Universidad de Granada

Grado en Ingeniería Informática.

Práctica final.

Nombre de la asignatura:
Ingeniería del Conocimiento.

Realizado por:
Néstor Rodríguez Vico

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE TELECOMUNICACIÓN.

Granada, 22 de junio de 2017.

Índice

1. Resumen de cómo funciona el sistema experto.

En esta práctica se ha desarrollado un sistema experto con 3 módulos:

- Módulo 1: En este módulo se va a permitir al usuario elegir la licencia que mejor se adecue a sus necesidades preguntándole por que valor le interesa que tengan ciertas características de las licencias.
- Módulo 2: En este módulo se va a permitir al usuario comprobar la compatibilidad de dos licencias. Concretamente, está enfocado a comprobar si la licencia que el usuario quiere ponerle a su código es compatible con la licencia que tiene otro código usado por el usuario en su propio código.
- Módulo 3: En este módulo la idea es aconsejar al usuario sobre como debe tratar los datos de carácter sensible que puede almacenar en sus empresa y organización.

2. Descripción del proceso seguido para el desarrollo.

2.1. Procedimiento seguido para obtener el conocimiento.

La manera en la que se ha obtenido el conocimiento varía en cada módulo. En el primero de ellos se ha usado una rejilla de repertorio [1] con las 7 licencias tratadas en nuestro sistema experto. Para obtener la información de las licencias he buscado por Internet las características de cada una de ellas para poder crear la rejilla de repertorio. En el segundo módulo se ha usado un árbol de decisión para poder determinar las características que permiten decidir si dos licencias son compatibles o no. Para ver si dos licencias son compatibles o no lo que he hecho ha sido buscar información en Internet. Finalmente, para el tercer módulo el conocimiento ha sido extraído de la Ley Orgánica de Protección de Datos.

2.2. Procedimiento de validación y verificación del sistema seguido.

El proceso de validación seguido ha sido el que hemos visto en clase. Se ha interactuado con el sistema preguntando por todas las licencias posibles del módulo 1, se ha preguntado por la compatibilidad de todas las licencias del módulo 2 y se han comprobado varios casos para el módulo 3. Para todas las pruebas se ha comprobado que el sistema da una respuesta correcta y que argumenta el porqué de la misma. Finalmente, he dejado a varias personas interactuar con el sistema para obtener opiniones y corregir los errores del mismo, como han podido ser ciertas respuesta que eran poco "humanas".

[1] La rejilla re repertorio no se ha entrgado porque un día en clase me dijo que no era necesario. En el caso de querer verla, avíseme y se la mando por correo.

3. Descripción del sistema desarrollado.

3.1. Variables de entrada del problema.

Las únicas variables de entrada que se han usado han sido las proporcionadas por el usuario en los distintos módulos. En cada módulo los primeros pasos son de interacción con el usuario para poder recopilar la información necesaria para el correcto funcionamiento de cada módulo. El funcionamiento del primer módulo depende de como vaya respondiendo el usuario a las distintas preguntas que le llevarán a la elección de una licencia final. El funcionamiento del segundo varía según las dos licencias que se quieren comparar y el funcionamiento del tercero depende de los datos que vaya a usar la organización del usuario, del tipo de organización y de donde estén almacenados los datos.

Otra variable de entrada usada en nuestro sistema experto son los documentos ARCO para el módulo 3. Estos documentos son leídos para ser rellenados posteriormente. [2]

3.2. Variables de salida del problema.

Como variables de salida tenemos la licencia elegida en el módulo 1, si son compatibles o no las licencias elegidas en el módulo 2 o las medidas que se deben implantar y los documentos necesarios en el módulo 3.

3.3. Conocimiento global del sistema.

Como conocimiento global del sistema tenemos varios grupos:

- El primer conocimiento que tiene nuestro sistema experto son las licencias que va a manejar. Estas licencias son comunes para el módulo 1 y 2. Para representar las licencias se ha usado una plantilla con la siguiente estructura:

```
(deftemplate Licencia
    (multifield Nombre)
    (field Enlazar)
    (field Distribucion)
    (field Modificacion)
    (field Patentes)
    (field UsoPrivado)
    (field Sublicencia)
    (field Marca)
    (field GPL3)
)
```

- La comunicación con el usuario, para facilitar la tarea, se va a realizar usando números. Es decir, cuando el usuario tenga que elegir entre una lista de opciones, si quiere elegir la opción 1 de la lista pulsará el número 1 en vez de tener que

[2] Los documentos ARCO han sido entregados en el fichero .zip de la práctica.

escribir dicha opción. Pero, para luego mostrar una información más completa por pantalla y no mostrar números, nuestro sistema tiene conocimiento que permite inferir que significa un número en ciertas circunstancias. Por ejemplo, si estamos hablando sobre el valor de la característica *Distribución* y el usuario al preguntarle por la misma introdujo un 4, mediante el hecho *(CambiarDistribucion 4 Copyleft)* sabemos que ese cuatro representa que el valor de *Distribución* es *Copyleft*. Este tipo de conocimiento se ha definido para todas las características y para todos los posibles valores de las mismas por lo comentado anteriormente, si al usuario le indicamos por pantalla *Copyleft* en vez del valor *4*, la respuesta será mucho más cercana y entendible.

- También hay un hecho que nos permite saber que estamos al inicio de nuestro sistema experto. Este hecho es *(Modulo 0)*.

