lo personal has sido siempre una referencia para mi, has hecho de padre, hermano y amigo, y eso tiene un valor incalculable.

Con mi hermana *Belén* no tuve que luchar en lo académico, por eso de que soy dos años mayor que ella, pero esa misma cercanía en edad es la que nos ha dado, desde que éramos pequeños, una amistad basada en una gran confianza y complicidad entre los dos. Con el paso de los años te has convertido en una fuente de inspiración para mí. Tu filosofía de la vida me ha ayudado, y me sigue ayudando, para afrontar muchos momentos de oscuridad en mi mente, que con tu saber estar y optimismo me has abierto muchos de los caminos que estaban cerrados por mi propia “cabezonería”.

Los 4 siempre hemos sido una piña, compartiendo tantas cosas durante toda la vida que no hay palabras suficientes para transmitir todo lo que significáis para mi. Además de ser *madre, hijos y hermanos* somos *íntimos amigos*, y ahí reside parte de nuestro secreto. Os quiero mucho.

En definitiva, muchas gracias a todos. Hay mucha gente que no he mencionado y que he intentado englobar dentro de los grandes grupos, donde espero os veáis reflejados. Vosotros sabéis quienes sois las personas que habéis sido participes, de una u otra forma, de la consecución de mi recorrido académico.

Esto no es más que el cierre de un ciclo que será continuado por muchos más. *Aparecerán nuevos objetivos que me ilusionen haciendo volar mis sueños, que procuraré poner en la tierra, para así finalmente hacerlos realidad.*

Índice general

Capítulo 1.

Introducción

1- Introducción

Desde siempre, para cualquier cosa en la vida, es necesaria la existencia de una "explicación", para que así pueda llegar a comprenderse de forma completa. Si además a la explicación la dotas de elementos gráficos, dibujos o imágenes, relacionados con lo que se explica, permite que el receptor tenga una mejor y más fácil comprensión.

En informática pasa un poco igual. Desde los desarrollos más pequeños, hasta extensos programas de gran complejidad, contar con una representación gráfica de la estructura de un programa a nivel de implementación, puede agilizar en gran medida el análisis de un caso concreto.

A partir de estos fundamentos básicos, se encuentra el origen del proyecto fin de carrera que se pasa a describir en los próximos apartados.

1.1- Motivación

Partiendo del proyecto de investigación acerca de un *simulador empresarial* de la empresa "Simuladores Empresariales S. L", sobre el que *Daniel Sánchez Cisneros*, alumno de la carrera de *Ingeniería técnica en Informática de Gestión en la Universidad Carlos III de Madrid*, realizó un *motor de reglas CLIPS* como proyecto fin de carrera, surgió la necesidad del desarrollo de una interfaz gráfica con la que poder interpretarlo.

El simulador está configurado como un juego en el que se tienen una serie de empresas dentro de un mercado, y en donde cada empresa es controlada por un *empresario* o *jugador* que puede tomar decisiones como subir precios, contratar más personal u otras operaciones, que influyen directamente en la posición de la empresa en el mercado.

El fundamento final es que las empresas compitan por si solas dentro del simulador, en función de las características iniciales que se le da a cada una de ellas.

Para desarrollar este sistema inteligente se eligió el lenguaje CLIPS, creando un sistema de *reglas* o *fórmulas* integradas dentro de una serie de *módulos*, que se relacionan entre sí mediante distintos *atributos de entrada y de salida,* que van generando los resultados asociados a estas *reglas* y que, en algunos casos, son componentes de *cálculos* superiores.

Ésta es la descripción superficial de cómo se estructura el motor de simulación, y que va a ser la información que se quiere tratar. Por lo tanto, éste proyecto fin de carrera comienza con el objetivo de que esas partes que fueron desarrolladas en CLIPS, sean interpretables de forma gráfica, y para ello es necesario tomar algunas decisiones que se describirán más adelante.

1.2- Objetivos

El objetivo principal de este proyecto es la obtención de una representación gráfica del sistema de *módulos*, *cálculos*, *reglas* y *atributos* que componen el programa en CLIPS, de forma que se pueda entender a través de *grafos* la relación entre todas las partes que conforman el motor de simulación.

Para ello hay que realizar un análisis profundo, en el que se estudien bien las partes en las que se divide la implementación en CLIPS, diferenciando que características tienen las *reglas*, para poder extraer de ellas la información que interesa desglosar, y luego así catalogarla en una serie de *módulos*.

El formato escogido donde plasmar toda esta información desglosada, es en forma de *grafo*, que se entiende como una "*serie de nodos relacionados entre si*". Llevado a este proyecto, los nodos serían la representación de las *reglas* y *atributos*, que estarían unidos mediante flechas en función de la influencia de unos sobre otros.

A continuación se muestra un planteamiento de forma gráfica, de las partes del desarrollo en CLIPS sobre el que está desarrollado el motor de simulación.

En la Figura 1.2.1 se puede apreciar una vista de cómo sería la relación entre varios módulos. Cada módulo contendría en su interior *reglas* y *cálculos* de unas características determinadas, que les hacen ser de ese *módulo*. Los *atributos* pueden ser de *entrada* y *salida* dependiendo de la *regla* que sea. Aunque las *reglas* sean de distintos *módulos*, los *atributos* pueden relacionarse con cualquiera de ellas, que produce la relación entre *módulos*:

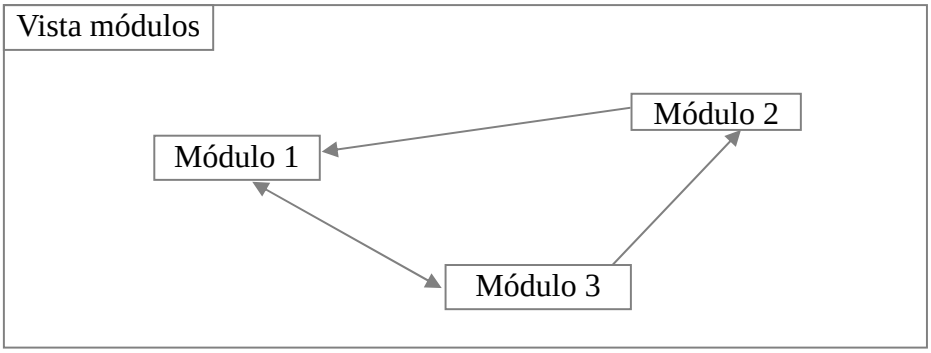


Figura 1.2.1

En la *Figura 1.2.2* se tiene una vista del interior de un *módulo*, en el que la *Regla 1* contiene el *Atributo 1*, que se sabe que es un *atributo de entrada* de ésta, por la flecha que le sale y conecta con la *regla* que le contiene. El *Atributo 2* se representa como *atributo de salida* de la *Regla 1,* mediante la flecha que sale y conecta con el *Atributo 2*. A su vez, la *Regla 2* tiene como *atributo de entrada* a *Atributo 2*, y como *atributo de salida* a *Atributo 1*:

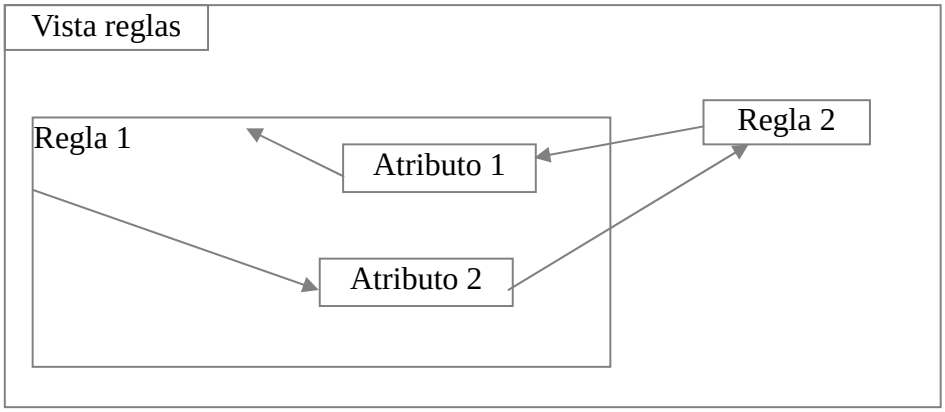


Figura 1.2.2

Ahora, juntando los conceptos de la *Figura 1.2.1* y *Figura 1.2.2*, con ayuda de la *Figura 1.2.3* y la *Figura 1.2.4,* se ofrece un ejemplo donde se muestra todo junto.

En la *Figura 1.2.3* se tiene de nuevo una vista con módulos relacionados. En esta ocasión, se puede interpretar que *Módulo 1* tiene dos *atributos* que son de *entrada* en *Módulo 2,* y uno que es de *entrada* en *Módulo 3*. Luego, *Módulo 2* tiene un *atributo* que es de *entrada* en *Módulo 3*:

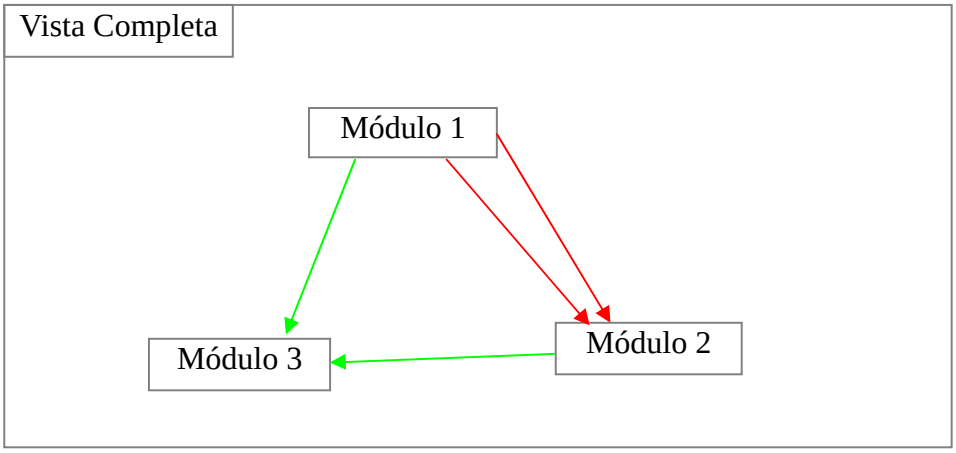


Figura 1.2.3

En la *Figura 1.2.4* se tiene la misma vista que en la *Figura 1.2.3,* solo que con los módulos desplegados. De esta forma, se puede ver con más claridad de donde vienen las flechas que unen los *módulos*. Además, la relación de colores facilita la comprensión de la pertenencia entre *reglas* y *atributos*.

En *Módulo 1 está Regla 1,* que tiene como *entrada* a *Atributo 1* y como *salida* a *Atributo 2*. A su vez, tanto *Atributo 1* como *Atributo 2* son *entrada* de *Regla 2*, que pertenece a *Modulo 2,* y que tiene como *salida* a *Atributo 3,* que junto con *Atributo 1* son entrada de *Regla 3*. Ésta última pertenece a *Módulo 3* y tiene como *salidas* a *Atributo 4* y a *Atributo 5*:

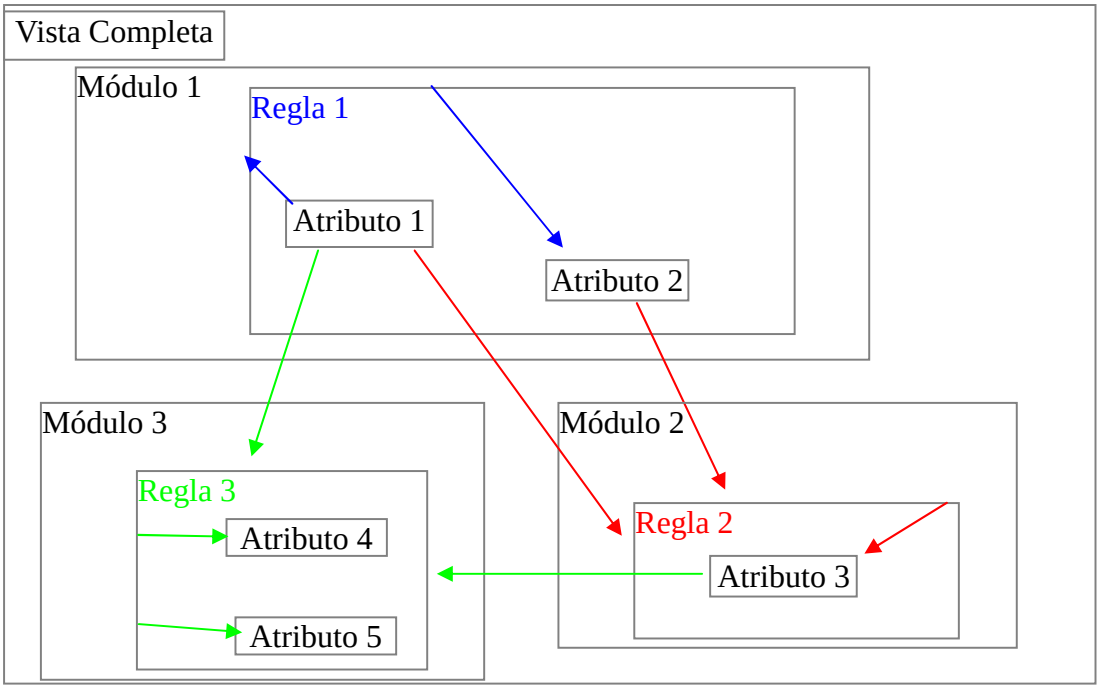


Figura 1.2.4

Las descripciones realizadas, con la ayuda de los ejemplos de la *Figura 1.2.1* a la *Figura 1.2.4*, son una muestra de la estructura que se quiere conseguir en este proyecto.

1.3- Descripción memoria

A lo largo de la presente memoria se irán desglosando las diferentes partes, que de una forma u otra, han influido en el desarrollo del sistema gráfico. Por ello, a continuación se va a hacer un breve resumen de lo que se va a ver en los próximos apartados de este proyecto fin de carrera.

En primer lugar, se van a describir las tecnologías con las que se ha tenido que trabajar. En este caso han sido dos, el lenguaje CLIPS en el que está desarrollado el motor de reglas y que es la fuente de información de este proyecto, y el lenguaje JAVA, que es con el que se ha implementado el sistema gráfico.

Tras tecnologías, se va a describir todo el proceso de creación del sistema gráfico, que se divide en 3 apartados.

El primero de ellos será la fase de análisis, a la que pertenecen las tareas de recopilación de la información necesaria para el desarrollo del sistema gráfico, mediante el estudio del programa en CLIPS.

Después se pasará a la fase de diseño donde se explica todo lo referente a la elaboración de la "*forma*" que se quiere del sistema gráfico.

Y en tercer lugar dentro de la fase de creación se tendría la fase de evaluación, donde se realizan una serie de pruebas de *"buen funcionamiento*" de las partes críticas de la aplicación desarrollada.

Tras el apartado de pruebas, llegará la parte de conclusiones y trabajos futuros, donde se hace una valoración de todo el trabajo realizado, y de las posibilidades que se abren tras la consecución del mismo.

La parte final de la memoria está destinada para la documentación extra acerca del proyecto realizado, que en este caso serán los referentes a *ejemplos del simulador*, al *diseño detallado*, el *manual de usuario* y la *gestión de proyecto*.

Capítulo 2.

Estudio Gestión/Tecnología

2- Estudio Gestión/Tecnología

Tanto para el desarrollo del motor de simulación, como para del sistema gráfico que lo representa, se han contado con unas tecnologías determinadas.

En primer lugar, en el *apartado 2.1* se va a describir el lenguaje de programación CLIPS, en el que está desarrollado el motor de simulación.

En el *apartado 2.2*, se va a analizar cómo está estructurado el desarrollo del motor de simulación en CLIPS.

Y en el *apartado 2.3*, se describirá el lenguaje JAVA y cómo ha sido su utilización en la implementación del sistema gráfico, tomando la información del código en CLIPS del motor de simulación.

2.1- CLIPS

El lenguaje de programación CLIPS, desarrollado por la NASA desde 1986, es una herramienta para la elaboración de sistemas expertos y de inteligencia artificial.

Las siglas "*CLIPS*" son el acrónimo de "*C Language Integrated Production System*" (Sistema de Producción Integrado en Lenguaje C). Este sistema de producción además incluye:

- Un sistema de mantenimiento de verdad con "*encadenamientos hacia delante*", que se entiende como que los *hechos* de la "*base de hechos*" son considerados como disparadores de las *reglas*. La exploración de las *reglas* en las condiciones descritas, corresponde a un razonamiento "*desde los hechos hacia los objetivos*".

- Adición dinámica de *reglas* y *hechos*.

- Y diferentes estrategias de resolución de conflictos.

Para explicar el funcionamiento de un programa en CLIPS, es imprescindible saber que se tienen una serie de *hechos* que activan una serie de *reglas*, y que se ejecutan en un orden determinado.

Los *hechos* son la declaración de los datos de entrada. Para su creación se necesita definir una serie de *plantillas* con unos determinados *campos*, que luego se *instanciarán* con la creación de los *hechos*.

Se entiende mejor en el *Ejemplo 2.1.1,* donde se puede ver la definición de una *plantilla* y su posterior *instanciación* a través de los *hechos*:

Ejemplo 2.1.1

A continuación, se tiene una "plantilla" llamada "persona" que tiene 3 campos, "nombre" de tipo STRING (cadena de caracteres), "capital" de tipo INTEGER (numérico) y "PYME" de tipo STRING:

```
(deftemplate empresa
     (slot nombre (type STRING))
     (slot capital (type INTEGER))
     (slot PYME (type STRING))
)
```

Ahora se muestra una definición de "hechos" llamado "mercado", que crea instancias de la plantilla "empresa". Se definen 3 empresas con un determinado valor en sus capitales, y "siendo o no" una PYME (Pequeña y mediana empresa):

```
(deffacts mercado
     (empresa (nombre "Empresa1") (capital 10) (PYME "no"))
     (empresa (nombre "Empresa2") (capital 20) (PYME "no"))
     (empresa (nombre "Empresa3") (capital 30) (PYME "si")) )
```

Las *reglas* son expresiones para realizar acciones dentro de un programa CLIPS, siempre y cuando existan unos *hechos* iniciales con los que ejecutar un programa. Se dividen en 2 partes, la izquierda y la derecha.

