TÍTULO DE GRADO EN
INGENIERÍA INFORMÁTICA

Desarrollo de Sistemas Inteligentes

Documentación de la lectura del fichero y primeras reglas

CURSO 2019 / 2020

CONVOCATORIA DE ENERO

Eduardo Salmerón Castaño — eduardo.salmeronc@um.es

Victor García Puche — victor.garciap@um.es

Nicolás Fuentes Turpín — nicolas.fuentest@um.es

Índice general

1 - Introducción

La segunda fase del proyecto consiste en desarrollar un Sistema Inteligente capaz de leer y sacar todos los datos necesarios de un fichero log de un ECG mediante un motor de reglas.

2 - Interpretar el fichero

El primer paso que hemos tomado para realizar esta fase, ha sido crear un fichero java que lea el fichero log, y saque todos los datos de este. El fichero en cuestión se llama "Parser.java". Además, mientras se leen los datos, se van insertando como hechos en la base de hechos.

Este proceso de parseo del fichero consta de varias partes:

2.1 - Creación e inicio de la "*KieSession*"

Para empezar el parseo, antes de nada debemos crear una base de hechos e inicializarla con el nombre de la sesión ("ksession-rules").

2.2 - Parseo del ECG (ciclos y pulsaciones por minuto)

A continuación, empezamos leyendo el fichero y leyendo las primeras lineas (La primera linea de todas la, obviamos ya que contiene el "*ECG pattern*" del fichero). Estas lineas se corresponden con el número de ciclos y el número de pulsaciones por minuto del ECG. para obtener ambos valores, utilizamos una expresión regular para eliminar todo lo que no sean dígitos ("\D+"):

```
// Obtenemos el numero de ciclos
line = reader.readLine();
Cycles cycles = new Cycles(Integer.valueOf(line.replaceAll("\\D+", "")));
// Obtenemos el ritmo cardiaco
line = reader.readLine();
BPM bpm = new BPM(Integer.valueOf(line.replaceAll("\\D+", "")));
```

Ahora insertamos estos valores en la BH:

```
kSession.insert(cycles);
kSession.insert(bpm);
```

2.3 - Parseo del ECG (ondas)

En esta parte leemos cada onda y creamos una instancia de esta, dependiendo de cuál sea. Esto lo hacemos de forma dinámica, es decir, tenemos una variable de tipo *Wave* y dependiendo de la letra que leamos, creamos la onda de un tipo u otro. Esto podemos hacerlo ya que todos los tipos de onda heredan de *Wave*.

Para calcular en qué ciclo está cada onda, tenemos un contador inicializado a cero, que se incrementa cada vez que el parser encuentra una onda P (la primera onda de un ciclo).

Una vez tenemos la onda creada con todos los datos, ya podemos insertarla también en la BH.

```
char letter = line.charAt(0);
if (letter == 'P')
    actualCycle++;
String[] dataArray = line.replaceAll("[A-Z()]", "").split(",");

Wave wave;
switch (letter) {
case 'P':
    wave = new P(Integer.valueOf(dataArray[0]), Integer.valueOf(dataArray[1]),
            Float.valueOf(dataArray[2]), actualCycle);
    break;
case 'Q':
    wave = new Q(Integer.valueOf(dataArray[0]), Integer.valueOf(dataArray[1]),
            Float.valueOf(dataArray[2]), actualCycle);
    break;
case 'R':
    wave = new R(Integer.valueOf(dataArray[0]), Integer.valueOf(dataArray[1]),
            Float.valueOf(dataArray[2]), actualCycle);
    break;
case 'S':
    wave = new S(Integer.valueOf(dataArray[0]), Integer.valueOf(dataArray[1]),
            Float.valueOf(dataArray[2]), actualCycle);
    break;
case 'T':
    wave = new T(Integer.valueOf(dataArray[0]), Integer.valueOf(dataArray[1]),
            Float.valueOf(dataArray[2]), actualCycle);
    break;
default:
    wave = null;
    break;
}
```

2.4 - Lanzar las reglas

Llegamos a la última parte, en la que lo único que debemos hacer es lanzar todas las reglas con "fireAllRules" para que el motor de reglas empiece a lanzar reglas con los hechos que le hemos insertado. Las reglas que hemos creado se explicarán a continuación.

3 - Creación de reglas

En este apartado vamos a explicar las reglas que hemos creado y qué función cumple cada una. Además, En cada regla se notifica por consola que se ha ejecutado la regla. El fichero de reglas se llama “DomainRules”.

3.1 - Creación del complejo QRS

```
rule "QRS Complex Creation"
    when
        $q: Q($c : cycle)
        $r: R(cycle == $c)
        $s: S(cycle == $c)
    then
        QRSComplex qrs = new QRSComplex($q.getStart(), $s.getEnd(), $c);
        insert(qrs);
        System.out.println("Se ha creado el complejo QRS del ciclo numero " + $c);
end
```

En esta regla se crean los complejos QRS, es decir, tomamos las ondas Q, R y S pertenecientes a un mismo ciclo, creamos el complejo QRS y lo insertamos como hecho.

3.2 - Creación del intervalo QT

```
rule "QT Intervale Creation"
    when
        $q: Q($c : cycle)
        $t: T(cycle == $c)
    then
        QTIntervale qt = new QTIntervale($q.getStart(), $t.getEnd(), $c);
        insert(qt);
        System.out.println("Se ha creado el intervalo QT del ciclo numero " + $c);
end
```

Aquí se crean los intervalos QT de cada ciclo de forma similar a la anterior, y se inserta en la BH.

3.3 - Creación del segmento ST

```
rule "ST Segment Creation"
    when
        $s: S($c : cycle)
        $t: T(cycle == $c)
    then
        STSegment st = new STSegment($s.getStart(), $t.getEnd(), $c);
        insert(st);
        System.out.println("Se ha creado el segmento ST del ciclo numero " + $c);
end
```

Por último, tenemos la regla de creación de segmentos ST. De forma similar a las anteriores, crea el segmento con las ondas S y T de cada ciclo y lo inserta como hecho.