- Con respecto al módulo 3 tenemos hechos que nos permiten representar que tipos de datos personales va a manejar el sistema. Estos hechos tienen la forma *(tipo nombre)*, donde *tipo* es el tipo del dato almacenado y *nombre* es el nombre del dato almacenado. Por ejemplo, el hecho *(DCI DNI)* indica que *DNI* es un dato de tipo *DCI* (dato de carácter identificativo). Al igual que he comentado anteriormente, en este módulo también tenemos hechos que permiten cambiar el valor numérico introducido por el usuario por su valor real para facilitar la comprensión de las respuestas proporcionadas por nuestro sistema experto.

3.4. Especificación de los módulos se han desarrollado.

Los módulos que se han desarrollado son los comentados anteriormente.

- El primer módulo que permite al usuario elegir la licencia que más se adapta a sus necesidades. En este módulo el conocimiento empleado no es más que las propias licencias y los hechos comentados anteriormente que permiten "traducir" la información introducida por el usuario en información más legible. El conocimiento que se va deducir no es más que el introducido por el usuario al preguntarle por las características de su licencia. Con este conocimiento proporcionado por el usuario nuestro sistema experto es capaz de ir moviéndose por el árbol para obtener una respuesta.

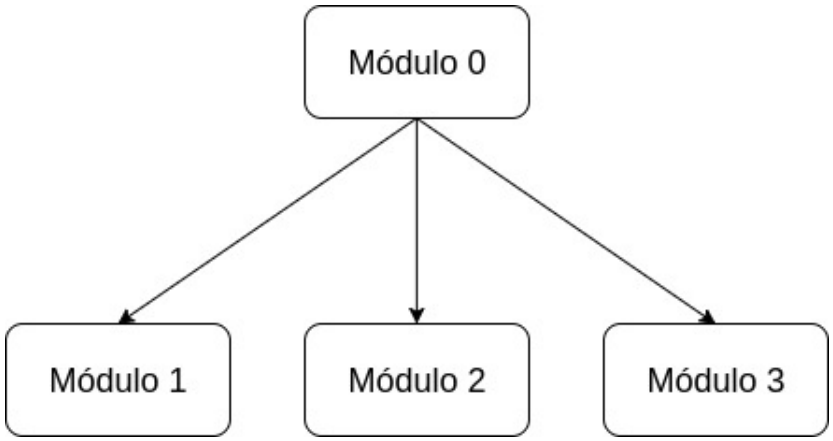
- Un segundo módulo que permite comprobar la compatibilidad de dos licencias. En este módulo el conocimiento empleado no es más que las propias licencias y los hechos comentados anteriormente que permiten "traducir" la información introducida por el usuario en información más legible. Nada más empezar el módulo se le solicita al usuario las dos licencias a comparar y el sistema experto se moverá por el árbol de decisión comprobando las características de las dos licencias para decidir si estas son compatibles o no.

- Un tercer módulo que informa al usuario de que debe hacer para cumplir con la Ley Orgánica de Protección de Datos. En este módulo el conocimiento empleado

es el adquirido de las páginas web proporcionadas por el profesor de la asignatura. También se adquiere conocimiento interactuando con el usuario y se deduce conocimiento, como puede ser si un dato permite identificar a un usuario o no y las medidas que el usuario del sistema experto debe implantar para cumplir con la Ley Orgánica de Protección de Datos.

3.5. Estructura de funcionamiento del esquema de razonamiento del sistema.

La estructura de funcionamiento de nuestro sistema experto es bien sencillo. Cuenta con 4 módulos. Un primer módulo 0 que sirve de bienvenida y que permite al usuario elegir con qué módulo desea interactuar y los tres módulos que ya se han comentado anteriormente. Una pequeña representación del esquema de funcionamiento es el siguiente:



3.6. La lista de hechos que utiliza el sistema durante su funcionamiento y la forma de representarlos.

Hay muchos hechos cuya estructura se repite, así que voy a comentar los más relevantes. Para saber en que módulo estamos tenemos un hecho que nos lo indica y el cual vamos asertando o retractándonos para ir moviéndonos por los módulos. Este hecho es *(Módulo i)* donde i indica el número del módulo. Tanto para el módulo 1 y para el módulo 2 tenemos hechos que representan las licencias de acuerdo al *template* comentado anteriormente. Un ejemplo de una licencia es:

```
(Licencia
    (Nombre Apache)
    (Enlazar Permisivo)
    (Distribucion Permisivo)
    (Modificacion Permisivo)
    (Patentes Si)
    (UsoPrivado Si)
    (Sublicencia Permisivo)
    (Marca No)
```

```
    (GPL3 Si)
)
```

También tenemos hechos para cambiar la información, como se ha comentado anteriormente. Un ejemplo puede ser: *((CambiarEnlazar 1 Permisivo))*, el cual es usado para cambiar el valor *1* de la característica *Enlazar* por su valor real, *Permisivo*.

Para el módulo 3 tenemos representado mediante hechos los datos que pueden ser utilizados por un usuario en su organización. Los datos se representan de la siguiente manera: *(tipo nombre)*, donde *tipo* es el tipo del dato almacenado y *nombre* es el nombre del dato almacenado. Por ejemplo, el hecho *(DEP Ideologia)* indica que *Ideologia* es un dato de tipo *DEP* (dato especialmente protegido). Al igual que en los módulos anteriores tenemos hechos para "traducir" la información. En este módulo tenemos también hechos para cambiar las siglas empleadas en los tipos de los datos. Por ejemplo, *((Cambiar Siglas DEP DatosEspecialmenteProtegidos))*.