La *parte izquierda de una regla*, son las condiciones necesarias para que una *regla* se active para un *hecho* determinado. En la condición puede ser que el *hecho* tenga un valor determinado, o un valor cualquiera en uno de sus campos. En el *Ejemplo 2.1.2* se tiene una muestra:

Ejemplo 2.1.2

La siguiente línea de código CLIPS, representa la parte izquierda de una regla. La condición exigida, es tener un "hecho" perteneciente a la plantilla "empresa", en el que su campo "nombre" da igual el valor que tenga, que su campo "capital" tiene que ser un valor "10" y que su campo "PYME" sea igual a "si":

```
?e <- (empresa (nombre ?n) (capital 10) (PYME "si"))
```

*El valor **?e** es la variable donde se almacena la referencia al "hecho", que active la "regla" cuando se cumplan las condiciones con la plantilla "empresa". La asignación se hace con el operador **<-**.*

Además de los valores concretos o indefinidos, que tiene que tener un *hecho*, sobre los campos de la plantilla instanciada dentro de una *regla*, también se pueden poner condiciones mediante la sentencia *test*.

En el *Ejemplo 2.1.3* se tiene una muestra de cómo se representaría, y de cómo funcionaría:

Ejemplo 2.1.3

*Para utilizar la sentencia "**test**" se pondría como aparece a continuación, donde en este caso, lo que se hace es verificar la variable **?c** es **<=** (menos o igual) que el valor **10**:*

```
(test (<= ?c 10))
```

La "*parte derecha*", es donde se va a realizar la acción sobre el "*hecho*" que ejecuta la "*regla*", cada vez que se cumple la "*parte izquierda*".

En el *Ejemplo 2.1.4,* se muestra como podría ser la *parte derecha* de una *regla*, donde se realizan cambios sobre un *hecho* que ha cumplido la *parte izquierda* de una *regla*. Se pueden hacer cambios sobre el *hecho* con la sentencia *modify*, o se puede mostrar algo por pantalla con la sentencia *printout*:

Ejemplo 2.1.4

*Con la sentencia "**modify**" se puede cambiar el valor de los campos de un "hecho". En este caso, se realiza un cambio sobre el "hecho" albergado en la variable **?e**, donde a la variable de "capital" **?c** se le suma **10**:*

```
(modify ?e (+ ?c 10)))
```

*Siempre que se quiere hacer una acción que suponga la aplicación de un operador de suma, resta, multiplicación, producto, división o cualquier otro, se pone el "operador" y a continuación los valores a operar en el orden adecuado, para que así sea correcto. Por ejemplo, si a **?c** se le quiere restar **10** se pondría:*

```
(modify ?e (- ?c 10)))
```

Si se pusiera:

```
(modify ?e (- 10 ?c)))
```

*Se restaría a **10** el valor de **?c**, por lo que el orden si que influye en el resultado.*

*Si un valor se quisiera mostrar por pantalla habría que utilizar la sentencia "**printout t**", como se ve a continuación:*

```
(printout t "CAPITAL = " ?c crlf))
```

*Para introducir texto a mano se pondría entre comillas dobles, en este caso **"CAPITAL = "**, y si se quiere imprimir el valor de uno de los campos, se pone el nombre de la variable que lo representa, que en este caso sería **?c**. Para cerrar el "**printout t**" se pone "**crlf**".*

Ahora que ya se sabe la estructuración de una "*regla*" en "*parte izquierda*" y "*parte derecha*", en el *Ejemplo 2.1.5* se tiene una muestra de cómo sería una regla al completo:

Ejemplo 2.1.5

Se tiene la regla "listar", que en su "parte izquierda" tiene como condiciones que haya un "hecho" de la plantilla "empresa" con "nombre", "capital" y "PYME", sin importar cuál sea su valor. A continuación, se comprobaría que el valor del capital sea inferior o igual a 10, caso en el que se pasaría a la "parte derecha", y donde se modificaría el valor del capital sumándole 10, imprimiendo el valor resultante:

```
(defrule listar
     ?e <- (empresa (nombre ?n) (capital ?c) (PYME ?p))

     (test (<= ?c 10))

=>
     (modify ?e (+ ?c 10)))
     (printout t "CAPITAL = " ?c crlf))
)
```

La *"prioridad"* de una "*regla*" se representa mediante el campo "*salience*", al que se le da un valor numérico. Una "*regla*" tendrá "*prioridad*" sobre otra si ese valor es superior que el de la otra.

El campo "*salience*" se colocaría al principio de la regla declarada, como se muestra en el *Ejemplo 2.1.6*:

Ejemplo 2.1.6

```
(defrule listar
     (declare (salience 123))
     ?e <- (empresa (nombre ?n) (capital ?c) (PYME ?p))
=>
     (modify ?e (+ ?c 10)))
     (printout t "CAPITAL = " ?c crlf))
)
```

Además de conocer que son los *hechos* y las *reglas* con su sistema de prioridades, es fundamental saber que:

- Cuando un *hecho* activa una *regla*, lo que pasa es que se crea una *instancia* de esa *regla* asociada a ese *hecho* determinado.

- Por *instancia* se puede entender como *copia* de una *regla*.

- Se ejecutarán primero las *instancias* de las *reglas* con mayor *prioridad*.

- A medida que se ejecutan las *instancias*, puede que otras *reglas* se activen por los cambios que se vayan produciendo en los *hechos*, y por ello quizá algunas de las nuevas *reglas* activas, tengan mayor prioridad que las que ya estaban activadas.

Con toda la información vista hasta ahora, ya se está en condiciones de comprender la ejecución de un programa en CLIPS.

En el E*jemplo 2.1.7* se tiene un caso explicado de cómo se desarrollaría un proceso de ejecución, en el que se tienen *3 hechos* y *3 reglas* con distintas prioridades:

Ejemplo 2.1.7

Se va a utilizar el "Hecho 1" descrito en el Ejemplo 2.1.1 que se muestra a continuación:

```
(deffacts mercado
     (empresa (nombre "Empresa1") (capital 10) (PYME "no"))
     (empresa (nombre "Empresa2") (capital 20) (PYME "no"))
     (empresa (nombre "Empresa3") (capital 30) (PYME "si"))
)
```

Y las 3 reglas siguientes:

```
(defrule beca
     (declare (salience 100))
     e? <- (empresa (nombre ?n) (capital ?c) (PYME ?p))
     (test (<= ?c 10))
=>
(modify ?e (+ ?c 10)))

(defrule tipo
     (declare (salience 93))
     e? <- (empresa (nombre ?n) (capital ?c) (PYME no))
     (test (< ?c 20))
=>
(modify ?e (PYME si)))


(defrule esPYME
     (declare (salience 123))
     e? <- (empresa (nombre ?n) (capital ?c) (PYME ?p))
=>
     (printout t ?n ?p "es PYME" crlf))
)
```

Se va a explicar paso a paso la ejecución de "Hecho 1" con las "reglas" descritas, desde la activación de las mismas, hasta su disparo:

Hecho 1: `(empresa (nombre "Empresa1") (capital 10) (PYME "no"))`

El "Hecho 1" realiza una instancia de todas las reglas. De "esPYME" porque se precisa que se tenga los campos "nombre", "capital" y "PYME" sin importar que valor contengan:

```
     e? <- (empresa (nombre ?n) (capital ?c) (PYME ?p))
```

De "tipo" porque para "nombre" y "capital" da igual el valor que tenga, y para "PYME" tiene que ser igual a "no". Luego el valor de "capital" tiene que ser menor que 20, y en este caso es igual a "10", así que también se activa:

```
e? <- (empresa (nombre ?n) (capital ?c) (PYME no))
(test (< ?c 20))
```

Y por último se activa "beca", porque para los campos "nombre" y "capital" no importa el valor que tengan, y para "PYME" tiene que ser "no":

```
e? <- (empresa (nombre ?n) (capital ?c) (PYME no))
```

Ahora, para el orden en el que se ejecutarán, toma su papel la prioridad de cada regla. En este caso la regla "beca" tiene una prioridad de 100, la regla "tipo" tiene prioridad de 93, y la regla "PYME" tiene una prioridad de 123.

De esto se deduce que el orden de ejecución para "Hecho 1" es primero "esPYME", luego "beca" y por último "tipo". A continuación se explica los cambios que se producen sobre "Hecho 1" en la "parte derecha" de cada regla:

1. ***"esPYME" con "Hecho 1"***

   En la parte derecha de esta regla se tiene:

   ```
   (printout t ?n ?p "es PYME" crlf))
   ```

   Está línea de código realiza una impresión por pantalla. En este caso saldrá por pantalla ***"Empresa1 no es PYME"*** *siendo "?n" = "Empresa1" y "?p" = "no".*

2. ***"beca" con "Hecho 1"***

   En la "parte derecha" de esta regla se tiene la siguiente línea de código:

   ```
   (modify ?e (+ ?c 10)))
   ```

La sentencia "modify" permite realizar modificaciones sobre los campos de un "hecho". En este caso se cambia el valor de "capital" en "Empresa1" sumándole "10". En ese momento "capital" pasa a ser de "20".

3. "tipo" con "Hecho 1"

Por último, en la "parte derecha" de "tipo" se modifica el campo "PYME" y se cambia el "no" por un "si":

```
(modify ?e (PYME sí)))
```

Con la ejecución de la regla "tipo", "Hecho1" quedaría de la siguiente manera:

```
(empresa (nombre "Empresa1") (capital 20) (PYME "si"))
```

De esta manera, se habrían ejecutado todas las "reglas" activadas por los "hechos iniciales".

Ahora "Hecho 1", con los datos modificados tras las "disparar" las "reglas" activadas con los "hechos iniciales", mira si ahora "activa" alguna "regla". En este caso crearía instancia solo de la regla "esPYME", y se procedería de la siguiente manera:

1. "esPYME" con "Hecho 1"

Esta vez imprimirá por pantalla "Empresa1 si es PYME", siendo "?n" = "Empresa1" y "?p" = "si":

```
(printout t ?n ?p "es PYME" crlf))
```

Con el disparo de esta "regla", terminaría la ejecución ya que solo se volvería a activar "esPYME", por lo que ya no se producirían más cambios.

2.2- Simulador

El lenguaje CLIPS, descrito en el *apartado 2.1*, fue el elegido por *Daniel Sánchez* en su *proyecto fin de carrera* para la creación del *motor de simulación*, sobre el que ahora se está trabajando.

Cómo el objetivo del simulador es que una serie de empresas compitan entre sí de forma automática, el desarrollo en un "*sistema experto*" de programación, es una solución acertada.

Llevado al lenguaje CLIPS, como se ha visto en *Ejemplo 2.1.7*, el simulador funcionaría de la siguiente manera:

- Cada *hecho* representaría una *empresa*.
- Cada *empresa* tiene unos valores iniciales en los campos que contiene.
- Hay una serie de *reglas*, que representan *fórmulas*, que realizan *cálculos*, relacionados con unos determinados *módulos* de estudio.
- Al inicio del simulador, cada *hecho* activará e instanciará las *reglas* de las que cumpla los requisitos de su *parte izquierda*.
- Luego se ejecutarán en el orden que marquen las prioridades de las *reglas instanciadas* por cada *hecho*.
- Tras ejecutar cada ristra de *reglas instanciadas*, cada *hecho* con sus *campos* modificados, activará las *reglas* de las que cumpla su *parte izquierda*, y así sucesivamente hasta que llegue un momento que se active solo una, que ya no produzca cambios que posibiliten la activación de otras *reglas*, o de ninguna.

De esta manera, cada *empresa* irá evolucionando a medida que va ejecutando *reglas* que van cambiando sus datos, y que le habilitan para ejecutar otras *reglas*, y así hasta que ya no activen ninguna más, y quede con unos datos finales distintos a los que tenían al inicio.

Las *reglas* que dan forma al *motor simulador de mercados*, están repartidas entre 9 *módulos*. A continuación se va a describir cada uno de ellos, y para el *módulo de calidad* también se incluyen dos ejemplos. Los ejemplos del resto de *módulos* están incluidos en el apartado "*5.1- Simulador (ejemplos)"* de los *apéndices*:

1) **Módulo de calidad**: Está dividido en dos partes. Una serie de reglas son referidas a las "*Decisiones de calidad*" definidas con el prefijo "*Deci-*". Y el otro sector está dedicado a "*Cálculos de calidad*", distinguidas por el prefijo "*Cal-*" que son *cálculos*, y que dependen de los resultados de otras *reglas* para poder devolver sus propios resultados.

   En el *Ejemplo 2.2.1* tenemos una muestra de una regla de "*Decisiones de calidad*", y en el *Ejemplo 2.2.2* una de "*Cálculos de calidad*":

   <u>Ejemplo 2.2.1</u>: Regla de "Decisiones de Calidad"

```
(defrule Deci-003

(declare (salience 458))

(object (is-a EMPRESA) (NOMBREEMPRESA ?neo) (IDEMPRESA
?ideo)(PERIODO ?po))
?res <- (object (is-a RESULTADOSCALIDAD) (IDEMPRESA ?ideo) (PERIODO
?po) (CAL003W ?calc3) (CAL005W ?calc5))
?dec <- (object (is-a DECISIONES))
(contadorcal (cequipos ?ideo))
(test (>= ?calc3 ?calc5))
=>
(modify-instance ?dec (DECI003 0.7)))
```

Ejemplo 2.2.2: Regla de "Cálculos de Calidad"

```
(defrule Cal-003

(declare (salience 480))

(object (is-a EMPRESA) (NOMBREEMPRESA ?neo) (IDEMPRESA ?ideo)
(PERIODO ?po))
(object (is-a HISTORIALDECISIONES) (IDEMPRESA ?ideo) (PERIODO ?po)
(DID ?did0))
(object (is-a HISTORIALDECISIONES) (IDEMPRESA ?ideo) (PERIODO
?p_1&:(eq (- ?po ?p_1) 1)) (DID ?did_1))
(object (is-a HISTORIALDECISIONES) (IDEMPRESA ?ideo) (PERIODO
?p_2&:(eq (- ?p_1 ?p_2) 1)) (DID ?did_2))
(object (is-a HISTORIALDECISIONES) (IDEMPRESA ?ideo) (PERIODO
?p_3&:(eq (- ?p_2 ?p_3) 1)) (DID ?did_3))
?res <- (object (is-a RESULTADOSCALIDAD) (IDEMPRESA ?ideo) (PERIODO
?po))
(contadorcal (cequipos ?ideo))

=>

(modify-instance ?res (CAL003W (* (/ (- ?did0 ?did_1) ?did_1)
100)))
(modify-instance ?res (CAL003X (* (/ (- ?did_1 ?did_2) ?did_2)
100)))
(modify-instance ?res (CAL003Y (* (/ (- ?did_2 ?did_3) ?did_3)
100)))

)
```

2) **Módulo de decisiones**: Configurado por *reglas* definidas con el prefijo "*Dem-*", como se puede ver en el *Ejemplo 2.2.3*.

3) **Módulo de pérdidas y ganancias**: Configurado por *reglas* definidas con el prefijo "*Pyg-*", como se puede ver en el *Ejemplo 2.2.4*.

4) **Módulo de tesorería**: Configurado por *reglas* definidas con el prefijo "*tsr-*", como se puede ver en el *Ejemplo 2.2.5*.

5) **Módulo de balance**: Configurado por reglas definidas con el prefijo "*Bal-*", como se puede ver en el *Ejemplo 2.2.6*.

6) **Módulo de ratios**: Configurado por *reglas* definidas con el prefijo "*Rat-*", como se puede ver en el *Ejemplo 2.2.7*.

7) **Módulo de valor de mercado**: Configurado por *reglas* definidas con el prefijo "*Vmc-*", como se puede ver en el *Ejemplo 2.2.8*.

8) **Módulo de informe sectorial**: Configurado por *reglas* definidas con el prefijo "*Ise-*", como se puede ver en el *Ejemplo 2.2.9*.

9) **Módulo de análisis de decisiones** Configurado por *reglas* definidas con el prefijo "*Anr-*", como se puede ver en el *Ejemplo 2.2.10*.

2.3- JAVA y JGRAPH

El lenguaje de programación elegido para el desarrollo del sistema gráfico ha sido JAVA, entre otras cosas porque contiene la programación "*orientado a objetos*".

Este tipo de programación se basa en el desarrollo desde una perspectiva, lo más cercana posible, a la vida real. Esto se consigue con la definición de *objetos*, que representen las partes que configuran el producto final que se quiere definir, estableciendo una comunicación entre las mismas para que el resultado sea el deseado.

Por ejemplo, para la configuración de un coche, se definiría un *objeto* para las *ruedas*, otro para la *carrocería*, otro para el *motor*, y así sucesivamente con todas sus partes. Luego, se establece la comunicación pertinente entre los *objetos*, con el *montaje* para que el *coche* pueda funcionar.

Pues para el desarrollo del *sistema gráfico* del *motor de simulación* es un poco igual. Se tiene una serie de *reglas*, relacionadas mediante *atributos*, englobadas dentro de una serie de *módulos* determinados. Estas mismas partes se pueden definir como *objetos*, que unidos de una forma determinada estructuren el resultado deseado.

Luego, dentro de JAVA, se ha utilizado una de sus librerías para la creación de los gráficos, a partir de la estructura de *objetos* creada en JAVA. En esta ocasión ha sido la librería JGRAPH, perteneciente a SWING, que ofrece un amplio servicio de generación de diagramas, *grafos* y *objetos*.

Capítulo 3.

Trabajo realizado

3- Trabajo realizado

En los siguientes apartados, se va a describir el proceso completo de cómo se ha llevado a cabo el desarrollo del *sistema gráfico*.

3.1- Análisis

Se parte de la necesidad de obtener la representación gráfica del conjunto de "*reglas*" que forman el *motor de simulación*.

Una vez que se comprende el funcionamiento de un programa en CLIPS, el siguiente paso es el de entender la distribución de las *reglas*, y su relación mediante los diferentes *atributos*.

Este análisis servirá para saber de qué forma tratar el código en CLIPS, y así extraer la información que interesa, pudiendo configurar el sistema gráfico.

Para ello se va a mostrar a continuación en el *Ejemplo 3.1.1* una *regla* del motor, y se va a desglosar su información viendo las partes de las que se compone, y cuáles son las referencias en las que apoyarse para obtener los datos que se necesitan:

Ejemplo 3.1.1

```
---------------------------------------------------------------------------------------------------------
( defrule Deci-003
(declare (salience 458))
(object (is-a EMPRESA) (NOMBREEMPRESA ?neo)(IDEMPRESA ?ideo) (PERIODO ?po))

?res <- (object (is-a RESULTADOSCALIDAD)(IDEMPRESA ?ideo) (PERIODO ?po)
(CAL003W ?calc3) (CAL005W ?calc5))
?dec <- (object (is-a DECISIONES))
(contadorcal (cequipos ?ideo))
(test (>= ?calc3 ?calc5))
=>
(modify-instance ?dec (DECI003 0.7))
---------------------------------------------------------------------------------------------------------
```

Ejemplo 3.1.1

De esta *regla* se sacan las siguientes valoraciones, que serán útiles a la hora de implementar el programa que extraiga la información que se necesita:

1) Como este lenguaje utiliza **"*paréntesis*"** para el acotado de *reglas*, y de los *campos* dentro de las mismas, y como se sabe que cada "*paréntesis de apertura*" tiene un "*paréntesis de fin*", éstos serán una buena referencia para controlar cuando empieza y termina cada *regla*.

