Informe de Prácticas

Practica 3 - ISSBC

Adam Mora Tortajada

18/03/2014

Contenido

Fichero: appDiagnostico.py

Este fichero únicamente realiza la función de crear instancias de la aplicación y de la ventana que se utilizara para representar de forma gráfica la funcionalidad del sistema implementado.

Para realizar esto previamente debemos importar los ficheros que contengan la parte de "vista" y "controlador" de nuestro diseño.

```
import ckVtsDiagnosticoD
import ckCtrlDiagnostico

observables =  ckCtrlDiagnostico.ckModApDiagnostico.bcEnfUrinarias.observables()
app = QtGui.QApplication(sys.argv) #Crea una instancia de aplicación
form = ckVtsDiagnosticoD.DiagnosticDlg("Fallos", observables) #Crea una instancia de
de una ventana
sys.exit(app.exec_())
```

Fichero: bcEnfUrinarias.py

En primer lugar nos encontramos con la estructura que nos permitiría predefinir los diferentes síntomas que puede presentar un paciente y así posteriormente poder mostrarlos por pantalla. Esta estructura se define con una clase denominada Observable() la cual será heredada por los distintos síntomas a especifica.

```
class Observable():
    def __init__(self,nombre=None,tipo=None,valoresPermitidos=None,valor=None):
        #print 'observable->',valor
        self.nombre=nombre
        self.valor=valor
        self.tipo=tipo
        self.valoresPermitidos=valoresPermitidos
```

En caso de querer definir un síntoma, debemos crear una clase (en este caso la clase Fiebre()) la cual herede la estructura Observable() y así poder generar un objeto. A continuación se muestra un ejemplo en el que se demuestra como cumplimentar los distintos tipos de datos que contiene un síntoma.

```
class Fiebre(Observable):
    def __init__(self,valor=None):
        nombre='fiebre'
        tipo='multiple'
        valoresPermitidos=['normal','alta','muy alta']
        Observable.__init__(self,nombre ,tipo ,valoresPermitidos,valor)
        self.valor=valor
```

Con la siguiente función generamos una lista la cual recogerá la base de conocimiento la lista de observables.

```
def observables():
    obs=[]
    obs.append(Fiebre())
    obs.append(Disuria())
    obs.append(Hematuria())
    obs.append(DolorPerineal())
    obs.append(DolorLumbar())
    return obs
```

Se crea una estructura base que será heredada por las distintas enfermedades y así poder predefinirlas en el sistema.

```
class Enfermedad():
    def __init__(self,nombre):
        self.nombre=nombre
        self.ayuda=u''
```

Definimos las distintas enfermedades (en este caso ProstatitisAguda). Para ello creamos una clase que represente una enfermedad y herede la clase "estructura base" denominada Enfermedad. Una vez realizado esto creamos las instancias de las distintas observables y establecemos un valor por defecto en cada una de ellas

```
class ProstatitisAguda(Enfermedad):
    def __init__(self):
        Enfermedad.__init__(self,nombre=u'Prostatitis aguda'.encode(encoding='iso-
8859-1'))

        fiebreAlta=Fiebre([u'alta',u'muy alta'])
        disuria=Disuria(True)
        hematuria=Hematuria(False)
        dolorPerineal=DolorPerineal('agudo')

        self.debePresentar =[fiebreAlta,disuria]
        self.puedePresentar = [dolorPerineal]
        self.noPuedePresentar = [hematuria]
        self.ayuda=u'Ayuda sobre prostatitis aguda'.encode(encoding='iso-8859-1')
```

Creamos una lista con las distintas hipótesis de las enfermedades posibles. Esta lista nos facilitará el trabajo para la deducción de las distintas enfermedades según los síntomas seleccionados por pantalla.

```
def hipotesis():
    prAg=ProstatitisAguda()
    prCr=ProstatitisCronica()
    cr=CalculoRenal()
    lHp=[prAg,prCr,cr]
    return lHp
```

Fichero: ckCtrlDiagnostico.py

Este fichero se encarga de responder a eventos e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información. También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta de 'modelo'.

