



Intelligence Artificielle

Programmation fonctionnelle, logique et par
contraintes

Hendrickx Derek et Paligot Gérard

21 Décembre 2011

Donatien Grolaux

Table of Contents

Introduction	1
Code source.....	2
MyCellule.scala	2
MyFrame.scala	4
Simplification.scala	5
Conclusion	13
Différents avis	13
Différents types de programmation.....	13
Réalisation du projet	14
Déroulement du cours	14

Introduction

Dans le cadre du cours de “Programmation fonctionnelle, logique et par contraintes”, il nous a été demandé d’élaborer un programme codé en Scala afin de simuler la génération d’un horaire sous certaines contraintes.

Le déroulement du cours se déroulait de la manière suivante : Les premières semaines ont été consacrées à une première approche du langage Scala suivant certaines façon de programmer (fonctionnelle, logique et par contraintes). Orienté essentiellement observation d’un code donné.

Après quoi, les séances restantes ne servaient que à des séances de question-réponse avec le professeur, M. Grolaux. Nous aborderons dans la conclusion de ce dossier notre avis sur l’organisation de ce cours.

Bonne lecture.

Code source

MyCellule.scala

```
class MyCellule(prof: Int, serie: Int, cours: Int) extends GridBagPanel {
  var mProf = new Label(convertIntToProf(prof))
  var mSerie = new Label(convertIntToSerie(serie))
  var mCours = new Label(convertIntToCours(cours))

  // Contraintes d'affichage
  val c = new Constraints
  c.gridwidth = 0
  c.anchor = GridBagConstraints.Anchor.West
  c.weightx = 1

  // Affichage des labels
  layout(mProf) = c
  layout(mSerie) = c
  layout(mCours) = c

  /* Match l'indice d'un prof avec sa valeur */
  def convertIntToProf(indice: Int) = {
    indice match {
      case 0 => ""
      case 1 => "Grolaux"
      case 2 => "Gancberg"
      case 3 => "Debacker"
      case 4 => "Seront"
      case 5 => "Legrand"
      case 6 => "Henriet"
      case 7 => "Emmeline"
      case _ => "Erreur"
    }
  }

  /* Match l'indice d'une série avec sa valeur */
  def convertIntToSerie(indice: Int) = {
    indice match {
      case 0 => ""
      case 1 => "Série 1"
      case 2 => "Série 2"
      case _ => "Erreur"
    }
  }
}
```

```

}

/* Match l'indice d'un cours avec sa valeur */
def convertIntToCours(indice: Int) = {
  indice match {
    case 0 => ""
    case 1 => "Internet/Intranet"
    case 2 => "Laboratoire réseaux"
    case 3 => "Patterns"
    case 4 => "JSP"
    case 5 => "Programmation Fonctionnelle, Logique et par Contraintes"
    case 6 => "EJB"
    case 7 => "C++"
    case 8 => "Programmation Graphique"
    case 9 => "Mobile"
    case 10 => "IHM"
    case 11 => "C"
    case 12 => "Admin Système"
    case 13 => "UML"
    case _ => "Erreur"
  }
}
}

```

MyFrame.scala

```
class MyFrame extends MainFrame {
  def tabJours = Array("Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi")

  // Initialise le conteneur des onglets
  val tabPanel = new TabbedPane

  // Initialise le MainFrame
  title = "Horaires"
  preferredSize = new Dimension(800, 570)
  contents = tabPanel
  centerOnScreen()
  visible = true

  def add(local: Int, matrice: Array[Array[MyCellule]]): Unit = {
    val scroll = new ScrollPane(new
Table(matrice.toArray.asInstanceOf[Array[Array[Any]]], tabJours) {
      rowHeight = 60
      enabled = false
      override protected def renderComponent(isSelected: Boolean,
focused: Boolean, row: Int, column: Int) = {
        apply(row, column).asInstanceOf[Component]
      }
    })
    // On ajoute le scroll à l'onglet
    tabPanel.pages += new TabbedPane.Page(convertIntToLocal(local), scroll)
  }

  /* Match l'indice d'un local avec sa valeur */
  def convertIntToLocal(indice: Int) = {
    indice match {
      case 0 => "Local 017"
      case 1 => "Local 019"
      case _ => "Erreur"
    }
  }
}
```