2) El nombre **"*Deci-003*"** resaltado en ***azul*** es el nombre de la ***regla***, y se encuentra siempre a continuación de "*defrule*". Saber esto es importante porque el nombre de las *reglas* es uno de los datos que se quiere almacenar.

3) Los nombres **"*CAL003W*", "*CAL005W*" y "*DECI003*"** resaltados en ***rojo*** son ***atributos***. En este caso ***los dos primeros son de entrada****,* que se identifica porque están ***antes del "=>"*** resaltado en verde (en la parte izquierda de la *regla*). Y el ***"DECI003" es el atributo de salida*** de la *regla*, y se sabe porque está ***después del "=>"*** (en la parte derecha de la "*regla*").

Además de los detalles que se acaban de desglosar, sabiendo la estructura global del motor, es importante tener en cuenta que:

1) Los "*atributos*" que en algunas *reglas* son de *entrada* en otras son de *salida*, y viceversa.

2) Las "*reglas*" de un determinado *módulo* tienen el mismo comienzo de nombre que las identifica y asocia a su *módulo* (Ej. **Deci**-003 es del módulo de **Decisiones**).

3) Las *reglas* que pertenecen a un *cálculo*, son aquellas que tienen el mismo nombre de comienzo y mismo valor numérico, pero que incluyen una letra como cierre del nombre (ejemplo: Deci-003Y sería regla del cálculo Deci-003).

Tras el análisis de estos fundamentos iniciales, muy importantes para implementar la aplicación, se pasa a describir cómo sería la estructuración del producto final, representado a través de un *diagrama de flujo* explicado a continuación mediante la *Figura 3.1.2* y la *Figura 3.1.3*:

Figura 3.1.2: Ventana de inicio

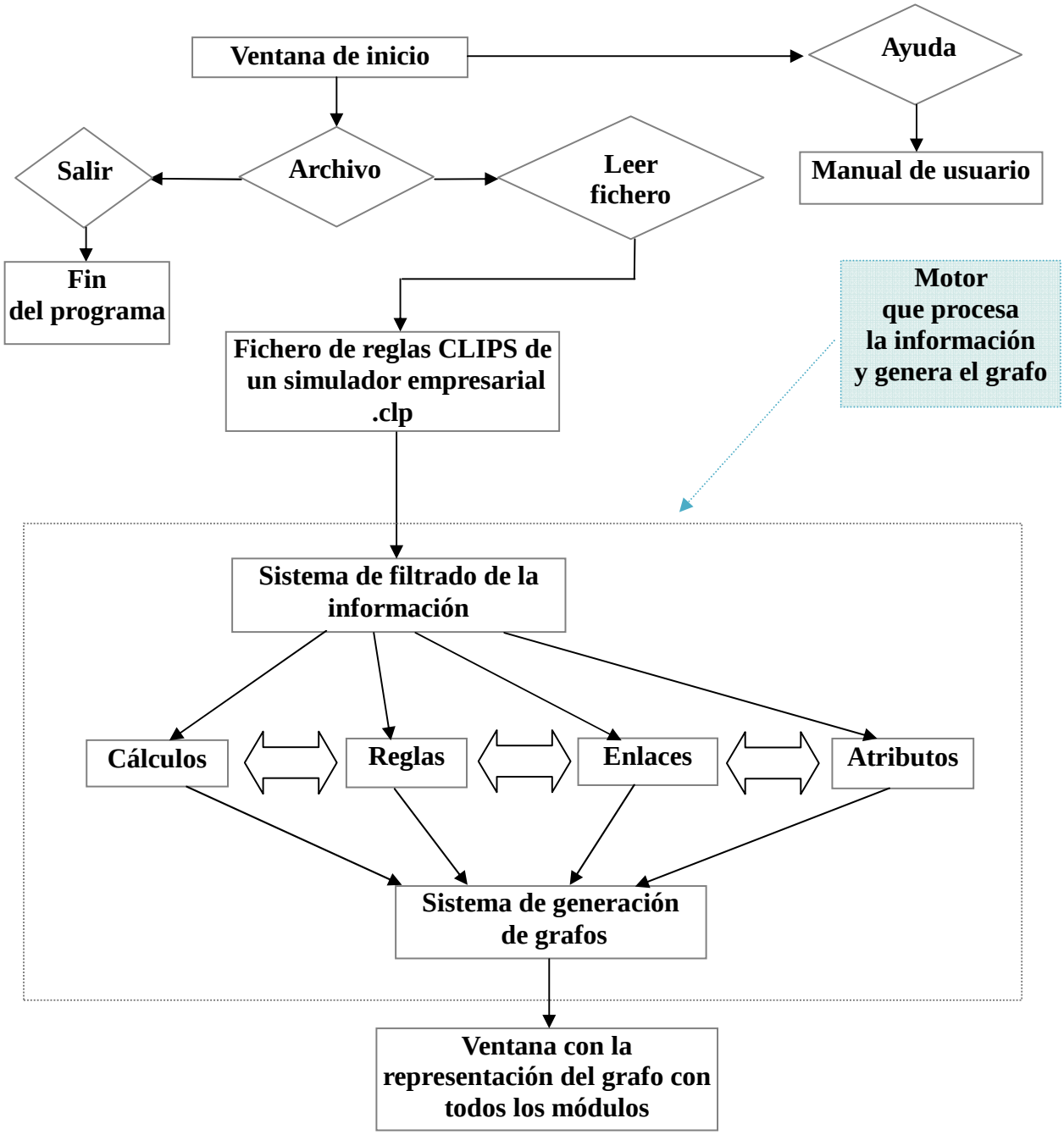


Figura 3.1.2

Se parte de la “*Ventana de inicio*”, en la que aparecerá nada más que el “*menú contextual*”, donde se tienen las opciones “*Archivo*” y “*Ayuda*”.

Si se selecciona la opción “*Ayuda*” se llegaría a “*Manual de usuario*”, y ahí finalizaría esa ruta.

Si se selecciona “*Archivo*” se tiene la opción de “*Leer fichero*” o de “*Salir*”. Esta última nos llevaría a “*Fin del programa*”, que terminaría con la ejecución.

Seleccionando *“Leer*” permite la carga de un fichero con extensión *.clp*, siempre que albergue el programa desarrollado en CLIPS del “*motor de simulación*”. Esto se realizará a través de una “*ventana de exploración y carga de archivos*”.

Una vez se carga el fichero correcto, el programa extrae la información sobre la relación entre las *reglas* mediante sus *atributos*, y muestra el *grafo* con los *módulos* existentes.

En la *Figura 3.2.2* se explican las posibilidades que se ofrecen en esta situación, habiendo cargado el fichero y teniendo en pantalla el *grafo* de los *módulos* relacionados entre sí.

Figura 3.1.3 Ventana con la representación del *grafo* con todos los *módulos*

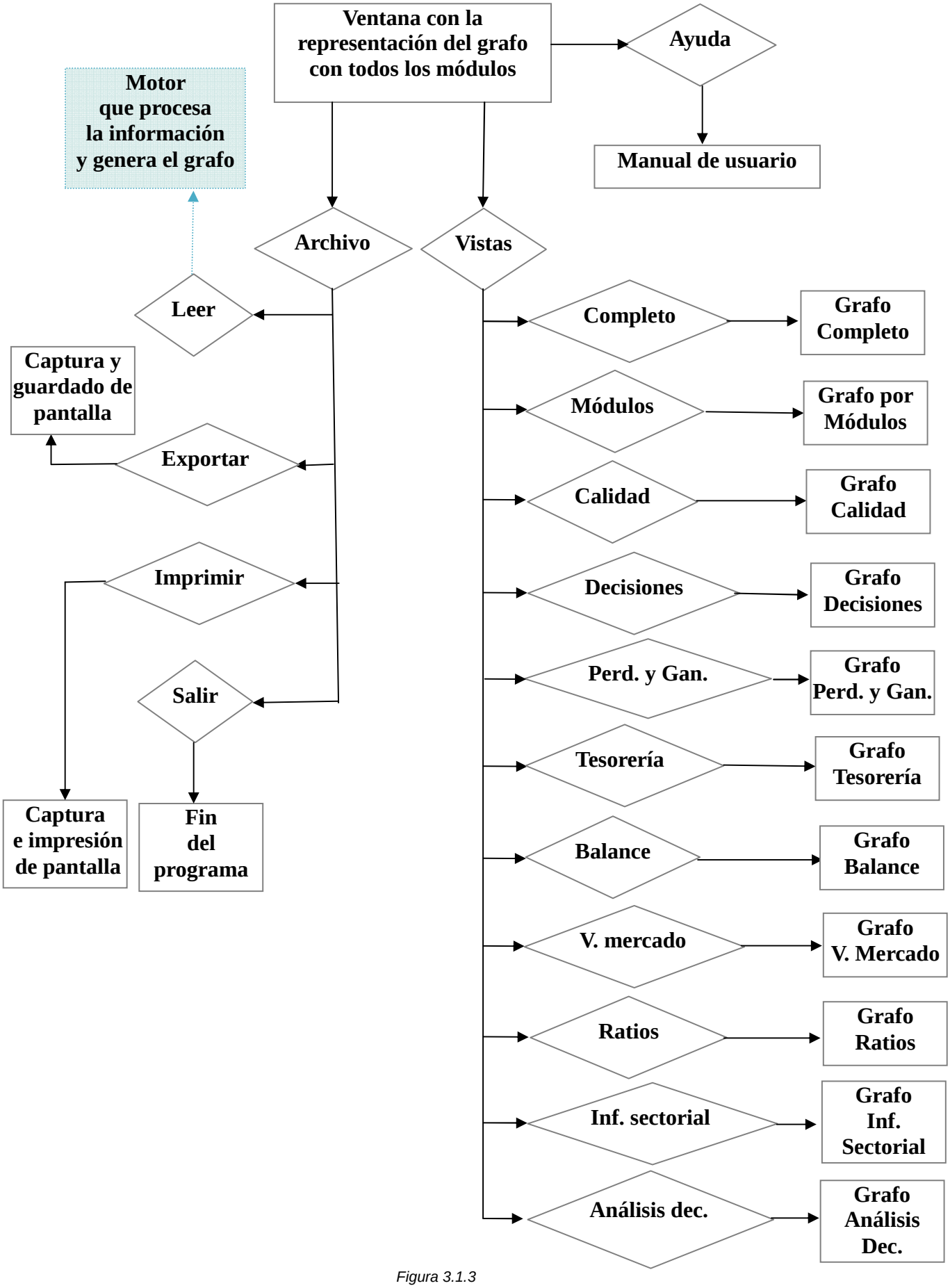


Figura 3.1.3

Una vez se tiene el *grafo* con los *módulos* en pantalla, se tienen algunas opciones más en el menú contextual, que se suman a las que había en la pantalla de inicio.

Se puede elegir entre las opciones de "*Archivo*", "*Vistas*" y "*Ayuda*". En este último ocurre igual que al principio, que se tiene la opción de ir a "*Manual de usuario*" solamente.

Luego, en "*Archivo*" se tienen las opciones de "*Leer*", "*Exportar*", *"Imprimir*" y *"Salir*". Tanto "*Leer*" como "*Salir*" mantienen las mismas posibilidades que en la *Figura 3.1.3*.

Si se elige "*Exportar*", se realiza una "*Captura y guardado de la pantalla*", que consiste en realizar el *pantallazo* de la imagen en pantalla y poderla guardar mediante una ventana de "*exploración y guardado de archivos*".

Si se selecciona "*Imprimir*" se realiza una "*Captura e impresión de la pantalla*", que como en el caso de "*Exportar*", también realiza un *pantallazo*, pero después abre un menú de impresión en papel.

Todo lo representado en la *Figura 3.1.2* y la *Figura 3.1.3* es la base sobre la que se apoya el *diseño del programa*, siendo la hoja de ruta donde tomar las referencias sobre "el qué" se quiere realizar para encontrar con mayor facilidad "*el cómo*".

3.2- Diseño

A continuación, se va a explicar cómo se han implementado el "*sistema de grafos del motor de reglas CLIPS*", en lo que se refiere a las "*clases"* que se han creado, la comunicación entre ellas, y la función de cada una dentro de la interfaz gráfica.

El desarrollo del "*Sistema gráfico de visualización del motor de reglas clips de un simulador empresarial*", como se ha dicho anteriormente, se ha realizado en el lenguaje de programación JAVA, usando sus librerías JGRAPH e ITEXT 5.1.2.

El código desarrollado en JAVA, se ha dividido en una serie de módulos que realizan una serie de tareas que se denominan "*clases de JAVA*".

En estas *clases* se implementan los algoritmos necesarios para la creación del *interfaz gráfico*, desde la carga de ficheros *.clp* del "*motor de simulación*", hasta la aparición por pantalla de los *grafos* que muestren la interrelación entre las *reglas* de los diferentes *módulos*.

Las *clases* que se han creado para este programa, son las que se muestran a continuación, diciendo brevemente de que tarea se encargan durante la ejecución:

- **Main.java**: Esta clase se encarga de iniciar el programa, mandando ejecutar la clase "*VentanaOrigen.java*". Aquí se inicia y se cierra la ejecución.

- **VentanaOrigen.java**: Esta clase genera la ventana de inicio del programa, desde la que se puede leer un archivo *.clp* con el *motor de simulación* en su interior, salir del programa y abrir el manual de usuario.

- **SplashFrame.java**: Esta clase se ejecutará en el momento que se mande leer un fichero *.clp*. Da soporte al control del procesado del fichero, dando información en pantalla del porcentaje de la carga y realizando la llamada a la clase "*lectura.java*", que continuará el filtrado y almacenamiento de la información que se necesita.

- **lectura.java**: Esta clase se encarga de la apertura y posterior lectura del fichero *.clp*. A continuación va leyendo una a una las *reglas*, almacenándolas temporalmente, para luego mandarlas a la función "*Almacenar.java*" donde se extraen y almacenan los datos que interesan.

  Después se ejecuta "*Grafos.java*", que es la clase que direcciona los datos almacenados hacia las clases de creación y mostrado de *grafos*.

- **Almacenar.java**: En esta clase se va a guardar toda la información acerca de las reglas", los *cálculos*, los *módulos*, los *atributos* y la relación que existe entre todos ellos, sobre una estructura de listas configurada en las clases "*Calculo.java*", "*reglas.java*", "*enlace.java*" y "*Atributo.java*", que se describen a continuación.

- **Calculo.java**: Estructura en la que se almacena el nombre del *cálculo* de cada *regla*. También guarda la relación de cada *cálculo* con las *reglas* que corresponda.

- **reglas.java**: Estructura en la que se almacena el nombre de cada *regla* y del *módulo* al que pertenece. También guarda la relación con el *listado de cálculos y enlaces* con los que se corresponda.

- **enlace.java**: En esta clase se almacena la información del listado de *atributos* en "*Atributo.java*", teniendo separados los *atributos de entrada* de los *atributos de salida,* guardando también su relación con las *reglas* según sean unos u otros *atributos*.

- **Atributo.java**: Estructura en la que se almacenan todos los *atributos* del sistema empresarial, sin diferenciar si son de *entrada* o de *salida*, ya que esa separación la realiza la clase "*enlace.java*" con la que al mismo tiempo están relacionados.

- **Grafos.java**: Esta clase se encarga de redirigir las peticiones de las distintas *vistas* llamando a las clases que generan los distintos *grafos*. Esta tarea la realiza, de una u otra forma, dependiendo de si se está en la ventana tras la carga del fichero, o si es la elección de una de las vistas disponibles en el menú de la ventana.

- **GrafoAltonivelSimple.java**: Esta clase es la encargada de configurar el *grafo* con la interrelación entre los *módulos* existentes en el *motor de simulación*, que luego se mostrará con la ayuda de la clase "*Ventana.java*". Es la clase que crea el grafo de inicio tras la carga de un fichero, y el que surge si se elige en el menú de la ventana la "*vista por módulos*".

- **Ventana.java**: Esta clase genera la "*ventana*" donde se muestra el *grafo* creado en la clase "*GrafoAltoNivelSimple.java*". Esta *ventana* incluye un menú más completo que la ventana de inicio además de poder "*leer"* un nuevo fichero, "*salir del programa*" y ver el "*manual de usuario*".

  Las nuevas opciones que aparecen son "*Exportar*" la captura de la imagen a formato *.jpg* o *.pdf*, "*imprimir*" la captura de la pantalla, y la selección de las *vistas* de los módulos existentes.

- **ImageFileFilter.java**: Esta clase es un módulo de apoyo para la captura de una imagen. Controla que no se introduzca un formato erróneo al guardar una imagen en "*jpg*".

- **grafoCompleto.java**: Esta clase configura el *grafo* con la totalidad de las *reglas* y *cálculos*, con las relaciones entre ellos a través de los *atributos* de *entrada* y de *salida*. Este *grafo* lo sacará por pantalla la clase "*VentanaCompleto.java*".

- **ModuloCalidad.java**: Esta clase configura el *grafo* que con las *reglas* integradas dentro del módulo de "*calidad*", relacionadas mediante los *atributos* de *entrada* y de *salida*.

- **ModuloDemanda.java**: Esta clase configura el *grafo* que con las *reglas* integradas dentro del módulo de "*decisiones de la demanda*", relacionadas mediante los *atributos* de *entrada* y de *salida*.

- **ModuloPyG.java**: Esta clase configura el *grafo* que con las *reglas* integradas dentro del módulo de "*pérdidas y ganancias*", relacionadas mediante los *atributos* de *entrada* y de *salida*.

- **ModuloTesoreria.java**: Esta clase configura el *grafo* que con las *reglas* integradas dentro del módulo de *"tesorería*", relacionadas mediante los "*atributos*" de "*entrada*" y de "*salida*".

- **ModuloBalance.java**: Esta clase configura el *grafo* que con las *reglas* integradas dentro del módulo de "*balance*", relacionadas mediante los *atributos* de *entrada* y de *salida*.

- **ModuloValorMercado.java**: Esta clase configura el *grafo* que con las *reglas* integradas dentro del módulo de "*valor de mercado*", relacionadas mediante los *atributos* de *entrada* y de *salida*.

- **ModuloRatios.java**: Esta clase configura el grafo que con las *reglas* integradas dentro del módulo de "*ratios*", relacionadas mediante los *atributos* de *entrada* y de *salida*.

