Tutorial de CBR...

April 21, 2016

1 Introduction

El objetivo de este prototipo es guardar problemas y soluciones para un sistema CBR, con las herramientas actuales de software: una base de datos relacional y un lenguaje de programación orientado por objetos.

Este sistema propone la definición de una familia de clases en las que se definirá el espacio del dominio de partida, P el codominio de las soluciones que se prentenden encontrar S y conjunto de algoritmos que transforman problemas a soluciones F. Teóricamente hablando, es posible visualizar el sistema como una herramienta que permite representar funciones de tipo

$$f : P \to S$$

, donde P es un problema, S es una solución y $f \in F$ es un algoritmo que transforma un problema específico en una solución.

En el CBR se requiere almacenar las soluciones para que después sean utilizadas cuando una instancia de un problema similar aparezca, evitando ser recalculada nuevamente. Este prototipo permite guardar esas soluciones en este momento. Una de las metas es lograr el ciclo CBR completo: retreive, reuse, revise y retain.

En este documento se muestra la implementación de varios problemas de ejemplo dentro del CBR implementado.

1.1 Ejemplo 1: suma de dos números

Dados dos números a y b, con $-100 \leq a, b \leq 100$, se quiere calcular la suma de los dos. En este caso definimos el problema como los dos números de entrada a y b. Es decir que el problema es definifo como $P = \{a, b\}$, donde $a, b \in \mathbb{R}$. La solución del problema es entonces un número $S = \{c\}$, donde $c \in \mathbb{R}$. Finalmente, el algoritmo solución es simplemente $c = a + b$.

Para codificar este problema en el CBR se deben implementar 3 clases: Problema, Solución y Algoritmo.

1.1.1 Clase Problema: SumProblem

Toda clase problema debe extender a la clase `Problem` e implementar el método `compare`, como lo muestra el Listado 1. La anotación `@Feature` expecifica que el atributo relacionado es un atributo de entrada para el CBR y será guardado en la base de datos. En este caso los atributos son `a` y `b`.

También se muestra la implementación del método `compare`. La impelementación se ha hecho para que devuelva un número real entre 0 y 1, donde 0 es el máximo de similaridad entro dos problemas.

Listing 1: Clase SumProblem

```
public class SumProblem extends Problem{

        @Feature(min=-100, max=100)
        public float a;

        @Feature(min=-100, max=100)
        public float b;

        @Override
        public double compare(Problem p) {
                SumProblem sp = (SumProblem)p;
                return (Math.abs(a-sp.a)+Math.abs(b-sp.b))/400;
        }

}
```

1.1.2 Clase Solución: SumSolution

La clase vinculada a la solución de un problema debe extender a la clase Solution y especificar el tipo de problema con el que está relacionada. El listado 2 muestra la implementación de esta clase. En este caso la clase parametrizada sería `Solution<SumProblem>`. La variable que contiene la solución en esto caso es `value`, y está anotada con la anotación `@Result`. Esta anotación le indica al CBR que debe guardar este resultado como una valor resultado.

El método `compare` indica que tan parecidas son dos soluciones. El método `check` indica si una solución para un problema es correcta o no.

Listing 2: Clase SumSolution

```
public class SumSolution extends Solution<SumProblem>{

        @Result
        public float value;

        @Override
        public double compare(Solution s){ return 0; }

        @Override
        public boolean check() { return true;  }

}
```

1.1.3 Clase Algoritmo: SumAlgorithm

Las clases que toman como entrada un problema y devuelven una salida son llamadas algoritmos por el CBR. Es posible implementar varias estrategias de solución para un mismo problema. El listado 3 muestra la implementación de esta clase. Para implementar un algoritmo se debe extender de la superclase `Algorithm` especificando la clase problema con la que está relacionada como parémetro de clase, en este caso `Algorithm<SumProblem>`.

La clase debe implementar el metodo `solve`, que recibe recibir una instancia del problema relacionado como parámetro y devolver una solución. En el listado 3 se puede observar una solución que toma los dos números guardados dentro de un problema, es decir las variables miembro `a` y `b` de una instancia de la clase `SumProblem`, y son sumados como valor respuesta. Este valor es guardado en una instancia de tipo `SumSolucion` y retornada como la solución del algoritmo.

Listing 3: Clase SumAlgorithm

```
public class SumAlgorithm extends Algorithm<SumProblem>{

        public SumSolution solve(SumProblem p){

                float sum = p.a + p.b;
                SumSolution result = new SumSolution();
```

```
            result.value = sum;

            return result;
        }

}
```

1.1.4 Uso del CBR

Una vez esté especificada la familia de clases que contiene el problema, la solución y los algoritmos a usar, es posible usar el CBR creando una instancia de CBR. La manera crear una instancia es mostrada en el código 4. Es necesario hacer dos cosas: 1) parametrizar la clase CBR con el problema y la solución a usar, y 2) especificar los algoritmos solución a usar.

Una vez hecho esto es posible crear problemas y usar el CBR para que solucione y guarde la solución dentro del sistema. El código 4 muestra la definición del problema cuando $a = 1$ y $b = 4$. Para que el problema sea solucionado se debe llamar al método `solveCase` que devuelve la solución del problema encontrada. Este método autmáticamente guarda la solución dentro del CBR. La información guardada dentro del CBR puede ser consultada con el método `showKnowledge`.

Listing 4: Clase SumAlgorithm

```
public static void main(String[] args) {

        CBR<SumProblem, SumSolution> cbr
            = new CBR<SumProblem, SumSolution>(
            SumProblem.class, new SumAlgorithm() );

        SumProblem case1 = new SumProblem();
        case1.a = 1;
        case1.b = 4;
        SumSolution r1 = cbr.solveCase(case1)

        cbr.showKnowledge();

}
```

1.2 Ejemplo 2: problema del número de monedas

El problema del mínimo número de monedas dada una cantidad.

To do.....

2 Conclusions

To do.....