Simplification.scala

```
object Simplification extends jacop {
  def indiceTab(jour: Int, heure: Int) = jour + heure
  def main(args: Array[String]): Unit = {
    // -----
    // Déclaration des constantes

    val nothing = 0

    // Cours
    val intranet = 1
    val laboReseaux = 2
    val patterns = 3
    val jsp = 4
    val pflc = 5
    val ejb = 6
    val cpp = 7
    val pgcpp = 8
    val imo = 9
    val ihm = 10
    val c = 11
    val adminSys = 12
    val uml = 13

    // Professeurs
    val grolaux = 1
    val gancberg = 2
    val debacker = 3
    val seront = 4
    val legrand = 5
    val henriet = 6
    val emmeline = 7

    // Séries
    val serie1 = 1
    val serie2 = 2

    // Locaux
    val l017 = 0
    val l019 = 1

    // Heures
    val h830 = 0
  }
}
```

```

val h930 = 1
val h1030 = 2
val h1130 = 3
val h13 = 4
val h14 = 5
val h15 = 6
val h16 = 7

// Jours
val lundi = 0
val mardi = 8
val mercredi = 16
val jeudi = 24
val vendredi = 32

// Liste
val jours = List(lundi, mardi, mercredi, jeudi, vendredi)
val heures = List(h830, h930, h1030, h1130, h13, h14, h15, h16)
val locaux = List(l017, l019)
val profs = List(grolaux, gancberg, debacker, seront, legrand, henriet,
emmeline)
val series = List(serie1, serie2)

// -----
// Declaration et initialisation de la matrice

type unSlotHoraire = (IntVar, IntVar, IntVar)
val matrice = for (z <- List.range(0, 2)) yield for (i <- List.range(0,
40)) yield (new IntVar("prof", 0, 7), new IntVar("serie", 0, 2), new
IntVar("cours", 0, 13));

// -----
// Déclarations des méthodes utilisant la matrice directement

/* Définit la boucle parcourant tous les locaux */
def tousLesLocaux(f: (Int) => Unit) = {
  for (local <- locaux) {
    f(local)
  }
}

/* Définit la boucle parcourant tous les slots une fois */
def tousLesSlotsHoraires(f: (Int) => Unit) = {
  for (slot <- List.range(0, 40)) {
    f(slot)
  }
}

```

```

    }
  }

  /* Définit la boucle parcourant tous les locaux pour tous les slots */
  def tousLesLocauxEtLesSlotsHoraires(f: (Int, Int) => Unit) = {
    for (local <- locaux) {
      for (slot <- List.range(0, 40)) {
        f(local, slot)
      }
    }
  }

  /* Le prof, la série et le cours pour un slot donné */
  def tousLesIntVarPourUnSlotDonne(slot: Int) = {
    for (local <- locaux) yield matrice(local)(slot);
  }

  /* Parcours les profs de ma liste */
  def lesProfsDeMaListe(l: List[unSlotHoraire]) = {
    for (i <- l) yield i._1
  }

  /* Parcours les cours de ma liste */
  def lesCoursDeMaListe(l: List[unSlotHoraire]) = {
    for (i <- l) yield i._2
  }

  /* Parcours les séries de ma liste */
  def lesSeriesDeMaListe(l: List[unSlotHoraire]) = {
    for (i <- l) yield i._3
  }

  /* Définit les heures et les jours que le prof donné n'aura pas cours */
  def neDonnePasCours(heuresPasCours: List[Int], jourPasCours: List[Int],
prof: Int) {
    tousLesLocaux((i) => {
      for (j <- jourPasCours)
        for (k <- List.range(0, 8))
          matrice(i)(indiceTab(j, k))._1 != prof

      for (j <- jours -- jourPasCours)
        for (k <- heuresPasCours)
          matrice(i)(indiceTab(j, k))._1 != prof
    })
  }
}