- **ModuloInformeSectorial.java**: Esta clase configura el *grafo* que con las *reglas* integradas dentro del módulo de "*informe sectorial*", relacionadas mediante los *atributos* de *entrada* y de *salida*.

- **ModuloAnalisisDecisiones.java**: Esta clase configura el *grafo* que con las *reglas* integradas dentro del módulo de "*análisis de decisiones*", relacionadas mediante los *atributos* de *entrada* y de *salida*.

- **VentanaCompleto.java**: Esta clase es la encargada de mostrar el *grafo* creado en la clase "*grafoCompleto.java*", a través de una ventana emergente con los menús contextuales correspondientes.

- **VentanaCalidad.java**: Esta clase es la encargada de mostrar el *grafo* creado en la clase "*ModuloCalidad.java*", a través de una ventana emergente con los menús contextuales correspondientes.

- **VentanaDemanda.java**: Esta clase es la encargada de mostrar el *grafo* creado en la clase "*ModuloDemanda.java*", a través de una ventana emergente con los menús contextuales correspondientes.

- **VentanaPyG.java**: Esta clase es la encargada de mostrar el *grafo* creado en la clase “*ModuloPyG.java*", a través de una ventana emergente con los menús contextuales correspondientes.

- **VentanaTesoreria.java**: Esta clase es la encargada de mostrar el *grafo* creado en la clase “*ModuloTesoreria.java*", a través de una ventana emergente con los menús contextuales correspondientes.

- **VentanaBalance.java**: Esta clase es la encargada de mostrar el *grafo* creado en la clase “*ModuloBalance.java*", a través de una ventana emergente con los menús contextuales correspondientes.

- **VentanaValorMercado.java**: Esta clase es la encargada de mostrar el *grafo* creado en la clase “*ModuloValorMercado.java*", a través de una ventana emergente con los menús contextuales correspondientes.

- **VentanaRatios.java**: Esta clase es la encargada de mostrar el *grafo* creado en la clase “*ModuloRatios.java*", a través de una ventana emergente con los menús contextuales correspondientes.

- **VentanaInformeSectorial.java**: Esta clase es la encargada de mostrar el *grafo* creado en la clase “*ModuloInformeSectorial.java*", a través de una ventana emergente con los menús contextuales correspondientes.

- **VentanaAnalisisDecisiones.java**: Esta clase es la encargada de mostrar el *grafo* creado en la clase “*ModuloAnalisisDecisiones.java*", a través de una ventana emergente con los menús contextuales correspondientes.

- **nodoCal.java**: Esta es una clase que sirve de apoyo para la configuración de los nodos que “*son cálculos*”, en las clases que realizan la generación de *grafos*.

- **nodos.java**: Esta es una clase que sirve de apoyo para la configuración de los nodos que son *reglas*, en las clases que realizan la generación de *grafos*.

- **nodoMod.java**: Esta es una clase que sirve de apoyo para la configuración de los nodos que son *módulos*, en las clases que realizan la generación de grafos.

A continuación, a partir del diagrama utilizado durante el *análisis* del sistema gráfico en la *Figura 3.1.2* y la *Figura 3.1.3*, se van a mostrar en la *Figura 3.2.1*, la *Figura 3.2.2* y la *Figura 3.3.3* las partes del programa en las que toman partida cada una de las *clases* que se acaban de explicar:

Figura 3.2.1: Ventana de inicio

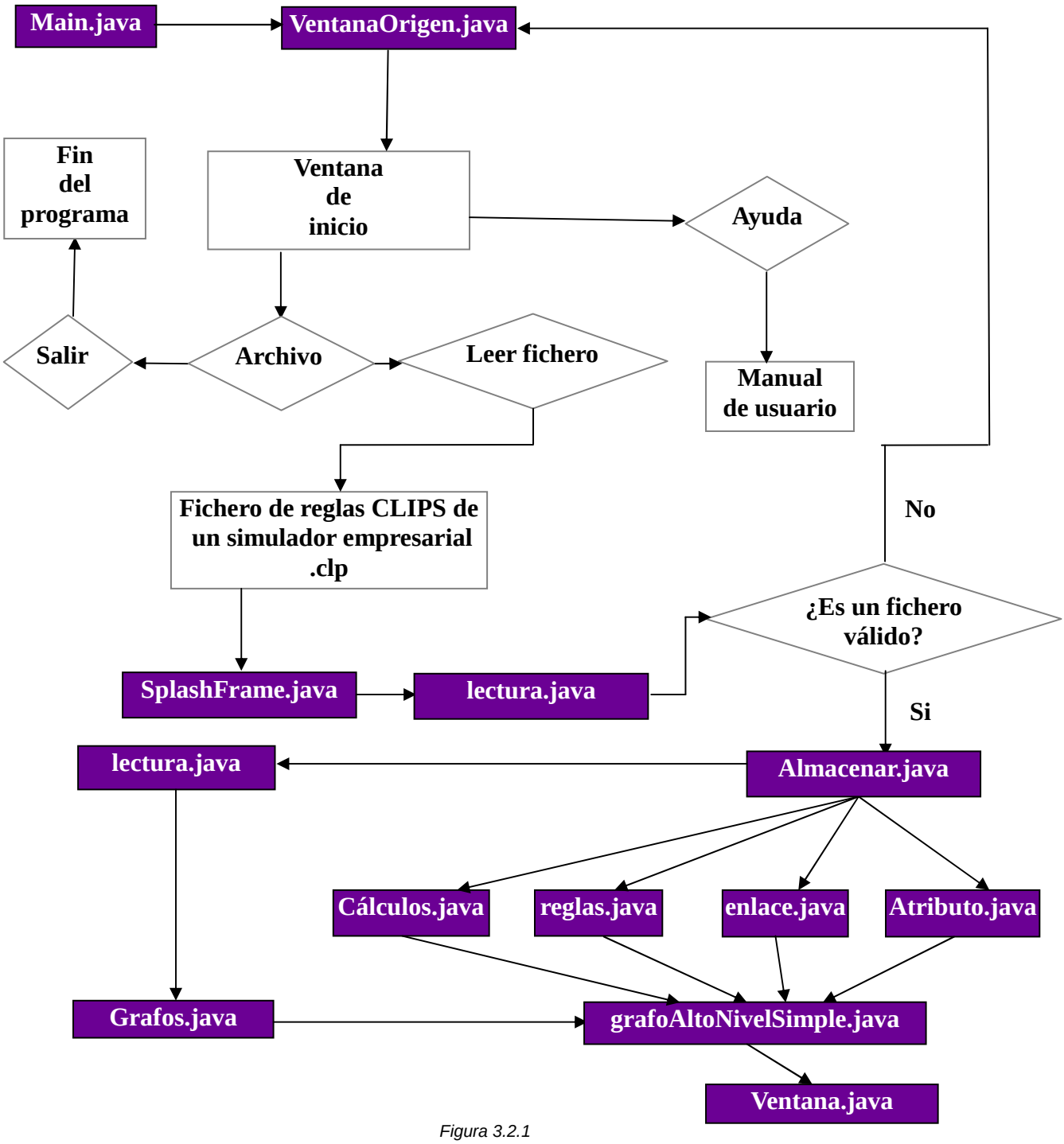


Figura 3.2.1

De esta primera figura se puede deducir que el programa se origina a partir de la clase *Main.java*, que es desde la que se llama a la clase *VentanaOrigen.java* que será la encargada de generar la "*Ventana de inicio"* de la aplicación.

Luego, cuando se lee un fichero *.clp* las clases *SplashFrame.java* y *lectura.java* son las que procesan y verifican que el archivo sea válido. Una vez se ha comprobado que es correcto con la clase *Almacenar.java* se va cogiendo la información de las *reglas* en el fichero y se va guardando en las clases *Cálculos.java, reglas.java, enlace.java y Atributo.java* la información ya filtrada para su posterior uso en la creación de *grafos.*

Una vez extraída toda la información y guardada en la estructura destinada para ello, mediante la clase *lectura.java* se va a llamar a la clase *grafos.java* que administra el tipo de *grafo* que se quiere generar en *grafoAltoNivelSimple.java,* que en este momento será el "*grafo por módulos inicial".* Finalmente el *grafo* se va a mostrar en pantalla por la clase *Ventana.java.*

Figura 3.2.2: Ventana con la representación del *grafo* con todos los *módulos*

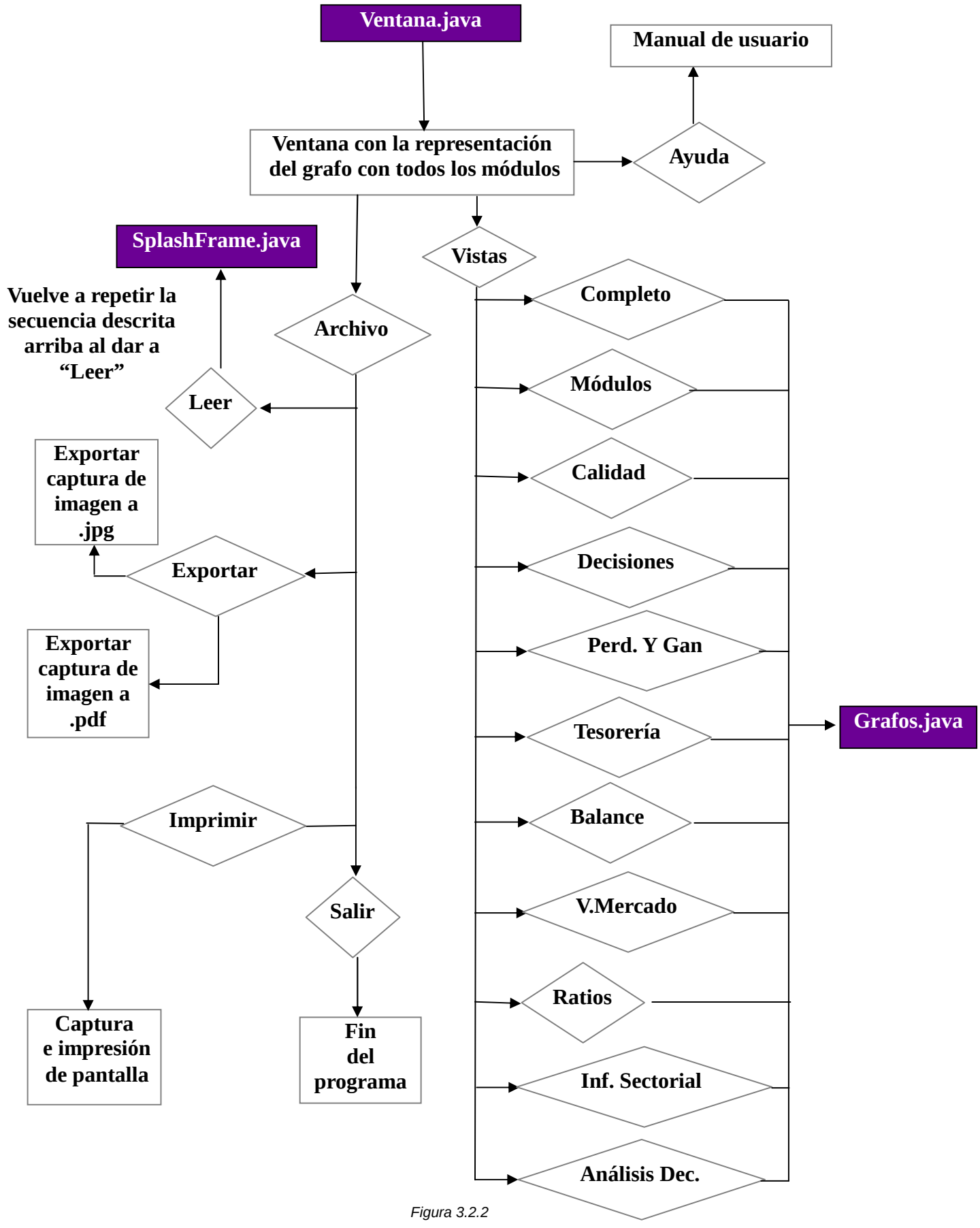


Figura 3.2.2

Desde la ventana con el “*grafo inicial por módulos”* se tienen disponibles todas las opciones del menú contextual.

En caso de dar a “Leer” la clase *SplashFrame.java* iniciaría el proceso de lectura como el descrito en la *Figura 3.2.1*. En el caso de que se seleccionen las opciones “*Exportar*”, “*Imprimir*”, “*Salir*” o “*Ayuda*” desde la misma clase “*Ventana.java*” se administrarán sus acciones.

Pero en el caso de seleccionar “*Vistas*” se llamará a “*Grafos.java*” que, dependiendo del tipo de vista seleccionada, direccionará a una *clase* en concreto. En la *Figura 3.2.3* se muestra como sería esa comunicación entre las *clases* dependiendo de la opción escogida.

Si por ejemplo se elige la opción *“Completo”* dentro de “*Vistas*”, se llamará a la *clase* “*grafoCompleto.java*” que creará el *grafo*, y ésta llamará a *“VentanaCompleto.java”* que lo mostrará por pantalla. Pues con el resto de posibilidades se procedería de la misma forma, pero con sus correspondientes *clases* asociadas al tipo de *vista*.

Figura 3.2.3: Vistas

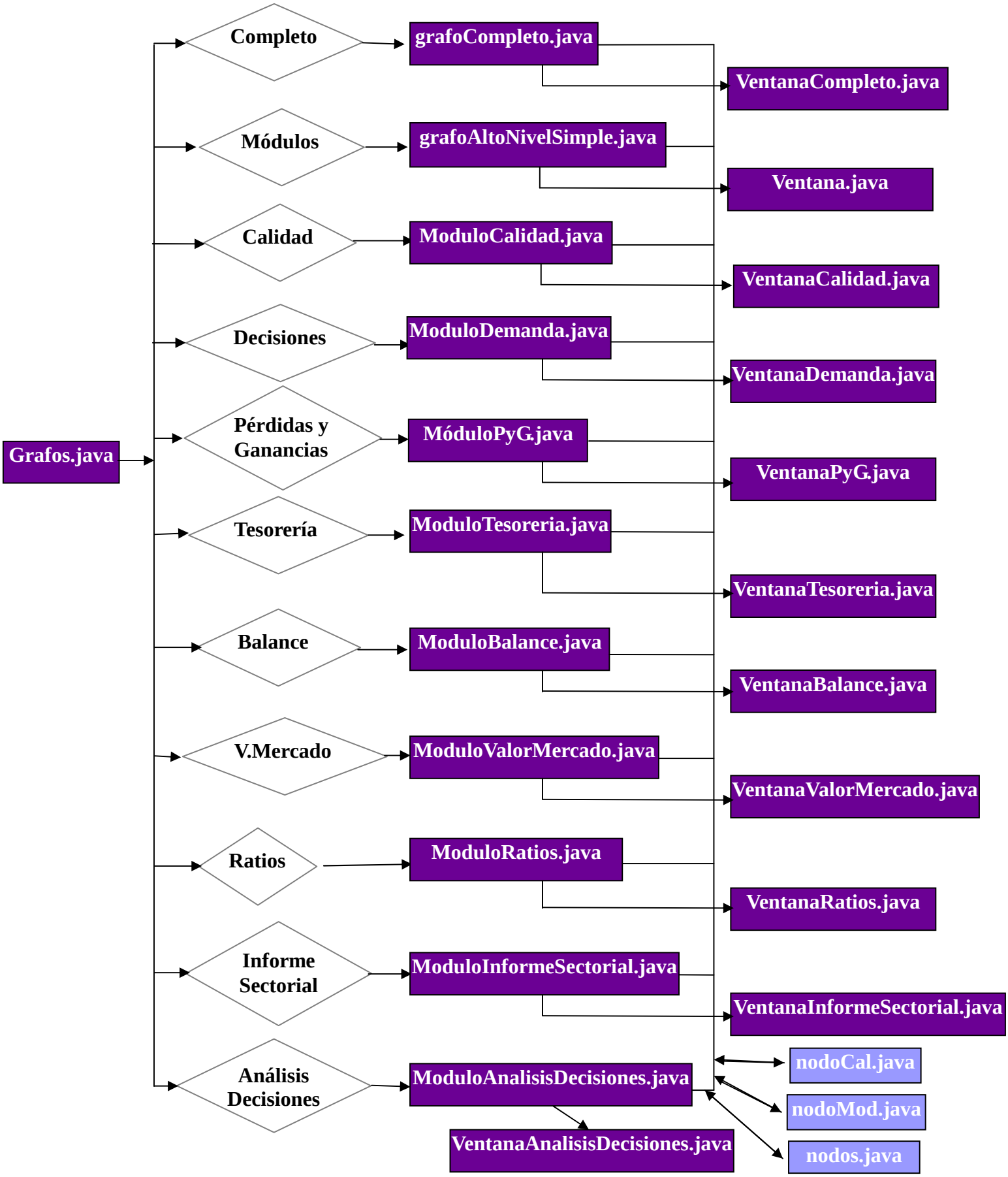


Figura 3.2.3

3.3- Evaluación

Tras la fase de implementación, se obtiene como resultado el programa que se ha llamado INVERVISION. Éste contiene todas las funcionalidades que se han ido explicando a lo largo de este proyecto, y que finalmente se convierte en el "*sistema de generación de grafos del motor de reglas CLIPS*".

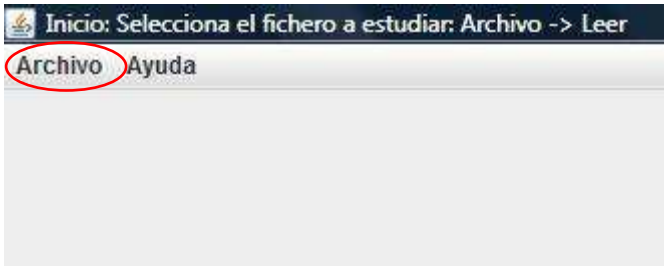
A continuación, se van a mostrar una serie de pruebas de "*buen funcionamiento*" del programa INVERVISION, donde se testean algunas de las partes críticas, más expuestas a dar problemas.