A continuación se muestra una función que nos permitirá realizar una inferencia sobre la CoberturaCausal() que hay definida en el modelo de este sistema. Esta "CoberturaCausal" presenta una lista de fallos y proporciona una lista de posibles hipótesis generadas según los distintos síntomas (observables) seleccionados por el usuario.

```
def eventCoberturaCausal(cdDiagnostico):
    fallos=[] #Lista que nos permitirá recoger los distintos fallos seleccionados
    for i in range(cdDiagnostico.tableWidgetPosiblesFallos.rowCount()):
        item1=cdDiagnostico.tableWidgetPosiblesFallos.item(i,0)
        if item1.checkState()==QtCore.Qt.CheckState.Checked:
            item2=cdDiagnostico.tableWidgetPosiblesFallos.cellWidget(i, 1)
            fallos.append( (item1.text(),item2.currentText()) )

    cc=ckModApDiagnostico.CoberturaCausal(fallos) #Invocamos inferencia de
CoberturaCausal pasando la lista de fallos como parametro
    cc.execute()
    lHipotesis=[]
    for h in cc.listaHipotesis: #recorremos la lista de hipótesis
        lHipotesis.append(h.nombre) #Obtenemos la lista de hipótesis según los
parámetros pasados a la inferencia de CoberturaCausal
    cdDiagnostico.listWidgetHipotesis.clear() #limpiamos la ventana para que en caso
de haber hecho un diagnóstico previo esta quede vacía de nuevo
    cdDiagnostico.listWidgetHipotesis.addItems(lHipotesis) #mostramos el nuevo
diagnóstico por ventana
```

Esta función nos permitirá controlar el evento de diagnóstico. En la primera parte obtendremos una lista de fallos y sus posibles valores. Estos fallos serán representados por tuplas.

```
def eventDiagnostica(cdDiagnostico,tr=False):
    cdDiagnostico.PlainTextEditExplicacion.clear()
    eventCoberturaCausal(cdDiagnostico,tr=False)
    fallos=[]
    for i in range(cdDiagnostico.tableWidgetPosiblesFallos.rowCount()):
        item1=cdDiagnostico.tableWidgetPosiblesFallos.item(i,0)
        if item1.checkState()==QtCore.Qt.CheckState.Checked:
            item2=cdDiagnostico.tableWidgetPosiblesFallos.cellWidget(i, 1)
            fallos.append( (item1.text(),item2.currentText()) )
```

Este apartado tratara de mostrar las distintas explicaciones en la ventana según el diagnóstico realizado. Para ello comprueba que los fallos captados son compatibles con la base de conocimiento, y posteriormente crea una instancia del método cc.

```
    observables=ckModApDiagnostico.identificaSignosSintomas(fallos)
    if not observables==None:
        mcc=ckModApDiagnostico.MetodoCoberturaCausal(observables)
        mcc.execute()
        print 'Justificación'
        print '=============='
        print mcc.explicacion
        print
        print 'Diagnóstico: '
        print '============ '
        for d in mcc.diagnostico:
            print d.nombre
        print 'fin'
        cdDiagnostico.PlainTextEditExplicacion.clear() #Limpia la ventana de explicación
        cdDiagnostico.PlainTextEditExplicacion.appendPlainText(mcc.explicacion)
#Cumplimenta dicha ventana con la información obtenida
        cdDiagnostico.PlainTextEditExplicacion.moveCursor(QtGui.QTextCursor.Start)
#Situa el cursor al principio
```

Cumplimentamos el apartado de diagnóstico. Previamente debemos borrar su contenido y posteriormente añadiremos el nuevo contenido.

```
        cdDiagnostico.listWidgetDiagnosticos.clear()
        lDiag=[]
        for d in mcc.diagnostico:
            lDiag.append(d.nombre)
            cdDiagnostico.listWidgetDiagnosticos.addItems(lDiag)
```

```
    return
    mcc=ckModApDiagnostico.MetodoCoberturaCausal(fallos)
    mcc.execute()
    print mcc.diagnostico
    print mcc.explicacion
```