```

```

/* Tous les profs de ma matrice */
def tousProfs(): List[IntVar] = {
  (for (slot <- List.range(0, 40)) yield matrice(l017)(slot)._1) :::
(for (slot <- List.range(0, 40)) yield matrice(l019)(slot)._1)
}

/* Tous les cours de ma matrice */
def tousCours(): List[IntVar] = {
  (for (slot <- List.range(0, 40)) yield matrice(l017)(slot)._3) :::
(for (slot <- List.range(0, 40)) yield matrice(l019)(slot)._3)
}

/* Renvoi tous les profs pour un jour donné */
def tousProfsJourPrecis(jour: Int): List[IntVar] = {
  (for (heure <- List.range(0, 8)) yield matrice(l017)(indiceTab(jour,
heure))._1) ::: (for (heure <- List.range(0, 8)) yield
matrice(l019)(indiceTab(jour, heure))._1)
}

/* Renvoi les séries pour le jour donné */
def tousSerieJourPrecis(jour: Int): List[IntVar] = {
  (for (heure <- List.range(0, 8)) yield matrice(l017)(indiceTab(jour,
heure))._2) ::: (for (heure <- List.range(0, 8)) yield
matrice(l019)(indiceTab(jour, heure))._2)
}

/* Assigne un cour pour un slot */
def assignationCoursPourHeuresJours(heures: List[Int], jours: List[Int],
cours: Int, local: Int) {
  tousLesLocaux((i) => {
    for (heure <- heures) for (jour <- jours)
matrice(local)(indiceTab(jour, heure))._3 == cours
  })
}

// -----
// Application de contraintes générales

/* Les prof et séries ne donnent pas cours dans le même local */
tousLesSlotsHoraires((i) => {
  for (p <- profs)
count(lesProfsDeMaListe(tousLesIntVarPourUnSlotDonne(i)), p) <= 1
  for (s <- series)
count(lesSeriesDeMaListe(tousLesIntVarPourUnSlotDonne(i)), s) <= 1
}

```

```

})

/* Assignations des cours à certains cours + gestion du slot vide */
tousLesLocauxEtLesSlotsHoraires((local, slot) => {
  OR(
    /* Grolaux donne le cours de patterns */
    AND(matrice(local)(slot)._3 == patterns, matrice(local)(slot)._1 ==
grolaux),
    AND(matrice(local)(slot)._3 == pflc, matrice(local)(slot)._1 ==
grolaux),
    /* Gancberg donne le cours de intranet */
    AND(matrice(local)(slot)._3 == intranet, matrice(local)(slot)._1 ==
gancberg),
    AND(matrice(local)(slot)._3 == laboReseaux, matrice(local)(slot)._1
== gancberg),
    /* Debacker donne le cours de jsp et ejb */
    AND(matrice(local)(slot)._3 == ejb, matrice(local)(slot)._1 ==
debacker),
    AND(matrice(local)(slot)._3 == jsp, matrice(local)(slot)._1 ==
debacker),
    /* Seront donne le cours de cpp et pgcpp */
    AND(matrice(local)(slot)._3 == pgcpp, matrice(local)(slot)._1 ==
seront),
    AND(matrice(local)(slot)._3 == cpp, matrice(local)(slot)._1 ==
seront),
    /* Legrand donne le cours de imo et ihm */
    AND(matrice(local)(slot)._3 == imo, matrice(local)(slot)._1 ==
legrand),
    AND(matrice(local)(slot)._3 == ihm, matrice(local)(slot)._1 ==
legrand),
    /* Henriet donne le cours de c et adminSys */
    AND(matrice(local)(slot)._3 == c, matrice(local)(slot)._1 ==
henriet),
    AND(matrice(local)(slot)._3 == adminSys, matrice(local)(slot)._1 ==
henriet),
    /* Emmeline donne le cours de uml */
    AND(matrice(local)(slot)._3 == uml, matrice(local)(slot)._1 ==
emmeline),
    /* Permet que le cours puisse ne rien avoir */
    AND(matrice(local)(slot)._3 == nothing, matrice(local)(slot)._1 ==
nothing))
    /* Indique que si une valeur = 0, toutes les autres doivent l'être */
    OR(count(List(matrice(local)(slot)._1, matrice(local)(slot)._2,
matrice(local)(slot)._3), 0) == 0, count(List(matrice(local)(slot)._1,
matrice(local)(slot)._2, matrice(local)(slot)._3), 0) == 3)