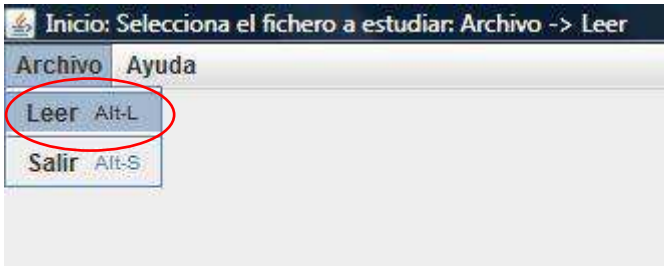
Los puntos a evaluar son sobre la "*lectura de ficheros*", la "*exportación a jpg y pdf*", la "*apertura de un fichero cuando ya se tiene uno abierto*" y el "*proceso de impresión*". Todos ellos se describen a continuación:

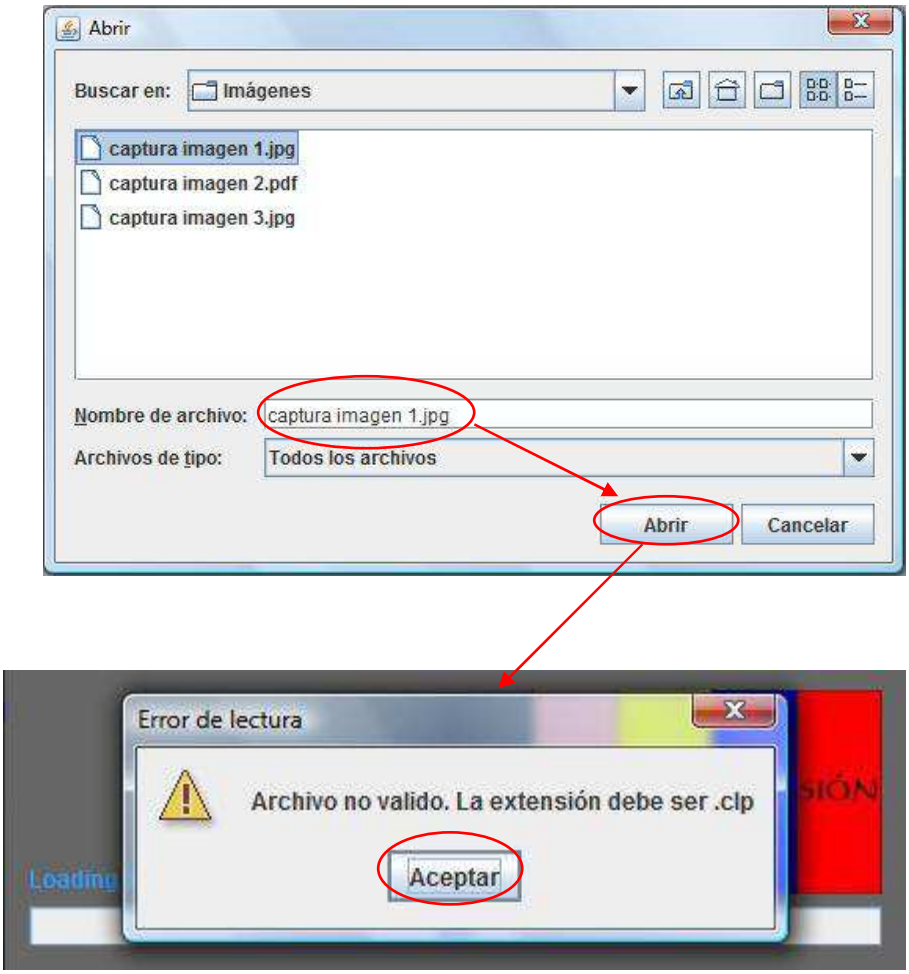
3.3.1- Leer fichero

La aplicación solo aceptará ficheros con extensión *.clp*. En cualquier otro caso el programa devolverá, mediante una ventana de "*error de lectura*", el mensaje "*Archivo no valido. La extensión debe ser .clp*".

A continuación se muestra un ejemplo de este tipo, en el que se intenta cargar un archivo con extensión .jpg:





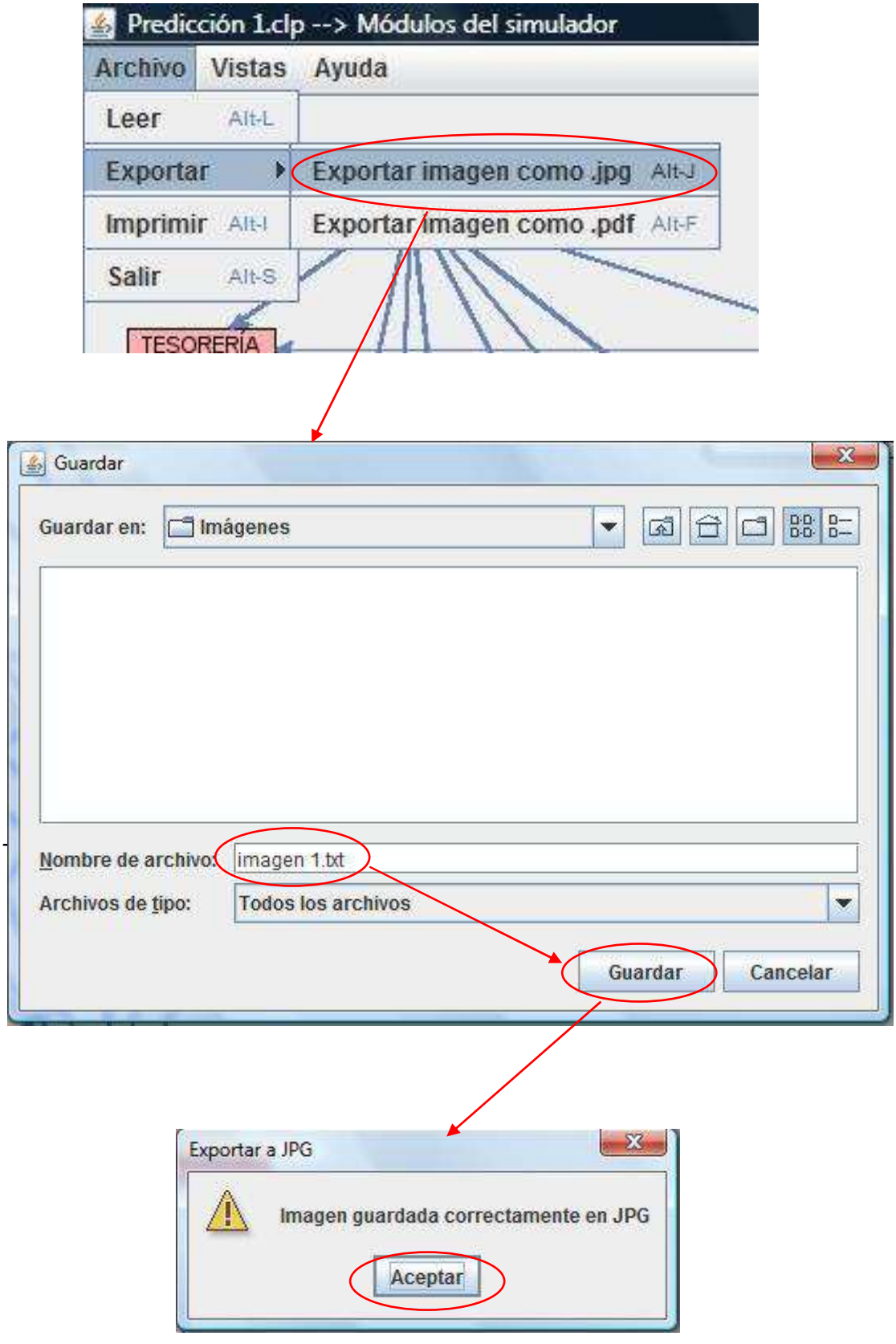


3.3.2- Exportar a jpg y pdf

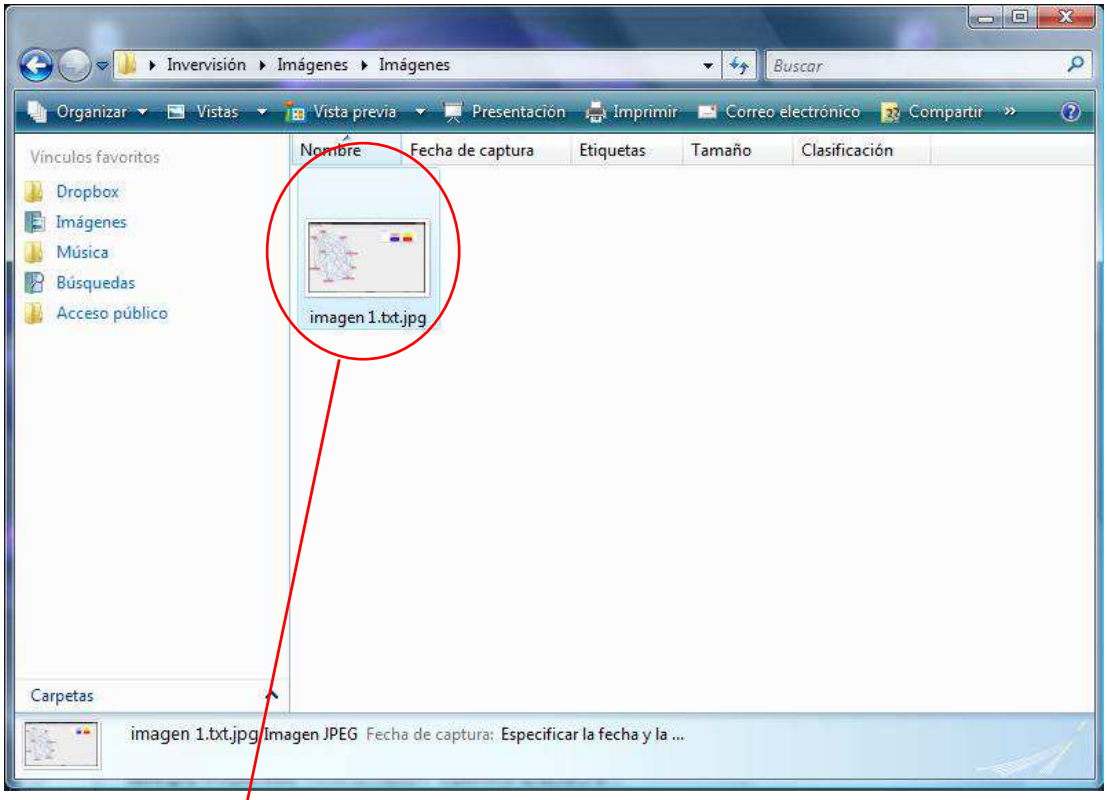
Al realizar una exportación a "*jpg*" o a "*pdf*", aunque al guardar el archivo se le ponga una extensión distinta, el programa guardará el archivo en el formato que se haya elegido, tomando la extensión escrita como parte del nombre del archivo.

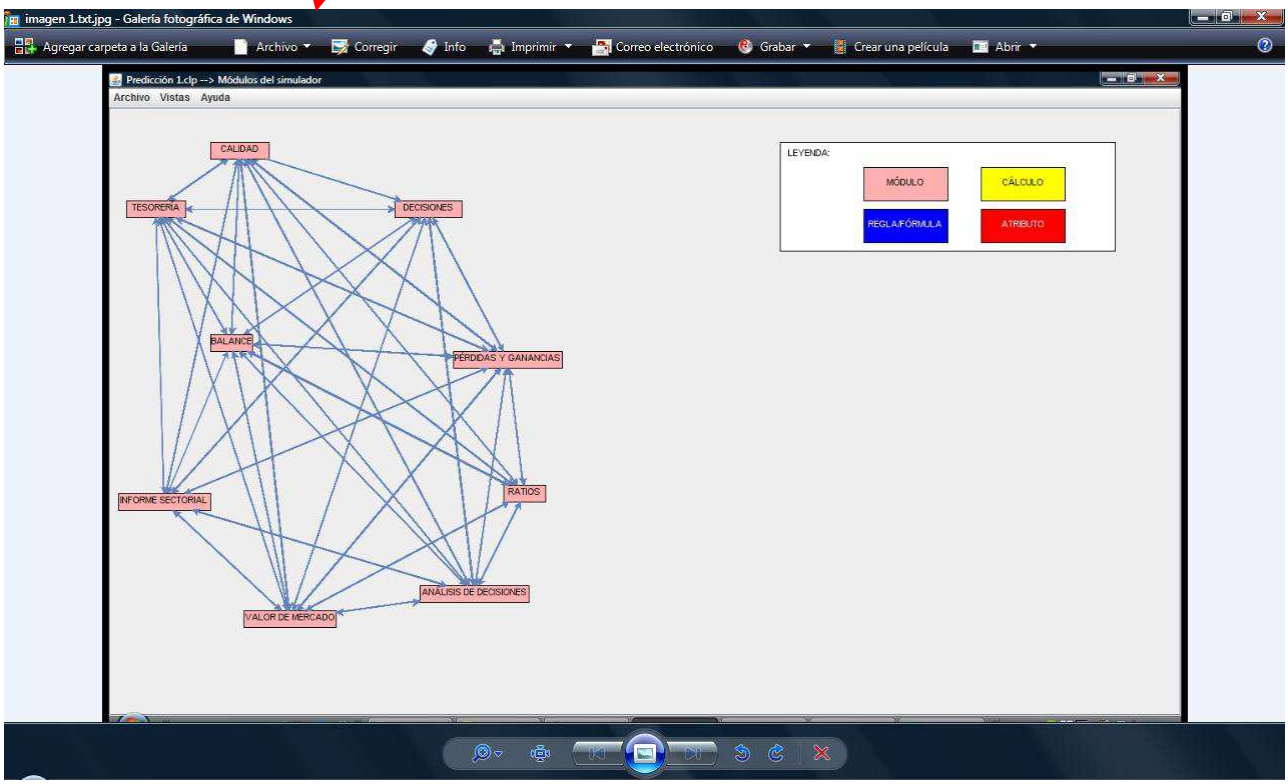
Ahora se van a mostrar dos ejemplos, uno de una exportación a "*jpg*" y otro a "*pdf*":

- Exportar a "*jpg*": Se pretende guardar la captura de la imagen como "*imagen 1.txt*", pero se va a guardar como "*imagen 1.txt.jpg*":

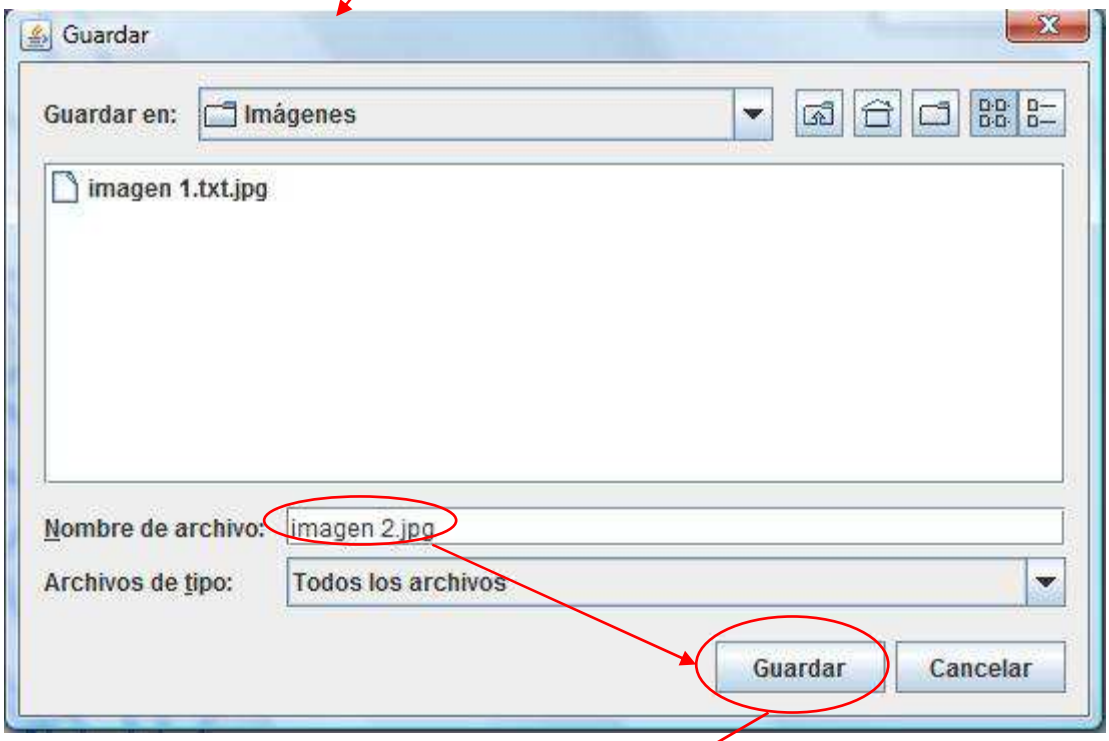


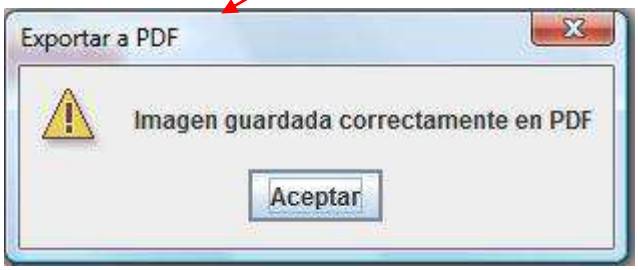
Si se va a la carpeta se puede ver cómo está guardada correctamente la imagen en "*jpg*":