Fichero: ckVtsDiagnosticoD.py

Presenta el 'modelo' en un formato adecuado para interactuar con la interfaz de usuario, por tanto requiere de dicho 'modelo' la información que debe representar como salida.
Al estar diseñado este sistema con PyQt, este fichero contendrá todas las configuraciones necesarias para mostrar todo su contenido a través de ventanas con sus respectivos Widgets.

```
class DiagnosticDlg(QtGui.QWidget):
    def __init__(self, name, observables=None, parent=None):
        super(DiagnosticDlg, self).__init__(parent)
        self.name = name
        labelListA=QtGui.QLabel("Selecione los Fallos Presentados",self)
        labelListB=QtGui.QLabel("",self)
        observables_list = [(f.nombre , f.valor)  for f in observables]
        header = ['NOMBRE', 'VALOR']

        self.tableWidgetPosiblesFallos = QtGui.QTableWidget(len(observables_list),2)
#Crea tabla de elementos
        self.tableWidgetPosiblesFallos.setColumnWidth(0, 400) #Tamaño de columnas
        self.tableWidgetPosiblesFallos.setColumnWidth(1, 400)
        self.tableWidgetPosiblesFallos.setHorizontalHeaderLabels(header) #Etiquetas de
columnas
```

En el apartado siguiente se crea un ítem u objeto y se le asigna el nombre del síntoma. Posteriormente se le asignan propiedades a las celdas de la primera columna.

```
        for i in range(len(observables)):
            item1 = QtGui.QTableWidgetItem(observables[i].nombre)
            item1.setCheckState(QtCore.Qt.Checked)
            item1.setFlags(QtCore.Qt.ItemIsUserCheckable|QtCore.Qt.ItemIsEnabled)
```

Comprueba el tipo de observable, si es múltiple se crea un campo ComBox que nos permitiría escoger entre una lista de posibilidades.

```
            if observables[i].tipo=='multiple':
                combobox = QtGui.QComboBox()
                for j in observables[i].valoresPermitidos:
```

```
                combobox.addItem(j)
            self.tableWidgetPosiblesFallos.setCellWidget(i, 1, combobox)
        elif observables[i].tipo=='boleano':
            combobox = QtGui.QComboBox()
            combobox.addItem('True')
            combobox.addItem('False')
            self.tableWidgetPosiblesFallos.setCellWidget(i, 1, combobox)

        self.tableWidgetPosiblesFallos.setItem(i, 0, item1) #Se establece el item en la
columna 0
```

Creamos una listwidget para las posibles hipótesis y otra distinta para los diagnósticos, y un editor de texto para situar la explicación.

```
    labelHipotesisL=QtGui.QLabel("Posibles Hipotesis",self)
    labelHipotesisR=QtGui.QLabel("",self)
    self.listWidgetHipotesis = QtGui.QListWidget()

    labelDiagnosticoL=QtGui.QLabel("Diagnóstico",self)
    labelDiagnosticoR=QtGui.QLabel("",self)
    self.listWidgetDiagnosticos = QtGui.QListWidget()

    labelExplicacionL=QtGui.QLabel("Explicacion",self)
    labelExplicacionR=QtGui.QLabel("     ",self)
    self.PlainTextEditExplicacion = QtGui.QPlainTextEdit()
```

Configuramos los botones para que realicen la función deseada y los incorporamos al buttonsLayout.

```
    self.coberturaCausalButton=QtGui.QPushButton('Cobertura Causal')#Para
invocar el método de cobretura causal
    self.diagnosticaButton=QtGui.QPushButton('Diagnostica') #Para ejecutar el
diagnóstico
    self.exitButton=QtGui.QPushButton('Exit') #Para salir del programa
    self.buttonsLayout = QtGui.QHBoxLayout() #Gestor de diseño horizontal

    self.buttonsLayout.addStretch()
    self.buttonsLayout.addWidget(self.coberturaCausalButton)
    self.buttonsLayout.addWidget(self.diagnosticaButton)
    self.buttonsLayout.addWidget(self.exitButton)
    self.buttonsLayout.addStretch()
```