En cuanto a los hechos deducidos del módulo 1 tenemos hechos que nos permiten ir guardando las características indicadas por el usuario. Por ejemplo, cuando se le pregunta por la característica *Enlazar*, el usuario introducirá un número que indicará el elemento elegido de la lista y se guardará un hecho de la siguiente forma: *(Enlazar número)*. De manera similar se hace para todas las características que tiene una licencia.

Para el módulo 2 no se deducen hechos, simplemente hay reglas que, según las precondiciones de las mismas, permiten estar en los distintos nodos de nuestro árbol de decisión para decidir la compatibilidad o no de las dos licencias.

Para el módulo 3 si tenemos varios hechos que se introducen en nuestra base de conocimiento a lo largo del funcionamiento del mismo. Los más evidentes son los que se le preguntan al propio usuario en cuanto a los datos que va a usar. Por ejemplo, cuando se le pregunte sobre que datos, por ejemplo, especialmente protegido que va a usar, se añadirá un hecho con la siguiente estructura: *(Usuario DEP número)*. De esta misma manera se hará para todos los datos disponibles. Una vez tenemos los datos recogidos, hay dos reglas para procesarlos todos y decidir si estos pueden identificar a un usuario o no. La primera de ellas guarda automáticamente los datos que sabemos que si permiten identificar a una persona. Para ello añade a nuestra base de conocimiento un hecho de la forma *(IdentificablePor tipoDato dato)* el cual indica que el usuario es identificable por *dato* que es de tipo *tipoDato*. La segunda regla permite al usuario decidir si un dato puede identificar al usuario y añade un hecho de la forma *(PuedeSerIdentificablePor tipoDato dato numero)*. Si *número* es un 1, querrá decir que el usuario es identificable por *dato* que es de tipo *tipoDato*. Posteriormente, todos los hechos *PuedeSerIdentificablePor* que tienen un 1 en el valor *numero* son procesados para añadir el hecho *(IdentificablePor tipoDato dato)* correspondiente.

3.7. Los hechos y las reglas de cada módulo.

Los hechos de cada modulo ya han sido comentados anteriormente.

3.8. Reglas del módulo 1.

En el módulo 1 las reglas son más o menos iguales. Desde mi punto de vista podemos identificar dos reglas distintas así que vamos a ver un ejemplo de cada una de ellas:

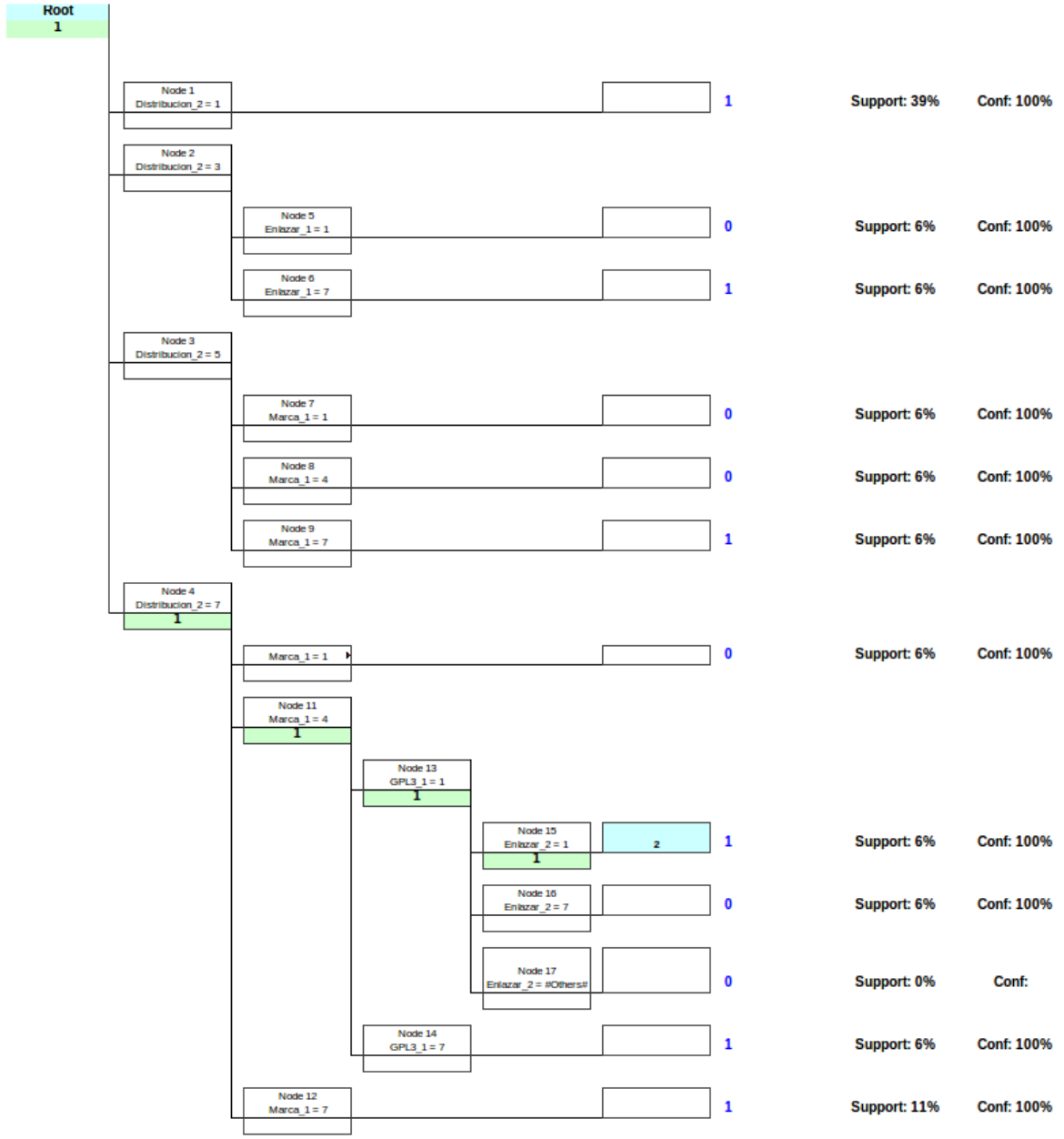
```
(defrule LicenciaUno
    ?m <- (Modulo 1)
    (Sublicencia 2)
    (CambiarSublicencia 2 ?tipo)

    (Licencia (Nombre $?nombre) (Sublicencia ?tipo))
    =>
    (printout t "Como quieres que tu licencia tenga una sublicencia " ?
       tipo " tu licencia correcta seria " $?nombre crlf)
    (retract ?m)
)

(defrule LicenciaDos
    (Modulo 1)
    (Sublicencia ?sub)
    (test (neq ?sub 2))
    (CambiarSublicencia ?sub ?tipo)
    =>
    (printout t "Quieres que tu licencia tenga una sublicencia " ?tipo ".
       Si va a enlazar el codigo, como le interesa? Pulse 1 para Permisivo
       , pusle 2 para Dominio Publico pulse 3 para Limitado o pulse 4 para
        solo enlazarlo con GNU-GPL v3: ")
    (assert (Enlazar (read)))
)
```