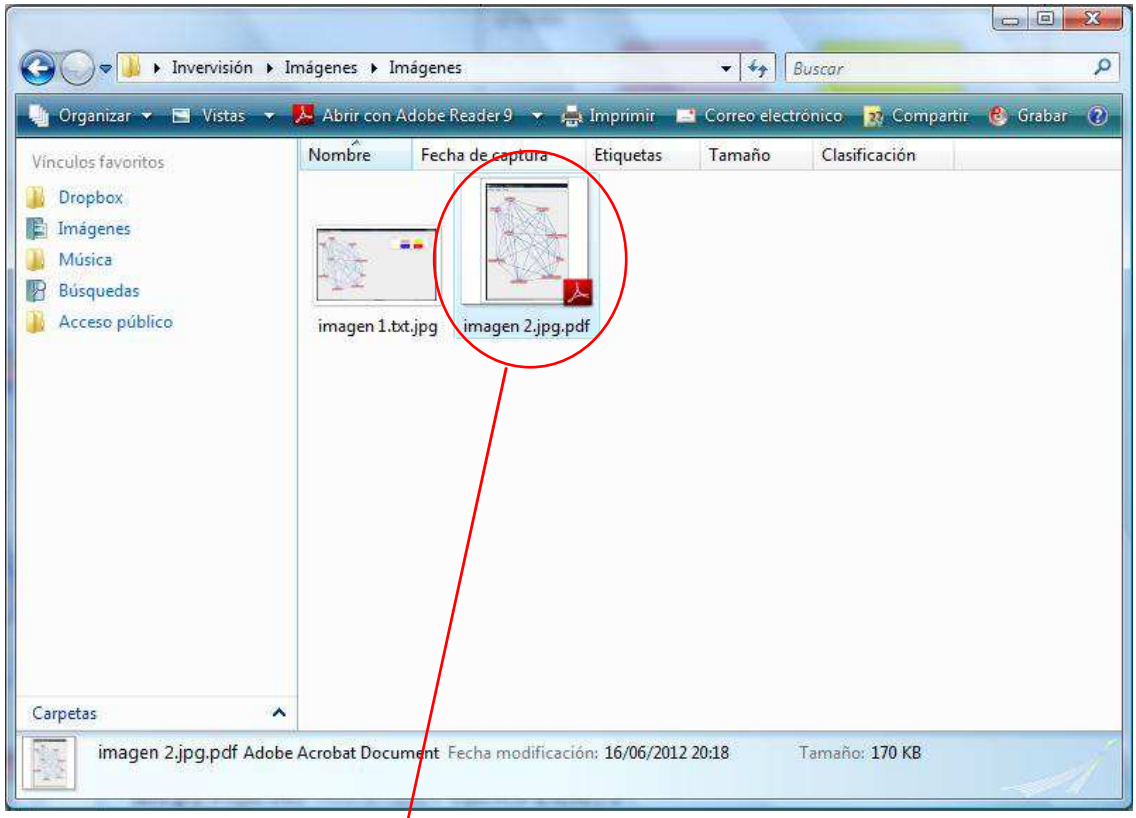


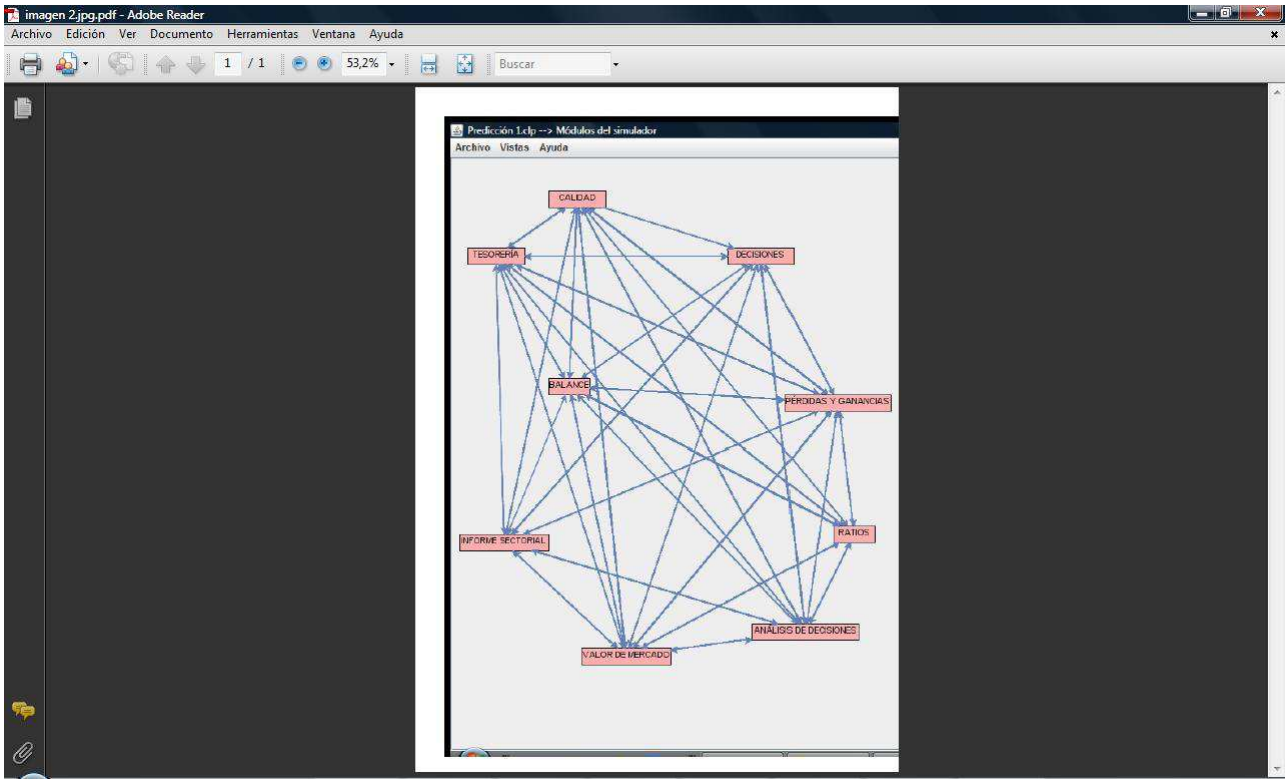
- Exportar a “pdf”: Se intenta guardar la “*captura de la imagen*” como “*imagen 2.jpg*”, pero se va a guardar como “*imagen 2.jpg.pdf*”:





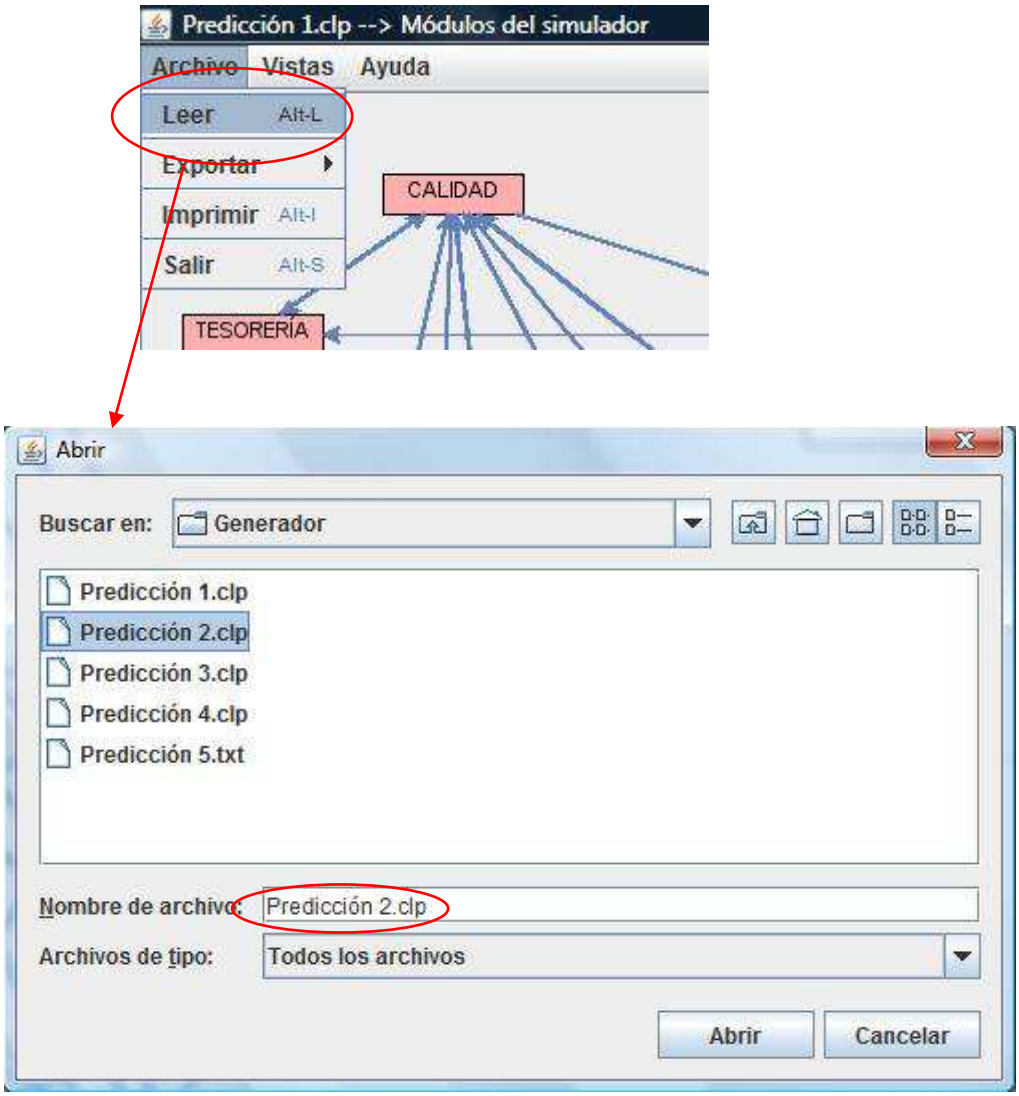
En la carpeta destino se puede comprobar que la imagen está guardada correctamente como "*pdf*":

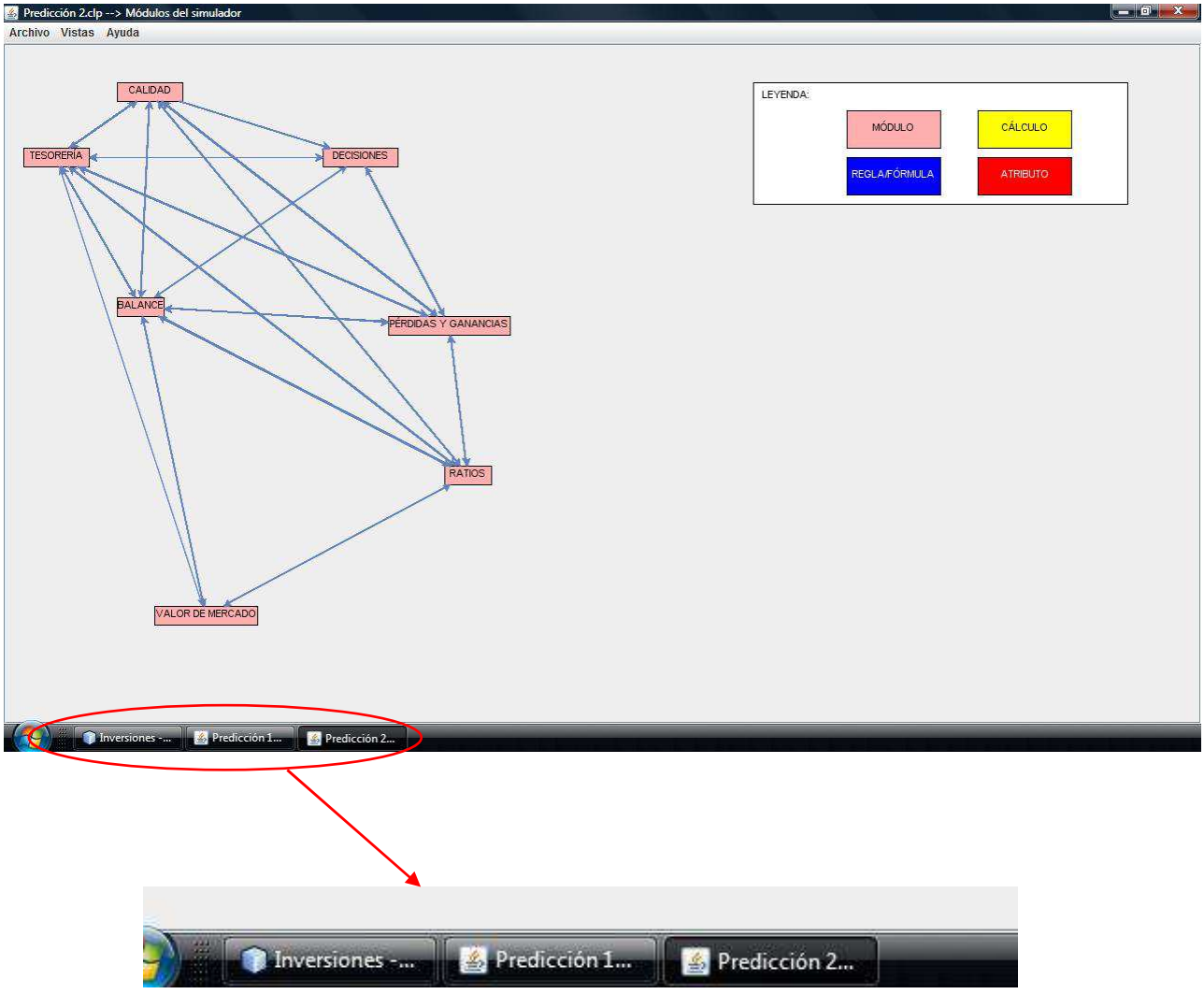




3.3.3-Abrir fichero, si ya hay uno o más abiertos

Existe la posibilidad de "*leer archivos*" aunque se tenga ya abierto uno. Se sigue el mismo procedimiento de "*lectura*" que al "*inicio del programa*", y el grafo resultante saldrá en una ventana aparte, sin cerrar lo que ya esté abierto. A continuación se tiene un ejemplo de este caso:



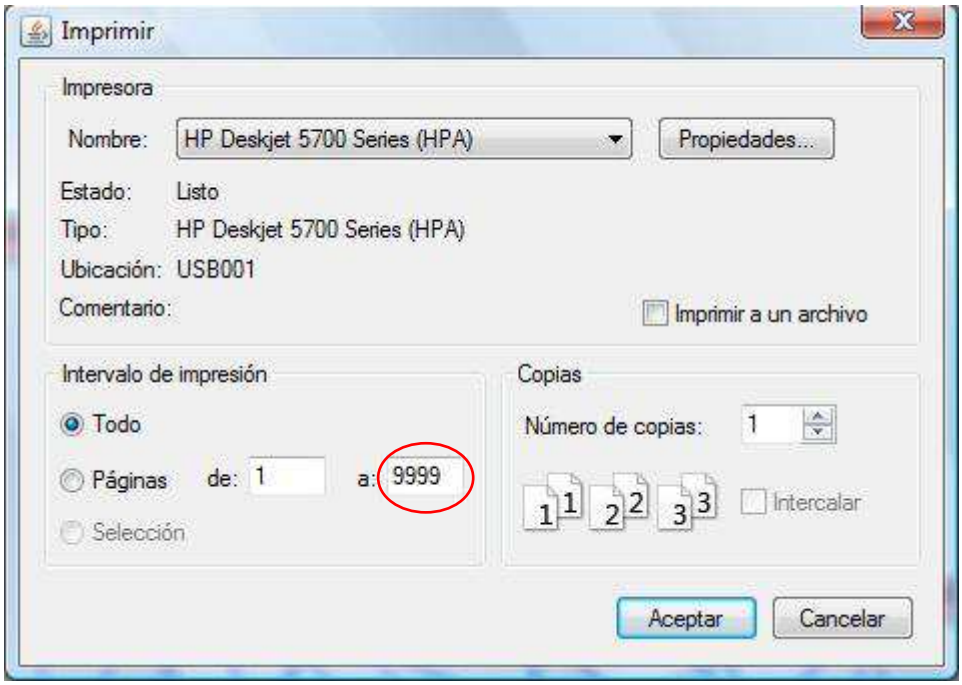


3.3.4- Imprimir

La funcionalidad de "*impresión*" es un caso en el que hay que tener un cuidado especial, ya que si no se siguen los pasos indicados la aplicación puede colapsarse.

Cuando se elige la opción "*Archivo -> Imprimir*" aparece el siguiente mensaje en pantalla:

Y a continuación aparece el menú de “*Preferencias de impresión*” del siguiente tipo:



En el caso de que no se sustituya el número “*9999*”, redondeado en rojo, por el número “*1*”, la aplicación se colapsará.

Este problema se produce derivado de que la impresión que se realiza es la “*captura*” de la imagen, y no de un número determinado de hojas de un documento. Por ello, es imprescindible acotar a una página la impresión de la imagen, para que la aplicación no se descontrole.

Capítulo 4.

Conclusiones y trabajos futuros

4- Conclusiones y trabajos futuros

Una vez finalizado el presente proyecto fin de carrera, es momento de realizar una valoración final del trabajo realizado, sacando a relucir los aciertos y los fallos que han acompañado a su desarrollo, y las posibilidades que se abren tras la consecución del mismo.

4.1- Conclusiones

Se ha desarrollado una aplicación que es capaz de interpretar un "*motor de simulación*" implementado en lenguaje CLIPS, para luego mostrar a través de *grafos* la relación existente entre las diferentes *fórmulas* y *atributos* que lo configuran.

Además, esta aplicación ofrece la funcionalidad de "*captura de imagen*", mediante la opción de exportarla a formato *.jpg* y *.pdf*, o de enviarla a la "impresora local" para su versión en papel.

El desarrollo del proyecto ha contado con numerosas dificultades, centralizadas sobre todo en dos:

- La comprensión de la estructura del motor desarrollado en CLIPS.

- El desarrollo de la aplicación mediante el lenguaje de programación JAVA.

El primero de ellos, el análisis del desarrollo en CLIPS, fue muy costoso debido sobre todo a la parte de compresión de cómo se relacionaban, dentro del código, las *reglas* entre sí mediante los *atributos*. Fue complicado realizar un patrón para identificar las partes, y así poder abordar el problema.

El segundo escoyo, va en parte relacionado con el primero. Para empezar, el lenguaje de programación JAVA, era un completo desconocido para mí, nunca antes había programado con él, por lo que solo esto ya suponía un reto. Si además se le añade la dificultad que implicaba la comprensión del desarrollo en CLIPS, la implementación a realizar en JAVA, para la extracción de la información que interesaba, crecía en dificultad.

En cualquier caso, aunque no haya sido de la forma más *eficiente*, aprovechando al máximo los recursos con los que se contaba, ni *óptima* en cuanto a funcionalidades que

se ofrece, se puede concluir que la aplicación *funciona*. Adaptada a los límites que fueron apareciendo durante el desarrollo, se puede tomar como una solución *válida*, pero con una *amplia posibilidad de mejora*.

4.2- Trabajos futuros

Cómo se puntualiza al final del *apartado 4.1*, esta aplicación tiene una "*amplia posibilidad de mejora*". Muchas de las funcionalidades ampliables forman parte de la idea original de este proyecto, y que por las dificultades encontradas, tuvieron que ser reformuladas.

La principal mejora en la que se podría trabajar sería la "*dinamización completa del motor de simulación*". Esto consistiría en que, desde el sistema gráfico, se pudieran modificar los nodos de las *reglas* y los *atributos*, que al mismo tiempo hiciera los mismos cambios sobre el programa en CLIPS, para que estos cambios se convirtieran en permanentes a todos los niveles.

Para aumentar aún más la calidad del producto, se podría desarrollar que el control de la ejecución del sistema empresarial en CLIPS, se incluyera en el propio sistema gráfico, para así tener todo centralizado.

Con estas dos funcionalidades se tendría una aplicación muy potente y útil, facilitando sobre todo la realización de pruebas sobre un motor de simulación en CLIPS, sin necesidad de estar constantemente accediendo al código para la realización de cambios.