Configuramos la rejilla de distribución de los controles. Nos permitirá mostrar los controles en el orden deseado.

```
grid = QtGui.QGridLayout()
grid.setSpacing(5)
grid.addWidget(labelListA, 0, 0)
grid.addWidget(self.tableWidgetPosiblesFallos, 1, 0,1,2)
grid.addWidget(labelListB, 0, 1)

grid.addWidget(labelHipotesisL, 2, 0)
grid.addWidget(labelHipotesisR, 2, 1)
grid.addWidget(self.listWidgetHipotesis, 3, 0,1,2)

grid.addWidget(labelDiagnosticoL, 4, 0)
grid.addWidget(labelDiagnosticoR, 4, 1)
grid.addWidget(self.listWidgetDiagnosticos, 5, 0,1,2)
grid.addWidget(labelExplicacionL, 6, 0)
grid.addWidget(labelExplicacionR, 6, 1)
grid.addWidget(self.PlainTextEditExplicacion, 7, 0,1,2)

#Diseño principal de la ventana
mainLayout = QtGui.QVBoxLayout()
mainLayout.addLayout(grid)
mainLayout.addLayout(self.buttonsLayout)
self.setLayout(mainLayout) #Asignar a la ventana la distribución de los controles


self.setGeometry(300, 300, 900, 1200)
self.setWindowTitle(u"TAREA DE DIAGNOSTICO".format(self.name))
self.show()

self.center()
```

Los botones por si mismos no realizan ninguna acción excepto visual. Para ello necesitamos realizar ciertas Conexiones entre los botones y funciones, y eventos en las listas:

```
self.tableWidgetPosiblesFallos.itemDoubleClicked.connect(self.moveRight)
self.coberturaCausalButton.clicked.connect(self.coberturaCausal)
self.diagnosticaButton.clicked.connect(self.diagnostica)
self.exitButton.clicked.connect(self.close)
```

Fichero: ckModApDiagnostico.py

Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación. Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada. Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador'.

La clase siguiente realiza un método de cobertura causal para la tarea de diagnóstico.

```
class MetodoCoberturaCausal():
    def __init__(self,fallos):
        self.fallos=fallos
        self.explicacion=''
        self.diferencial=[]
        self.diagnostico=[]
        pass
```

La siguiente función nos permite obtener el conjunto diferencial a partir de unos observables dados los fallos escogidos por los usuarios.

```
    def obtenerConjuntoDiferencial(self):
        cc=CoberturaCausal(self.fallos)
        self.diferencial=cc.execute()
        return self.diferencial
```

Se obtiene el conjunto diferencial invocando a la inferencia de cobertura causal y nos permite la ejecución del método de cobertura causal para la tarea de diagnóstico.
Obtendremos el conjunto diferencial invocando a la inferencia de cobertura causal, lo que nos permitiría construir la explicación basándonos en una hipótesis

```
    def execute(self,tr=False):
        self.explicacion+=u'Ejectutando cobertura causal. '.encode(encoding='iso-8859-1')
        cc=CoberturaCausal(self.fallos)
        self.diferencial=cc.execute()
        self.explicacion+=u'Se obtiene el conjunto diferencial: \n'.encode(encoding='iso-
8859-1')

        for f in self.diferencial:
            self.explicacion+=f.nombre+'\n'

        while len(self.diferencial)>0:
            seleccionar=Seleccionar(self.diferencial)
            hipotesis=seleccionar.execute()
            if tr:
```

```
            print 'hipotesis seleccionada', hipotesis
            print '======================'

        self.explicacion+=u'\nProbamos la  hipótesis de '.encode(encoding='iso-8859-
1')
        self.explicacion+=hipotesis.nombre+'\n'
        if tr:
            print
            print 'antes de verificar', self.fallos,hipotesis
            print 'hipotesis->',hipotesis.debePresentar

        verifica=Verificar(self.fallos,hipotesis
        verifica.execute(tr=tr)

        if verifica.resultado==False
            self.diferencial.remove(hipotesis)
            pass
        else:
            self.diagnostico.append(hipotesis)#Añade a la lista de diagnósticos
compatibles la hipotesis
            self.diferencial.remove(hipotesis)#Suprime la hipótesis evaluada
        self.explicacion+=verifica.justificacion  #Añade a la justificación la justificación
de la verificación
    pass
    if tr:
        print 'EL DIAGNOSTICO ES :',self.diagnostico
```