La primera de ellas representa la estructura general de una regla que da una licencia como respuesta. La estructura de estás reglas tiene como precondición los hechos que definen las características que caen en el nodo hoja asociado a dicha regla. Una vez tenemos las características de las licencias, podemos buscar en nuestro conocimiento una licencia que se adecue a las necesidades para obtener el nombre de la misma para poder dar una respuesta al usuario. Otra característica de este tipo de reglas es que se retractan del hecho *Modulo 1*. Esto se hace para indicar que hemos terminado de trabajar en el módulo 1.

La segunda representa una regla asociada a un nodo intermedio. Es una regla que tiene como precondición los hechos de los niveles superiores y que formula una pregunta para añadir un nuevo hecho a nuestra base de conocimiento para poder seguir desplazándonos por nuestro árbol hasta llegar a un nodo hoja.

3.9. Reglas del módulo 2.

En el módulo 2 las reglas son más o menos iguales. El árbol implementado es el siguiente:



Desde mi punto de vista podemos identificar dos reglas distintas así que vamos a ver un ejemplo de cada una de ellas:

```
(defrule ComprobarDistribucionUno
    ?m <- (Modulo 2)
    (LicenciaMiCodigo ?lMia)
```

```
    (LicenciaCodigoUsado ?lUsado)
    (CambiarNombreLicencia ?lMia $?miLicencia)
    (CambiarNombreLicencia ?lUsado $?usoLicencia)

    (Licencia (Nombre $?usoLicencia) (Distribucion ?d))

    (CambiarDistribucionModulo2 ?numeroDistribucion ?d)

    (test (eq ?numeroDistribucion 1))
    =>
    (printout t "Las licencias son compatibles porque " ?usoLicencia "
       tiene una distribucion " ?d "." crlf)
    (retract ?m)
)

(defrule ComprobarDistribucionDos
    (Modulo 2)
    (LicenciaMiCodigo ?lMia)
    (LicenciaCodigoUsado ?lUsado)
    (CambiarNombreLicencia ?lMia $?miLicencia)
    (CambiarNombreLicencia ?lUsado $?usoLicencia)

    (Licencia (Nombre $?usoLicencia) (Distribucion ?d))

    (CambiarDistribucionModulo2 ?numeroDistribucion ?d)

    (test (neq ?numeroDistribucion 1))
    =>
    (assert (SegundoNivelModulo2))
)
```

La idea es más o menos igual que en el módulo 1, ya que en ambos módulos estamos implementando un árbol en CLIPS. A diferencia de las reglas del módulo 1, en este módulo no vamos insertando respuestas del usuario, sino que vamos insertando hechos que nos permiten indicar en que nivel del árbol estamos.

3.10. Reglas del módulo 3.

En el módulo 3 es donde tenemos más variedad de reglas, ya que hay más trabajo que realizar. Primero tenemos reglas que permiten al usuario introducir los datos que su organización va a usar. Estás reglas tienen la siguiente estructura:

```
(defrule LeerDEP
    (declare (salience -1))
    (Modulo 3)
    (Usuario DEP ?elemento)
    (test (neq ?elemento 0))
    =>
    (printout t "DatosEspecialmenteProtegidos: 1 para ideologia, 2 para
       afiliacion sindical, 3 para religion, 4 para creencias, 5 para
       origen racial o etnico, 6 para salud y vida sexual: ")
```

```
    (assert (Usuario DEP (read)))
)
```

La idea es una regla que se ejecutará en bucle hasta que el usuario introduzca un 0, es decir, que no quiera introducir más datos. El uso de la prioridad es para que se pidan primero todos los datos de tipo *DEP* antes de empezar a pedir los datos de otro tipo.