Luego, en relación con la apariencia, sería buena una *modernización* del diseño del sistema de ventanas, menús contextuales y grafos, dándole otro aire más *dinámico* y *actual*, ya que las vistas de ahora son un tanto antiguas.

Estos serían algunos posibles avances, pero es seguro que se podría evolucionar la aplicación en otras muchas direcciones.

Capítulo 5.

Apéndices

5- Apéndices

5.1- *Simulador(ejemplos)*

Ejemplo 2.2.3: Regla de "Decisiones"

```
(defrule Dem-010

     (declare (salience 432))

(object (is-a EMPRESA) (IDEMPRESA ?ideo) (PERIODO ?po))
     (object (is-a HISTORIALDECISIONES)(IDEMPRESA ?ideo) (PERIODO
?po) (DPRECIOEMP ?dprecioemp_s))
(object (is-a RESULTADOSCALIDAD) (IDEMPRESA ?ideo) (PERIODO ?po)
(CAL014B ?est&: (>= ?est 7.0)))

?res <- (object (is-a RESULTADOSDEMANDA) (IDEMPRESA ?ideo) (PERIODO
?po) (ESTADODEM ?estdem&: (< ?estdem 6))
     (DEM002W ?demA))
     (contadordem (cequipos ?ideo))
=>
(modify-instance ?res (DEM010 (- (/ ?dprecioemp_s ?demA) 1))))
```

Ejemplo 2.2.4: Regla de "Pérdidas y ganancias"

```
(defrule Pyg-003

     (declare (salience 396))

     (object (is-a EMPRESA) (IDEMPRESA ?ideo)(PERIODO ?po))
     ?res <- (object (is-a RESULTADOSPYG) (IDEMPRESA ?ideo)
(PERIODO ?po) (PYG001 ?pyg001) (PYG002C ?pyg002C))
     (contadorpyg (cequipos ?ideo))
     =>
     (modify-instance ?res (PYG003 (- ?pyg001 ?pyg002C)) )
)
```

Ejemplo 2.2.5: Regla de "Tesorería"

```
(defrule Tsr-030

     (declare (salience 296))

     (object (is-a EMPRESA) (IDEMPRESA ?ideo)(PERIODO ?po))
     (object (is-a PESOS) (PPGTOEXTRAREDUCAP ?ppgtoextrareducap))
     (object (is-a HISTORIALDECISIONES)(IDEMPRESA ?ideo) (PERIODO
?po) (DREDCAP ?dredcap))
     (object (is-a RESULTADOSPYG) (IDEMPRESA ?ideo) (PERIODO ?po)
(PYG010 ?pyg10))
     ?res <- (object (is-a RESULTADOSTESORERIA) (IDEMPRESA ?ideo)
(PERIODO ?po))
     (contadortsr (cequipos ?ideo))
     =>
     (modify-instance ?res (TES030 (* ?ppgtoextrareducap ?dredcap
?pyg10)))
)
```

Ejemplo 2.2.6: Regla de "Balance"

```
(defrule Bal-046

     (declare (salience 248))

     (object (is-a EMPRESA) (IDEMPRESA ?ideo)(PERIODO ?po))
     ?res <- (object (is-a RESULTADOSBALANCE) (IDEMPRESA ?ideo)
(PERIODO ?po) (BAL045A ?bal045A) (BAL045B ?bal045B))
     (contadorbal (cequipos ?ideo))
     =>

     (modify-instance ?res (BAL046 (- ?bal045A ?bal045B))))
```

Ejemplo 2.2.7: Regla de "Ratios"

```
(defrule Rat-074

     (declare (salience 148))

     (object (is-a EMPRESA) (IDEMPRESA ?ideo)(PERIODO ?po))
     (object (is-a RESULTADOSTESORERIA) (IDEMPRESA ?ideo) (PERIODO
?po) (TES044B ?tes044B))
     (object (is-a RESULTADOSBALANCE) (IDEMPRESA ?ideo) (PERIODO
?po) (BAL058 ?bal058) (BAL071 ?bal071&: (> ?bal071 0)))
     ?res <- (object (is-a RESULTADOSRATIOS) (IDEMPRESA ?ideo)
(PERIODO ?po) (ESTADORAT 1))
     (contadorrat (cequipos ?ideo))
     =>
     (modify-instance ?res (RAT074 (/ (- ?bal058 ?tes044B)
?bal071)))
)
```

Ejemplo 2.2.8: Regla de "Valor de mercado"

```
(defrule Vmc-006
     (declare (salience 112))
     (object (is-a EMPRESA) (IDEMPRESA ?ideo)(PERIODO ?po))
     (object (is-a HISTORIALAREAFINANCIERA) (IDEMPRESA ?ideo)
(PERIODO ?pd) (RVALORMER ?valormer))
     ?res <- (object (is-a RESULTADOSVALORMERCADO) (IDEMPRESA
?ideo) (PERIODO ?po) (VMC004 ?vmc004))
     (contadorrat (cequipos ?ideo))
     (test (= (- ?po ?pd) 1))
     =>
     (modify-instance ?res (VMC006 (/ ?vmc004 ?valormer))))
```

Ejemplo 2.2.9: Regla de "Informe sectorial"

```
(defrule Ise-016

     (declare (salience 070))

     (object (is-a EMPRESA) (IDEMPRESA ?ideo)(PERIODO ?po))
     (object (is-a ENTORNO) (PERIODO ?po) (PIPC ?pipc))
     ?res <- (object (is-a RESULTADOSINFORMESECTORIAL) (IDEMPRESA
?ideo) (PERIODO ?po))
     (contadorise (cequipos ?ideo))
     =>
     (modify-instance ?res (ISE016 ?pipc))  )
```

Ejemplo 2.2.10: Regla de "Análisis de decisiones"

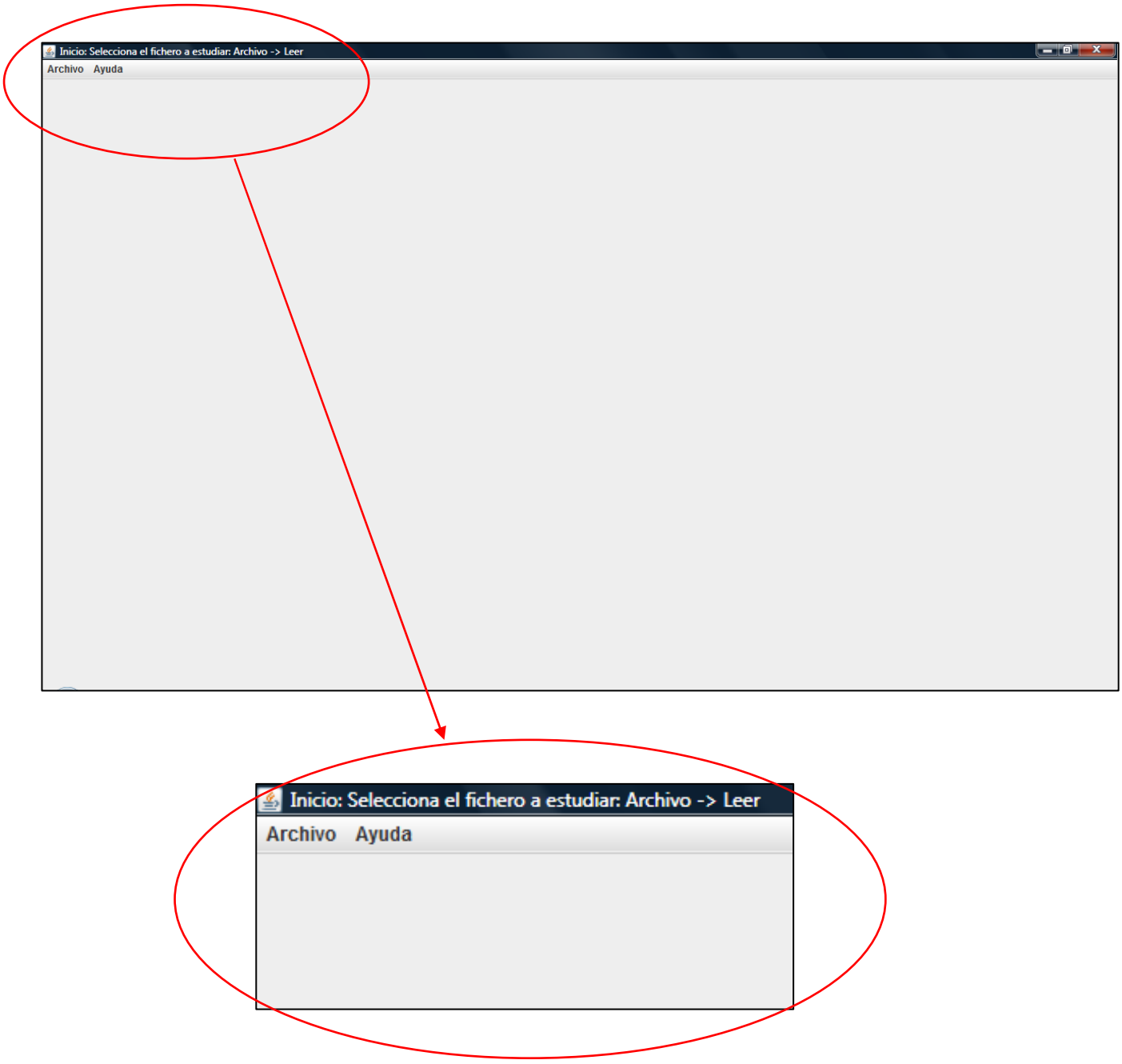
```
(defrule Anr-016

     (declare (salience 053))

     (object (is-a EMPRESA) (IDEMPRESA ?ideo)(PERIODO ?po))
     (object (is-a HISTORIALDECISIONES)(IDEMPRESA ?ideo) (PERIODO
?po) (DID ?did))
     (object (is-a RESULTADOSPYG) (IDEMPRESA ?ideo) (PERIODO ?po)
(PYG007F ?pyg007f))
     ?res <- (object (is-a RESULTADOSANR) (IDEMPRESA ?ideo)
(PERIODO ?po) (ESTADOANR 2))
     (contadoranr (cequipos ?ideo))
     =>
     (modify-instance ?res (ANR016 (/ ?did ?pyg007f)))
)
```

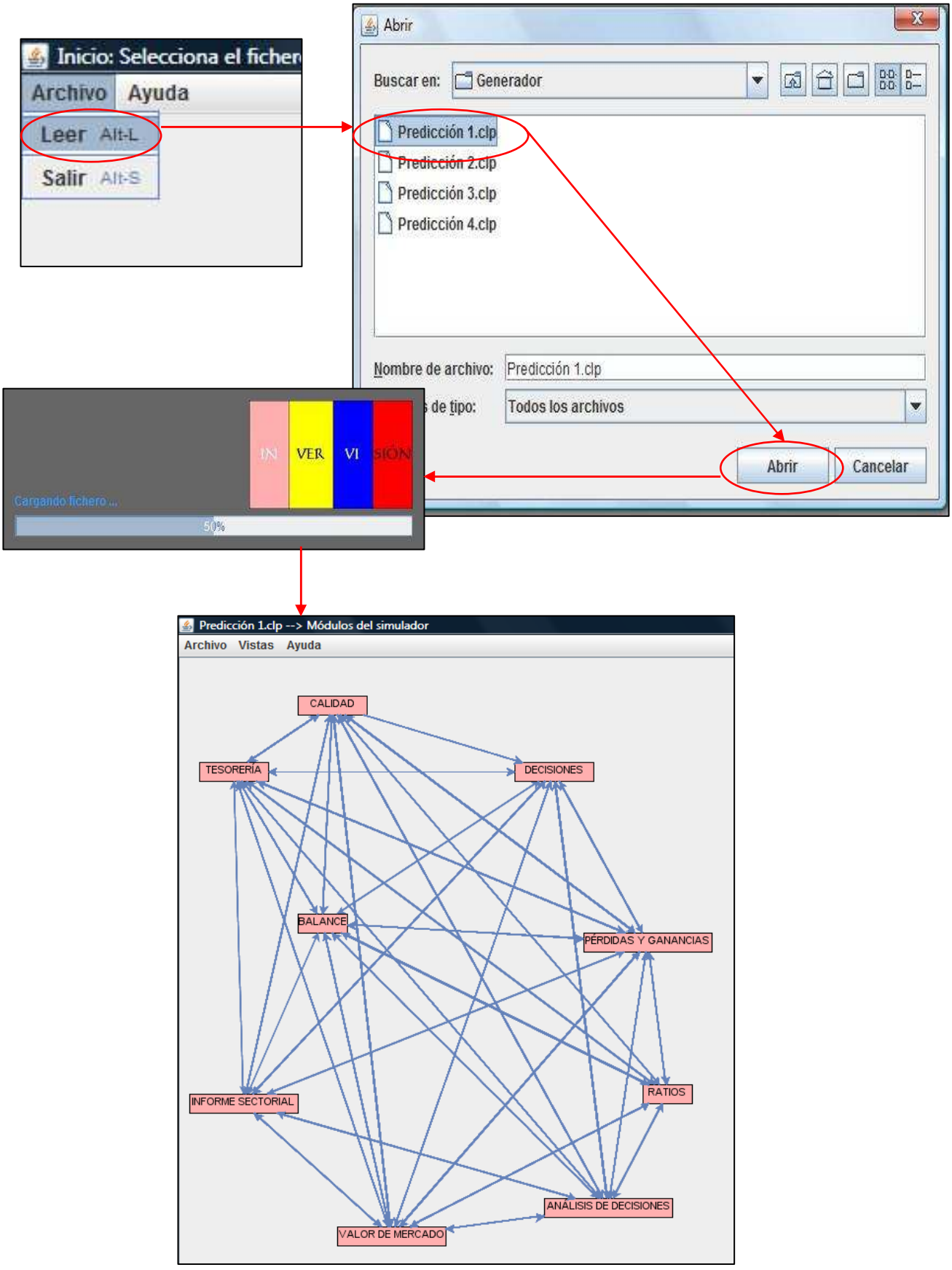
5.2- Manual de usuario

A continuación se va a mostrar la "*guía de uso*" de la aplicación INVERVISIÓN. Todo se inicia con la ejecución del programa desarrollado en JAVA con la que aparecerá la siguiente pantalla:



5.2.1- Leer fichero

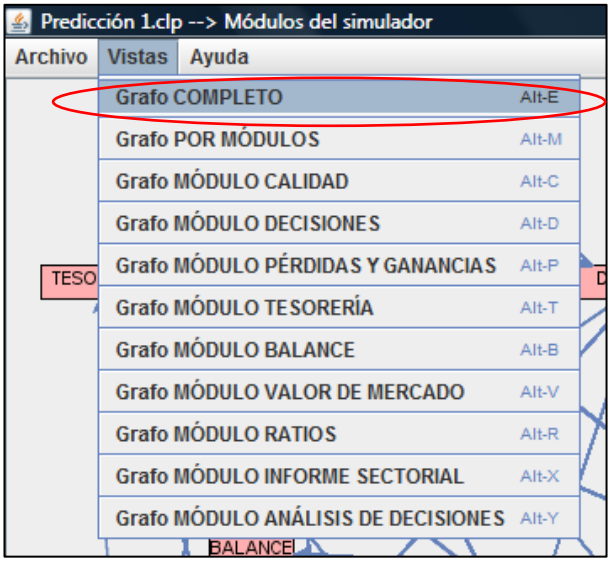
Para realizar la carga de un fichero hay que dar a “Archivo” ➔ “Leer”, en la ventana de exploración se selecciona el archivo con extensión *.clp* que contenga el “*motor de reglas CLIPS*” y tras dar a *Abrir* se mostrará el “*grafo por módulos*” :



5.2.2- Vistas

Tras la apertura del fichero se puede elegir mostrar los *grafos* de los *módulos* por separado a través del menú contextual "*Vistas*":

Por ejemplo, para mostrar la *vista* del conjunto de todas las *reglas* que componen el *motor de simulación* hay que seleccionar "*Vistas*" ➔ "*Grafo Completo*" y aparecerán en pantalla la totalidad de las *reglas* relacionadas por sus *atributos:*



Se puede apreciar que los nodos tienen un botón con un “+” que si se pulsa la *regla* se despliega y muestra los atributos que tiene en su interior. Así es como, la regla *Deci-004* tiene como *atributos de entrada* a *CAL003X* y *CAL005X* y como *atributo de salida* a *DECI004*, que luego son también atributos de entrada y salida de otras *reglas*.

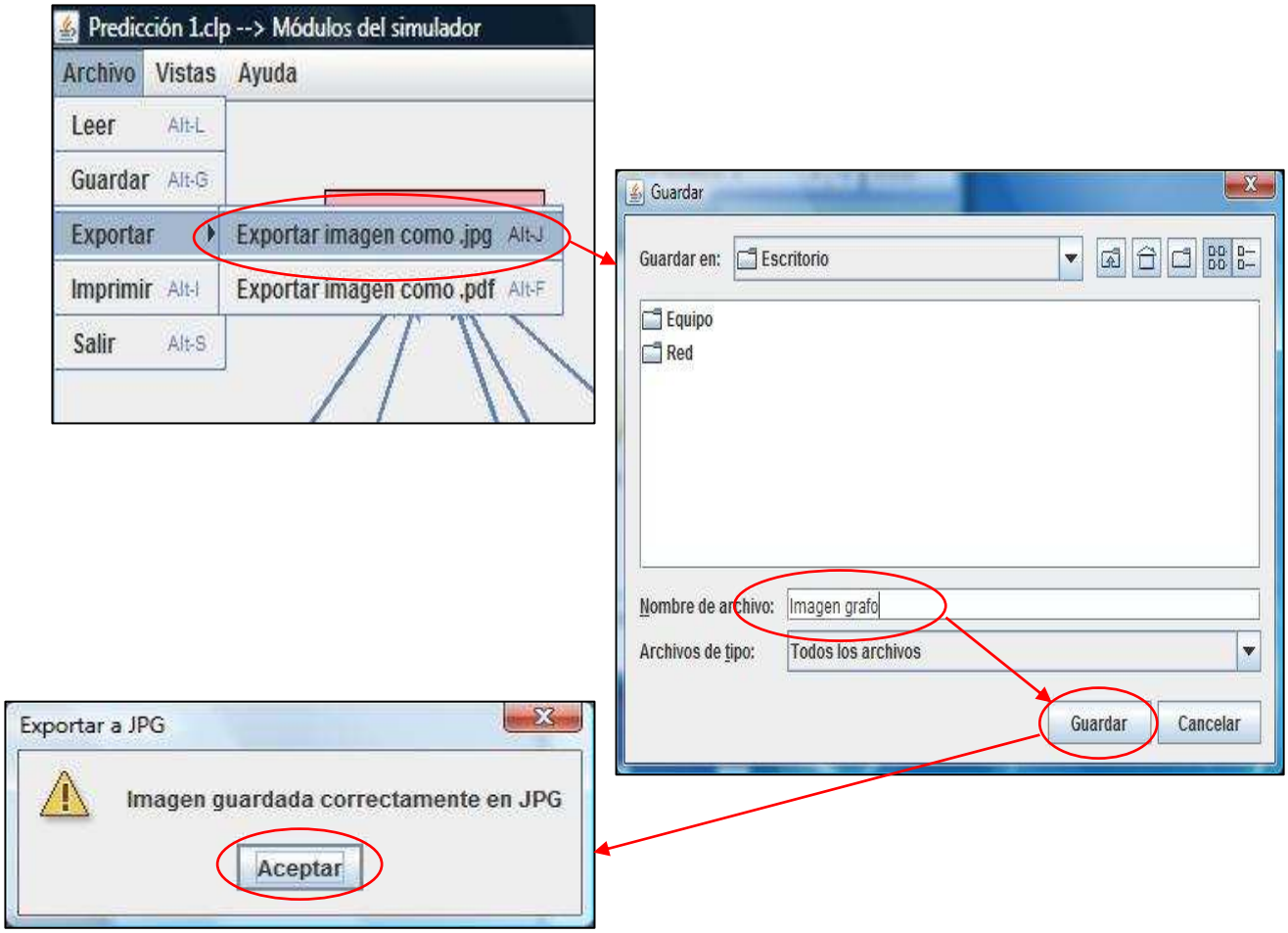
Todos los nodos que aparecen en pantalla pueden ser cambiados de sitio, de tamaño o colocarlos como se convenga en la extensión de la ventana. Con el resto de vistas funcionaría de la misma manera.