Ahora vamos a verificar si una hipótesis de avería es compatible con un conjunto de fallos.

```
class Verificar(Inferencia):
  def __init__(self,fallos,hipotesis):
    Inferencia.__init__(self)
    self.fallos=fallos
    self.resultado=None
    self.hipotesis=hipotesis
    self.justificacion=''
  def execute(self,tr=False):
    #Eliminar aquellas hipotesis que no tengan todos los sintomas en debe de estar
    resultado=True
    if tr:
        print 'verificando la hipotesis:',self.hipotesis.nombre, self.hipotesis
    for fh in self.hipotesis.debePresentar:#Cada fallo de la hipotesis debePresentar
debe de estar en fallos
                                # con sus valores aprepiados
        if tr:
            print 'debe presentar:', fh,fh.nombre, fh.valor,'->',self.fallos
```

```
            print 'debe presentar:', (fh.nombre, fh.valor) ,'->',[(f.nombre,f.valor) for f in
self.fallos]

          if not (fh.nombre in [f.nombre for f in self.fallos]):#Si el nombre del fallo de la
hipótesis no está
                                                  #en la lista de nombres de los fallos
presentados
                                                  #Construye la explicación
             self.resultado=False
             self.justificacion+=u'    No puede ser  '.encode(encoding='iso-8859-1')
             self.justificacion+=self.hipotesis.nombre
             self.justificacion+=u' porque debería presentar el fallo
'.encode(encoding='iso-8859-1')
             self.justificacion+=fh.nombre+' \n'
             #print type(f.valor)
             if isinstance(fh.valor,bool):
                self.justificacion+=str(fh.valor)+'\n'
             elif isinstance(fh.valor,str):
                self.justificacion+= fh.valor+'\n'
             resultado=False
          else: #El nombre del fallo de la hipótesis está pero debe de coincidir los
valores
             falla=False #Bandera
             for e in self.fallos:
                if e.nombre==fh.nombre:#comprueba que coincide en valores
                   if  isinstance(fh.valor,list):#Si el valor del fallo de la hipótesis es de tipo
lista
                      if not e.valor in fh.valor:#Comprueba que el valor del fallo presentado
está en esa lista
                         falla=True #El valor del fallo presentado no está en la lista
                         break
                   else:#El valor del fallo de la hipótesis no es de tipo lista
                      if not e.valor==fh.valor:#Si no coincide los valores falla
                         falla=True
                         break

             if falla:#Si se ha fallado se añade a la justificación--->Mejorar
                self.justificacion+=u'    No puede ser  '.encode(encoding='iso-8859-1')
                self.justificacion+=self.hipotesis.nombre
                self.justificacion+=u' porque debería presentar el fallo
'.encode(encoding='iso-8859-1')
                self.justificacion+=fh.nombre+' \n con con valor apropiado.'
                resultado=False

          if resultado==False:#Si ha resultado fallida la verificación salimos de la
verificación.
             self.resultado=False
             return (False,self.justificacion)
```

```
            else:
                self.justificacion+=u' Puede ser  '.encode(encoding='iso-8859-
1')+self.hipotesis.nombre+'.\n'


        #Eliminar aquellas hipotesis que tenga algun fallo en no debe tener
        for f in self.hipotesis.noPuedePresentar:
            if (f.nombre, f.valor) in [(e.nombre,e.valor) for e in self.fallos]:
                self.resultado=False
                self.justificacion+=u'    No puede ser  '.encode(encoding='iso-8859-1')
                self.justificacion+=self.hipotesis.nombre
                #self.justificacion+=f.nombre+' '
                self.justificacion+=u' porque esta enfermedad no puede presentar el fallo
'.encode(encoding='iso-8859-1')
                self.justificacion+=f.nombre+' con valor '
                if isinstance(f.valor,bool):
                    self.justificacion+=str(f.valor)+'\n'
                elif isinstance(f.valor,str):
                    self.justificacion+= f.valor+'\n'
                resultado=False
        if resultado==False:
            self.resultado=False
            return (False,self.justificacion)

        self.resultado=True
        return (True,self.justificacion)
```

Ejercicio Práctico

Enlace al documento

www.uco.es/~i02motoa/issbc/cooodigo/pr3/Adam_Mora_Tortajada_-_Informe_3.pdf