Una vez tenemos los datos recogidos, hay dos reglas para imprimir los datos para que el usuario vea cuales son los que va a utilizar. Estas reglas son la siguientes:

```
(defrule ImprimirDatos
    (Modulo 3)
    (Usuario DTBS 0)
    =>
    (printout t "Los datos que vas a usar son los siguientes: " crlf)
)

(defrule ImprimirDatos2
    (declare (salience -1))
    (Modulo 3)
    (Usuario DTBS 0)
    (Usuario ?tipo ?dato)
    (Cambiar ?tipo ?dato ?datoCambiado)
    (Cambiar Siglas ?tipo ?tipoTexto)
    =>
    (printout t ?tipoTexto " " ?datoCambiado crlf)
)
```

La idea es simple, recorrer todos los hechos que representan un dato introducido por el usuario e imprimir dicho dato junto con su tipo (cambiando las siglas por su texto correspondiente).

A continuación tenemos 3 reglas que permiten al usuario introducir información acerca de la organización que va a usar los datos. Dichas reglas son las siguientes:

```
(defrule TipoOrganizacion
    (declare (salience -2))
    (Modulo 3)
    (Usuario DTBS 0)
    =>
    (printout t "Que tipo de organizacion va a usar los datos? Pulsa 1
       para usuario domestico, 2 para empresa privada o pulsa 3 para
       organizaciones dependientes de las administraciones publicas: ")
    (assert (TipoOrganizacion (read)))
)

(defrule UsoDatos
    (declare (salience -3))
    (Modulo 3)
    (Usuario DTBS 0)
    =>
```

```
    (printout t "Que tipo de uso se le van a dar a los datos? Pulsa 1 para
        una utilizacion general de bases de datos (que incluye sector
       privado y publico) o pulsa 2 para bases de datos para investigacion
        judicial y policial: ")
    (assert (UsoDatos (read)))
)

(defrule UbicacionDatos
    (declare (salience -4))
    (Modulo 3)
    (Usuario DTBS 0)
    =>
    (printout t "Donde van a estar almacenados los datos? Pulsa 1 para
       Espana, pulsa 2 para Europa y pulsa 3 para fuera de Europa: ")
    (assert (UbicacionDatos (read)))
)
```

Las reglas son autoexplicativas. La primera de ellas permite indicar al usuario el tipo de organización que va a usar los datos, la segunda el uso que se le va a dar a los datos y la tercer la ubicación de los mismos.

Una vez tenemos toda la información recogida, debemos procesar el conocimiento que nos ha aportado el usuario. Para ello tenemos 3 reglas:

- La primera de ellas nos permite decidir que datos permiten identificar a una persona de forma directa, y por lo tanto los almacenamos en un hecho que su primera característica es *IdentificablePor*.
- La segunda regla está pensada para detectar datos que permiten identificar a una persona en determinadas circunstancias. Por ejemplo, el sexo de una persona no es identificativo de por sí, pero si estamos en una situación en la cual hay cien persona, 99 son de un sexo y 1 de otro, el sexo pasa a ser un dato que permite identificar a una persona. Por eso es necesario que el usuario nos indique si esta situación puede darse o no.
- Finalmente, la tercera regla se encarga de añadir a nuestra base de conocimiento los datos que el usuario ha indicado que si pueden identificar a una persona en la situación en la que se van a usar los mismos. Para ello simplemente debe recorrer todos los hechos que empiezan por *PuedeSerIdentificablePor* que tienen un 1 al final (que representa una respuesta positiva) y añadir el hecho correspondiente.

```
(defrule IdentificacionSegura
    (UsoDatos ?n)
    (UbicacionDatos ?u)

    (Usuario ?tipoDato ?introducido)
    (test (neq ?introducido 0))
    (test (neq ?introducido *))
    (Cambiar ?tipoDato ?introducido ?datoCambiado)
```

```
    (test (or (eq ?datoCambiado DNI) (eq ?datoCambiado Direccion) (eq ?
       datoCambiado Imagen) (eq ?datoCambiado Voz)
              (eq ?datoCambiado NSeguridadSocial) (eq ?datoCambiado
                 Telefono) (eq ?datoCambiado NombreApellidos)
              (eq ?datoCambiado FirmaHuella) (eq ?datoCambiado
                 FirmaElectronica) (eq ?datoCambiado TarjetaSanitaria)

              (eq ?datoCambiado CreacionesArtisticas) (eq ?datoCambiado
                 Literarias) (eq ?datoCambiado CientificasTecnicas)

              (eq ?datoCambiado DatosBancarios) (eq ?datoCambiado
                 DatosEconomicosNomina) (eq ?datoCambiado TarjetasCredito)
        )
    )
    =>
    (printout t "El usuario puede ser identificado por usar el dato: " ?
       datoCambiado crlf)
    (assert (IdentificablePor ?tipoDato ?datoCambiado))
)

(defrule PosibleIdentificacion
    (declare (salience -1))
    (UsoDatos ?n)
    (UbicacionDatos ?u)

    (Usuario ?tipoDato ?introducido)
    (test (neq ?introducido 0))
    (test (neq ?introducido *))
    (Cambiar ?tipoDato ?introducido ?datoCambiado)

    (test (and (neq ?datoCambiado DNI) (neq ?datoCambiado Direccion) (neq
       ?datoCambiado Imagen) (neq ?datoCambiado Voz)
              (neq ?datoCambiado NSeguridadSocial) (neq ?datoCambiado
                 Telefono) (neq ?datoCambiado NombreApellidos)
              (neq ?datoCambiado FirmaHuella) (neq ?datoCambiado
                 FirmaElectronica) (neq ?datoCambiado TarjetaSanitaria)

              (neq ?datoCambiado CreacionesArtisticas) (neq ?datoCambiado
                  Literarias) (neq ?datoCambiado CientificasTecnicas)

              (neq ?datoCambiado DatosBancarios) (neq ?datoCambiado
                 DatosEconomicosNomina) (neq ?datoCambiado
                 TarjetasCredito)
        )
    )
    =>
    (printout t "En la circunstancia en la que vas a usar los datos puede
       darse el caso de que una persona pueda ser identificada por " ?
       datoCambiado "? Pulsa 1 para si y 2 para no: ")
    (assert (PuedeSerIdentificablePor ?tipoDato ?datoCambiado (read)))
)
```

```
(defrule ProcesarPosiblesIdentificaciones
    (PuedeSerIdentificablePor ?tipoDato ?datoCambiado 1)
    =>
    (assert (IdentificablePor ?tipoDato ?datoCambiado))
)
```