5.2.3- Exportar Imagen

Hay dos formas de exportar una imagen, en formato *.jpg* o en *.pdf*. El acceso se encuentra en el menú "*Archivo*" ➔ "Exportar":

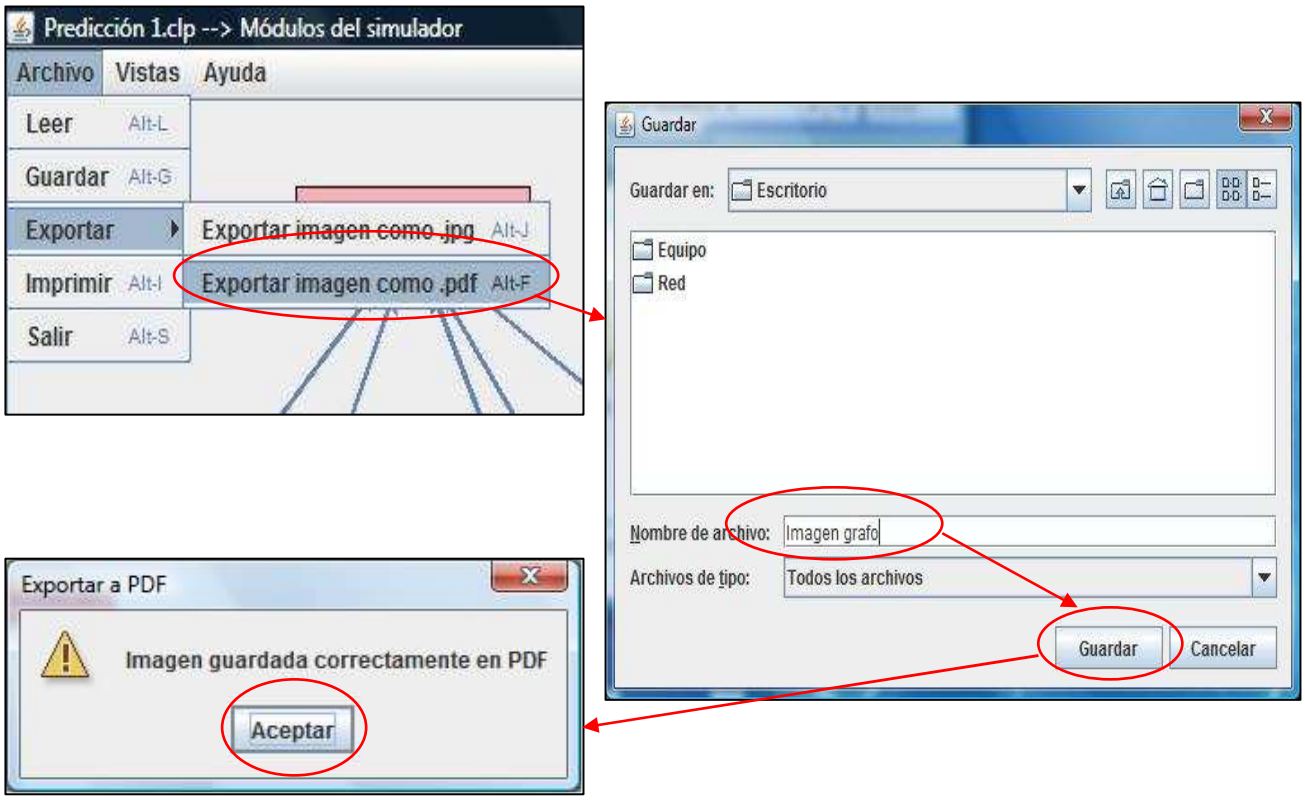
5.2.3.1- Exportar imagen a .jpg

Para realizar la captura de una imagen y guardarla en formato *.jpg* hay que dar a "*Archivo*" ➔ "*Exportar*" ➔ "*Exportar imagen como .jpg*" y desde el explorador de carpetas guardamos el archivo donde se convenga:



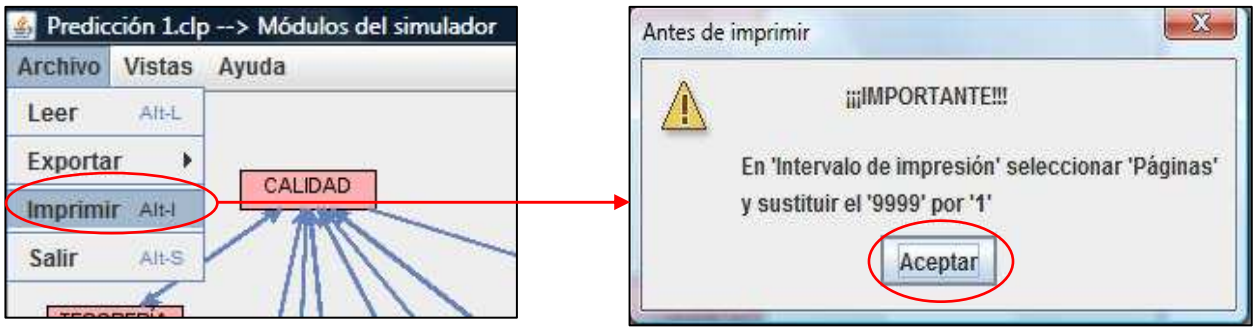
5.2.3.2- Exportar imagen a .pdf

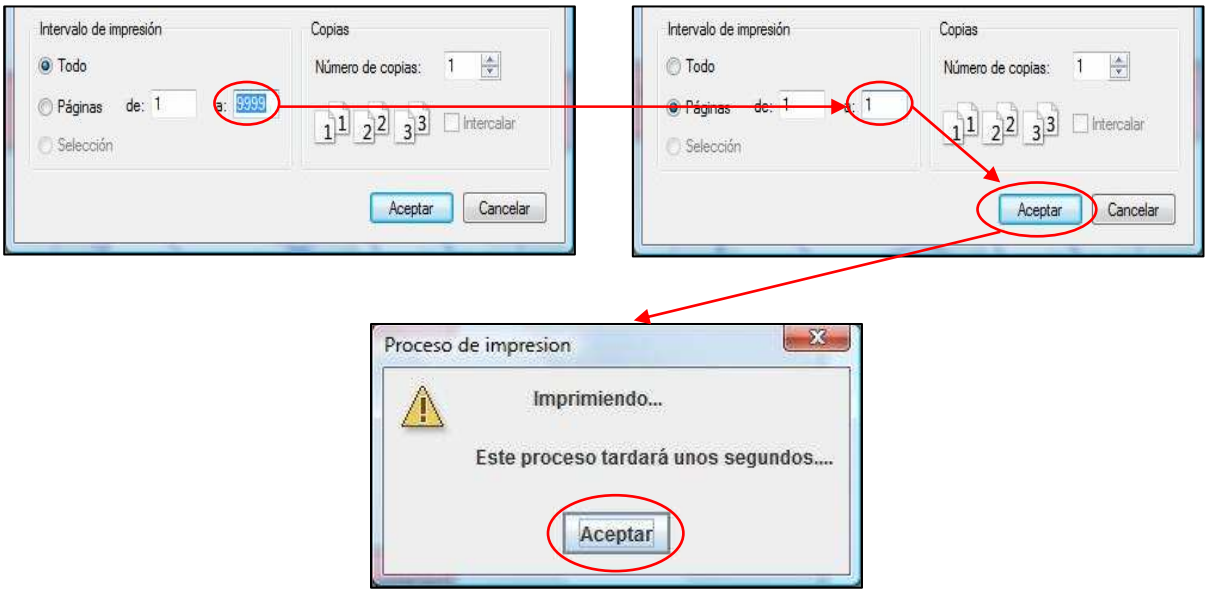
Para realizar la captura de una imagen y guardarla en formato *.pdf* hay que dar a "*Archivo*" ➔ "*Exportar*" ➔ "*Exportar imagen como .pdf*" y desde el explorador de carpetas guardamos el archivo donde se convenga:



5.2.4- Imprimir imagen

Para imprimir en papel la captura de la imagen hay que dar a "*Archivo*" ➔ "*Imprimir*". Aparecerá un mensaje de aviso sobre como configurar las opciones de impresión para que no haya problemas. A continuación se puede ver el proceso:



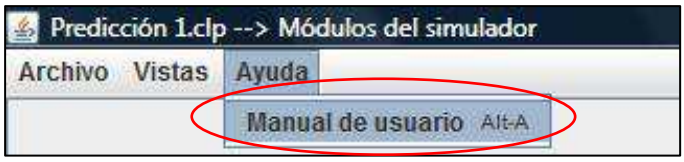


5.2.5- Salir del programa

Para salir del programa hay que dar a "*Archivo*" ➔ "*Salir*", y se cerrarán todas las ventanas que haya abiertas de INVERVISIÓN. Si se quiere cerrar solo una ventana cuando hay varias abiertas hay que dar a "X" en la esquina superior derecha.

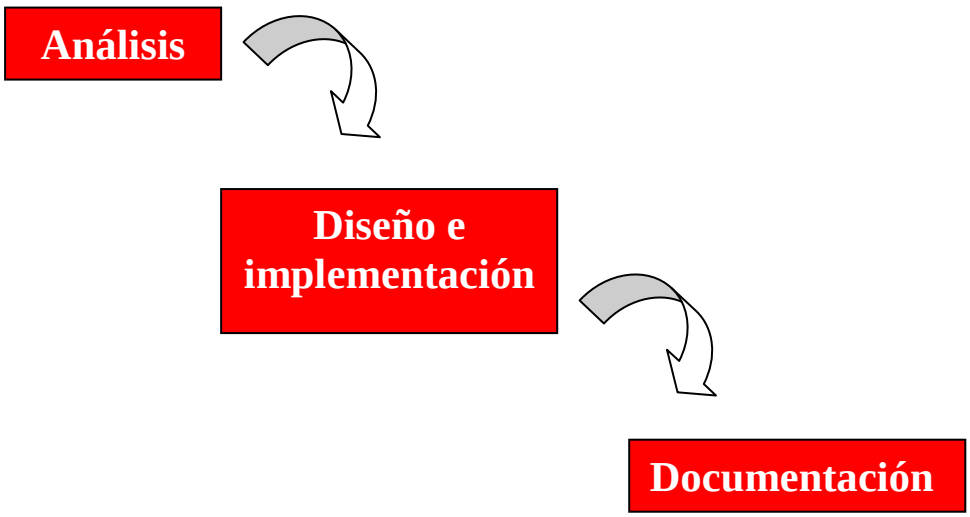
5.2.6- Ayuda

Para acceder a la "guía de uso" de INVERVISIÓN hay que dar a "*Ayuda*" ➔ "*Manual de usuario*" con lo que se abrirá un documento con formato *.pdf* con este manual.



5.3- Gestión del proyecto

En este apartado se va a mostrar cómo ha sido la “planificación del proyecto” desde su inicio hasta su fin. El “*ciclo de vida*” que se ha seguido en el proyecto ha sido el “modelo en cascada”, que implica que el inicio de la tarea siguiente debe esperar siempre a que termine la anterior. En el siguiente esquema se muestra como se ha estructurado en este caso:



Este modelo se ha planteado realizando bastantes pruebas antes de pasar a la siguiente fase, que en muchas ocasiones implicaba rediseñar la estructura de algunos de los módulos, por lo que el tiempo de desarrollo también se veía afectado.

5.3.1- Planificación del proyecto

El proyecto se inició a principios de octubre del año 2009, y concluyó a mediados de julio de 2012. En la *Figura 5.3.1.1* se muestra la "*planificación de tareas*" desglosada que se ha llevado a cabo:

	Nombre	Duración	Inicio	Terminado	Predecesores
1	**Análisis**	88,875 days	**6/10/09**	**5/02/10**	
2	Comprensión motor CLIPS	20 days	6/10/09	2/11/09	
3	Adaptación a JAVA	30 days	3/11/09	14/12/09	2
4	Recopilación de requisitos	38,875 days	15/12/09	5/02/10	3
5	**Diseño e implementación**	405 days	**8/02/10**	**26/08/11**	**1**
6	Creación estructura almacenamiento de reglas y atributos	45 days	8/02/10	9/04/10	
7	Extracción datos del motor CLIPS	105 days	12/04/10	3/09/10	6
8	Creación de grafos	155 days	6/09/10	8/04/11	7
9	Creación interfaz gráfica	100 days	11/04/11	26/08/11	8
10	**Documentación**	223,5 days	**5/09/11**	**12/07/12**	**5**
11	1ª versión de la memoria	74,875 days	5/09/11	19/12/11	
12	2ª versión de la memoria	139,75 days	19/12/11	2/07/12	11
13	Versión definitiva de la memoria	8,875 days	2/07/12	12/07/12	12

Figura 5.3.1.1

En cada una de las fases se ha invertido bastante tiempo, sobre todo en la fase de "diseño e implementación" donde se tenía que construir la aplicación INVERVISIÓN, que añadido a que era mi primer contacto con el lenguaje de programación JAVA, fue complicado implementar el intérprete para el "*motor de reglas CLIPS"* según las necesidades que implicaba.

Ha sido un proceso lento, que se ha ido desarrollando poco a poco, intentando minimizar los errores sin importar la duración del proyecto. A continuación, en la *Figura 5.3.1.2* se muestra el "*Diagrama de Gantt*" del proyecto junto a *la tabla de planificación de tareas* que identifica cada módulo en el calendario:

Nombre
Análisis
Comprensión motor CLIPS
Adaptación a JAVA
Recopilación de requisitos
Diseño e implementación
Creación estructura almacenamiento de reglas y atributos
Extracción datos del motor CLIPS
Creación de grafos
Creación interfaz gráfica
Documentación
1ª versión de la memoria
2ª versión de la memoria
Versión definitiva de la memoria
2010
2011
2012
2013
Q4
Q1
Q2
Q3
Q4
Q1
Q2
Q3
Q4
Q1
Q2

Figura 5.3.1.2

5.3.2- Análisis de costes

Respecto a los *recursos* utilizados hay que diferenciar entre los "*recursos humanos*" y los "*recursos materiales*".

Debido a que este proyecto ha sido desarrollado como trabajo de fin de carrera por una única persona regida por un tutor de la universidad, solo se ha contado con dos *recursos humanos*, el *desarrollador* que ha hecho las veces de *analista, programador y documentalista* del proyecto, y el tutor que ha hecho de *jefe de proyecto*. En la *Tabla 5.3.2.1* se muestran los costes de ambos recursos en función de las horas trabajadas:

	Periodo	Días trabajados	Horas trabajadas/día	€/hora	Coste total (€)
Analista/ programador/ Documentalista	Del 5 de octubre de 2009 al 26 de febrero de 2010	98	8	10	7840
	Del 1 de marzo de 2010 al 4 de febrero de 2011	234	4	10	9360
	Del 7 de febrero de 2011 al 31 de octubre de 2011	182	8	10	14560
	Del 2 de noviembre de 2011 al 12 de julio de 2012	171	4	10	6840
Jefe de Proyecto	Del 5 de octubre de 2009 al 12 de julio de 2012 (Excluyendo los meses de julio y agosto de cada año, y teniendo una reunión de 2 horas al mes)	30	2	30	1800
TOTAL					**40400**

Tabla 5.3.2.1

En cuanto a los *recursos materiales* se han precisado de varias herramientas necesarias para el desarrollo de este proyecto, que se van a detallar a continuación:

- Como soporte *hardware* se ha contado con un ordenador portátil *TOSHIBA SATELLITE L40 – 12K*, con el que se ha realizado todo el desarrollo.

- A nivel *software*, desde el sistema operativo "*Windows VISTA*", se ha contado con varios programas. Para el desarrollo del programa en JAVA se ha utilizado el *entorno de desarrollo libre "Netbeans IDE 6.7.1*". Luego, para el desarrollo de la memoria se ha utilizado el programa "*Microsoft Office Word 2003*". Y por último para la realización de la planificación de tareas del proyecto, se ha utilizado el *software libre de administración de proyectos "OpenProj*" y el "*Microsoft Office Excel 2003*" para la elaboración de las tablas de costes.

- Un recurso muy importante también ha sido la "*conexión a Internet*" que se ha tenido siempre activa mientras se ha desarrollado el proyecto, y que ha sido muy útil cuando se ha tenido que utilizar para solventar cualquier problema. También ha sido fundamental para estar siempre en contacto con el tutor del proyecto.

- Luego, también se ha hecho bastante uso de *material de papelería* como apoyo físico, para hacer anotaciones o esquemas que ayudaran a comprender muchas de las partes de este proyecto.

A continuación se muestra la tabla de costes de los *recursos* materiales en *Tabla 5.3.2.2*, con la que se va a dar por finalizado el *análisis de costes*:

	Precio (€)
ordenador portátil *TOSHIBA SATELLITE L40 – 12K + Windows VISTA*	600
Netbeans IDE 6.7.1	0
Microsoft Office 2003	139
OpenProj	0
Conexión a Internet	48 € x 34 meses = 1632
material de papelería	10
TOTAL	**2381**

Tabla 5.3.2.2