Una vez tenemos recopilados los datos que va a usar el usuario, debemos informar de que debe hacer para cumplir con la LOPD. Para ello tenemos una regla que detectará si el usuario va a usar algún dato personal y le dará unas recomendaciones de que debe hacer y que no. La regla es la siguiente:

```
(defrule ResumenMedidas
        (declare (salience -10))
        (Modulo 3)
        (Usuario ?a ?b)
        =>
        (printout t "Para cumplir con el articulo 4 de la LOPD, el
            empresario debe recoger los datos usando medios que no sean
            fraudulentos, solo puede recoger unicamente aquellos datos
            personales que sean adecuados ... )
)
```

Finalmente, vamos a rellenar los formularios ARCO. Para ello, primero leemos de unos ficheros de texto la plantilla proporcionada por la página web indicada por el profesor. Para ello usamos las siguientes reglas:

```
(defrule AbrirAcceso
    (Modulo 3)
    =>
    (open "Acceso" mi_fichero)
    (assert (SeguirLeyendoAcceso))
    (assert (Fichero (Nombre Acceso) (Texto *)))
)

(defrule LeerAcceso
    (Modulo 3)
    ?f <- (SeguirLeyendoAcceso)
    ?d <- (Fichero (Nombre Acceso) (Texto $?texto))
    =>
    (bind ?leido (readline mi_fichero))
    (retract ?f)
    (retract ?d)

    (if (neq ?leido EOF) then
        (assert (Fichero (Nombre Acceso) (Texto $?texto ?leido)))
        (assert (SeguirLeyendoAcceso))
    )

    (if (eq ?leido EOF) then
        (close mi_fichero)
        (assert (Fichero (Nombre Acceso) (Texto $?texto)))
        (assert (FicheroAcceso Listo))
```

```
    )
)
```

La primera de ellas se encarga de abrir el fichero de nombre *acceso* y la segunda de ellas de leerlo. La idea es añadir a nuestra base de conocimiento un hecho *Fichero* con el nombre del fichero que estamos procesando y el contenido del mismo, para procesarlo a continuación.

Una vez tenemos la plantilla lista, debemos pedirle al usuario que nos indique cuales son los datos con los que debe rellenar el formulario. Para ello, hemos creado un template con la siguiente estructura:

```
(deftemplate RespuestaDatos
    (field Identificador)
    (field DatoRecogido)
    (multifield Respuesta)
)
```

Y tenemos una regla que pregunta al usuario lo necesario y va añadiendo hechos a nuestro conocimiento para luego poder rellenar la plantilla ya leída.

```
(defrule PedirDatos
    (declare (salience -20))
    (Modulo 3)
    (UsoDatos ?n)
    (UbicacionDatos ?u)

    (FicheroAcceso Listo)
    (FicheroCancelacion Listo)
    (FicheroOposicion Listo)
    (FicheroRectificacion Listo)
    =>
    (printout t "Voy a preguntarte por algunos datos que necesito para
       rellenar los fomularios ARCO. Primero sobre el responsable del
       fichero. Cual es su nombre? ")
    (assert (RespuestaDatos (Identificador Responsable) (DatoRecogido
       Nombre) (Respuesta (readline))))
    (printout t "En que calle vive? ")
    (assert (RespuestaDatos (Identificador Responsable) (DatoRecogido
       Direccion) (Respuesta (readline))))
    (printout t "En que numero vive? ")
    (assert (RespuestaDatos (Identificador Responsable) (DatoRecogido
       Numero) (Respuesta (readline))))
    (printout t "Cual es su codigo postal? ")
    (assert (RespuestaDatos (Identificador Responsable) (DatoRecogido CP)
       (Respuesta (readline))))
    (printout t "En que localidad vive? ")
    (assert (RespuestaDatos (Identificador Responsable) (DatoRecogido
       Localidad) (Respuesta (readline))))
    (printout t "En que provincia vive? ")
    (assert (RespuestaDatos (Identificador Responsable) (DatoRecogido
       Provincia) (Respuesta (readline))))
```

```
    (printout t "En que comunidad autonoma vive? ")
    (assert (RespuestaDatos (Identificador Responsable) (DatoRecogido
       CAutonoma) (Respuesta (readline))))
    (printout t "Cual es su DNI? ")
    (assert (RespuestaDatos (Identificador Responsable) (DatoRecogido DNI)
        (Respuesta (readline))))

    (printout t "Ahora sobre el interesado o el representante legal. Cual
       es su nombre? ")
    (assert (RespuestaDatos (Identificador Interesado) (DatoRecogido
       Nombre) (Respuesta (readline))))
    (printout t "En que calle vive? ")
    (assert (RespuestaDatos (Identificador Interesado) (DatoRecogido
       Direccion) (Respuesta (readline))))
    (printout t "En que numero vive? ")
    (assert (RespuestaDatos (Identificador Interesado) (DatoRecogido
       Numero) (Respuesta (readline))))
    (printout t "En que localidad vive? ")
    (assert (RespuestaDatos (Identificador Interesado) (DatoRecogido
       Localidad) (Respuesta (readline))))
    (printout t "En que provincia vive? ")
    (assert (RespuestaDatos (Identificador Interesado) (DatoRecogido
       Provincia) (Respuesta (readline))))
    (printout t "En que comunidad autonoma vive? ")
    (assert (RespuestaDatos (Identificador Interesado) (DatoRecogido
       CAutonoma) (Respuesta (readline))))
    (printout t "Cual es su codigo postal? ")
    (assert (RespuestaDatos (Identificador Interesado) (DatoRecogido CP) (
       Respuesta (readline))))
    (printout t "Cual es su DNI? ")
    (assert (RespuestaDatos (Identificador Interesado) (DatoRecogido DNI)
       (Respuesta (readline))))

    (printout t "En que ciudad se va a firmar el presente documento? ")
    (assert (RespuestaDatos (Identificador General) (DatoRecogido Ciudad)
       (Respuesta (readline))))
    (printout t "Que dia (numero) se va a firmar el presente documento? ")
    (assert (RespuestaDatos (Identificador General) (DatoRecogido Dia) (
       Respuesta (readline))))
    (printout t "Que mes se va a firmar el presente documento? ")
    (assert (RespuestaDatos (Identificador General) (DatoRecogido Mes) (
       Respuesta (readline))))
    (printout t "Que anio (solo las dos ultimas cifras) se va a firmar el
       presente documento? ")
    (assert (RespuestaDatos (Identificador General) (DatoRecogido Anio) (
       Respuesta (readline))))

    (assert (DatosPedidos))
)
```

Finalmente, tenemos una regla que permite procesar el fichero y añadir los datos introducidos por el usuario. La regla es la siguiente:

```
(defrule ProcesarDatos
    (Fichero (Nombre ?nombreFichero) (Texto ?asterisco ?texto))
    (DatosPedidos)

    (RespuestaDatos (Identificador Responsable) (DatoRecogido Nombre) (
        Respuesta ?Nombre))
    (RespuestaDatos (Identificador Responsable) (DatoRecogido Direccion) (
        Respuesta ?Direccion))
    (RespuestaDatos (Identificador Responsable) (DatoRecogido Numero) (
        Respuesta ?Numero))
    (RespuestaDatos (Identificador Responsable) (DatoRecogido CP) (
        Respuesta ?CP))
    (RespuestaDatos (Identificador Responsable) (DatoRecogido Localidad) (
        Respuesta ?Localidad))
    (RespuestaDatos (Identificador Responsable) (DatoRecogido Provincia) (
        Respuesta ?Provincia))
    (RespuestaDatos (Identificador Responsable) (DatoRecogido CAutonoma) (
        Respuesta ?CAutonoma))
    (RespuestaDatos (Identificador Responsable) (DatoRecogido DNI) (
        Respuesta ?DNI))

    (RespuestaDatos (Identificador Interesado) (DatoRecogido Nombre) (
        Respuesta ?NombreDos))
    (RespuestaDatos (Identificador Interesado) (DatoRecogido Direccion) (
        Respuesta ?DireccionDos))
    (RespuestaDatos (Identificador Interesado) (DatoRecogido Numero) (
        Respuesta ?NumeroDos))
    (RespuestaDatos (Identificador Interesado) (DatoRecogido Localidad) (
        Respuesta ?LocalidadDos))
    (RespuestaDatos (Identificador Interesado) (DatoRecogido Provincia) (
        Respuesta ?ProvinciaDos))
    (RespuestaDatos (Identificador Interesado) (DatoRecogido CAutonoma) (
        Respuesta ?CAutonomaDos))
    (RespuestaDatos (Identificador Interesado) (DatoRecogido CP) (
        Respuesta ?CPDos))
    (RespuestaDatos (Identificador Interesado) (DatoRecogido DNI) (
        Respuesta ?DNIDos))

    (RespuestaDatos (Identificador General) (DatoRecogido Ciudad) (
        Respuesta ?Ciudad))
    (RespuestaDatos (Identificador General) (DatoRecogido Dia) (Respuesta
        ?Dia))
    (RespuestaDatos (Identificador General) (DatoRecogido Mes) (Respuesta
        ?Mes))
    (RespuestaDatos (Identificador General) (DatoRecogido Anio) (Respuesta
         ?Anio))
    =>
    (bind ?final (sym-cat (sub-string 0 (- (str-index "....." ?texto) 1) ?
        texto) ?Nombre))
    (bind ?final (sym-cat ?final (sub-string (+ (str-index "....." ?texto)
         5) (- (str-index "......" ?texto) 1) ?texto) ?Direccion))
```

```
(bind ?final (sym-cat ?final (sub-string (+ (str-index "......" ?texto
    ) 6) (- (str-index "......." ?texto) 1) ?texto) ?Numero))
(bind ?final (sym-cat ?final (sub-string (+ (str-index "......." ?
    texto) 7) (- (str-index "........" ?texto) 1) ?texto) ?CP))
(bind ?final (sym-cat ?final (sub-string (+ (str-index "........" ?
    texto) 8) (- (str-index ".........." ?texto) 1) ?texto) ?Localidad))
(bind ?final (sym-cat ?final (sub-string (+ (str-index "........." ?
    texto) 9) (- (str-index ".........." ?texto) 1) ?texto) ?Provincia)
    )
(bind ?final (sym-cat ?final (sub-string (+ (str-index ".........." ?
    texto) 10) (- (str-index ".........." ?texto) 1) ?texto) ?
    CAutonoma))
(bind ?final (sym-cat ?final (sub-string (+ (str-index ".........." ?
    texto) 11) (- (str-index ".........." ?texto) 1) ?texto) ?DNI))

(bind ?final (sym-cat ?final (sub-string (+ (str-index ".........."
    ?texto) 12) (- (str-index ".........." ?texto) 1) ?texto) ?
    NombreDos))
(bind ?final (sym-cat ?final (sub-string (+ (str-index ".........."
     ?texto) 13) (- (str-index ".........." ?texto) 1) ?texto) ?
    DireccionDos))
(bind ?final (sym-cat ?final (sub-string (+ (str-index
    ".........." ?texto) 14) (- (str-index ".............." ?texto
    ) 1) ?texto) ?NumeroDos))
(bind ?final (sym-cat ?final (sub-string (+ (str-index
    ".............." ?texto) 15) (- (str-index ".............." ?
    texto) 1) ?texto) ?LocalidadDos))
(bind ?final (sym-cat ?final (sub-string (+ (str-index
    ".............." ?texto) 16) (- (str-index ".............." ?
    texto) 1) ?texto) ?ProvinciaDos))
(bind ?final (sym-cat ?final (sub-string (+ (str-index
    ".............." ?texto) 17) (- (str-index ".............."
    ?texto) 1) ?texto) ?CAutonomaDos))
(bind ?final (sym-cat ?final (sub-string (+ (str-index
    ".............." ?texto) 18) (- (str-index
    ".............." ?texto) 1) ?texto) ?CPDos))
(bind ?final (sym-cat ?final (sub-string (+ (str-index
    ".............." ?texto) 19) (- (str-index
    ".............." ?texto) 1) ?texto) ?DNIDos))

(bind ?final (sym-cat ?final (sub-string (+ (str-index
    ".............." ?texto) 20) (- (str-index
    ".............." ?texto) 1) ?texto) ?Ciudad))
(bind ?final (sym-cat ?final (sub-string (+ (str-index
    ".............." ?texto) 21) (- (str-index
    ".............." ?texto) 1) ?texto) ?Dia))
(bind ?final (sym-cat ?final (sub-string (+ (str-index
    ".............." ?texto) 22) (- (str-index
    ".............." ?texto) 1) ?texto) ?Mes))
(bind ?final (sym-cat ?final (sub-string (+ (str-index
    ".............." ?texto) 23) (- (str-index
    ".............." ?texto) 1) ?texto) ?Anio))
```

```
    (assert (FicheroProcesado (Nombre ?nombreFichero) (Texto ?final)))
)
```

Sé que parece una regla rara y puede que no muy clara, pero es sencilla. En el fichero txt que tenemos los campos a rellenar vienen indicados por una secuencia de puntos. Está secuencia de puntos parte de cinco puntos para el primer campo, seis puntos para el segundo y así sucesivamente. La idea de esta reglas es ir cogiendo los trozos de texto entre las secuencias de puntos e ir concatenándolas de manera intercalada con los datos introducidos por el usuario. Para saber donde están los puntos se ha usado la función *str-index* y para crear las subcadenas la función *sub-string*. Esta regla es aplicable a los 4 archivos, ya que todos ellos siguen el mismo patrón.

Para terminar, una vez tenemos el texto listo, tenemos una regla que se encarga de guardar el texto en un fichero de salida. Esta regla es la siguiente:

```
(defrule GuardarFicheros
        (FicheroProcesado (Nombre ?n) (Texto ?texto))
        =>
        (bind ?nombre (str-cat ?n "Procesado"))
        (open ?nombre data "w")
        (printout data ?texto crlf)
        (printout t "Se ha generado un fichero llamado " ?nombre " en el
           cual esta el documento relleno." crlf)
        (close data)
)
```

4. Breve manual de uso del sistema.

El sistema está muy guiado para que sea fácil su utilización. El primer contacto con el mismo nos preguntará que deseamos hacer, es decir, que módulo queremos ejecutar. Elegiremos nuestro módulo pulsando 1, 2 o 3. En cada modulo se nos van a ir haciendo preguntas. Para facilitar la interacción con el usuario y evitar que el mismo tenga que escribir demasiado, se ha intentado que la mayoría de la interacción entre el sistema experto y el usuario sea mediante números, es decir, que el sistema le de al usuario una lista y este elija su opción mediante números. Por lo tanto el uso es bien sencillo, simplemente debemos ir eligiendo las opciones que se nos plantean en cada módulo y finalmente obtendremos lo que buscábamos.