```

```

})

// -----
// Série

// Chaque jour, la série 1 doit avoir au moins 2 cours par jour
for (jour <- jours) count(tousSerieJourPrecis(jour), serie1) >= 2
// Chaque jour, la série 2 doit avoir au moins 2 cours par jour
for (jour <- jours) count(tousSerieJourPrecis(jour), serie2) >= 2

// -----
// Grolaux

// Définit les heures et les jours de non cours
var heurePasCours = List(h830, h1130, h13, h16)
var jourPasCours = List(jeudi, vendredi)

neDonnePasCours(heurePasCours, jourPasCours, grolaux)

// Grolaux ne donne que 2 cours au plus par jours
for (jour <- jours -- jourPasCours) count(tousProfsJourPrecis(jour),
grolaux) == 3

// -----
// Gancberg

// Définit les heures et les jours de non cours
heurePasCours = List(h830, h1130, h13, h16)
jourPasCours = List(jeudi, vendredi)

neDonnePasCours(heurePasCours, jourPasCours, gancberg)

// Gancberg ne donne que 3 cours au plus par jours
for (jour <- jours -- jourPasCours) count(tousProfsJourPrecis(jour),
gancberg) == 4

// -----
// Debacker

// Définit les heures et les jours de non cours
heurePasCours = List(h830, h930)
jourPasCours = List(lundi, vendredi)

neDonnePasCours(heurePasCours, jourPasCours, debacker)

```

```

// Debacker ne donne que 2 cours au plus par jours
for (jour <- jours -- jourPasCours) count(tousProfsJourPrecis(jour),
debacker) == 4

// -----
// Seront

// Définit les heures et les jours de non cours
heurePasCours = List(h13, h14)
jourPasCours = List(lundi, mardi, mercredi, jeudi)

neDonnePasCours(heurePasCours, jourPasCours, seront)

// Seront ne donne que 2 cours au plus par jours
for (jour <- jours -- jourPasCours) count(tousProfsJourPrecis(jour),
seront) == 5

// -----
// Legrand

// Définit les heures et les jours de non cours
heurePasCours = List(h13, h14)
jourPasCours = List(lundi, mardi, mercredi, jeudi)

neDonnePasCours(heurePasCours, jourPasCours, legrand)

// Seront ne donne que 2 cours au plus par jours
for (jour <- jours -- jourPasCours) count(tousProfsJourPrecis(jour),
legrand) == 3

// -----
// Henriet

// Définit les heures et les jours de non cours
heurePasCours = List(h830, h930, h16)
jourPasCours = List(vendredi)

neDonnePasCours(heurePasCours, jourPasCours, henriet)

// Seront ne donne que 2 cours au plus par jours
for (jour <- jours -- jourPasCours) count(tousProfsJourPrecis(jour),
henriet) == 2

// -----
// Emmeline

```

```

// Définit les heures et les jours de non cours
heurePasCours = List(h13, h14)
jourPasCours = List(lundi, mercredi, jeudi)

neDonnePasCours(heurePasCours, jourPasCours, emmeline)

// Seront ne donne que 2 cours au plus par jours
for (jour <- jours -- jourPasCours) count(tousProfsJourPrecis(jour),
emmeline) == 2

// -----
// Génération du résultat

val f = matrice flatten

val tousIntVar = lesProfsDeMaListe(f) ::: lesCoursDeMaListe(f) :::
lesSeriesDeMaListe(f)

val result = satisfy(search(tousIntVar.toArray, first_fail,
indomain_middle))

val frame = new MyFrame
if (result) {
  for (local <- locaux) {
    val horaire =
      for (heure <- heures) yield {
        for (jour <- jours) yield {
          new MyCellule(matrice(local)(indiceTab(jour,
heure))._1.value(), matrice(local)(indiceTab(jour, heure))._2.value(),
matrice(local)(indiceTab(jour, heure))._3.value())
        }
      }.toArray
    frame.add(local, horaire.toArray)
  }
} else {
  println("Pas de résultat")
}
}
}

```

Conclusion

Nous avons trouvé très instructif la programmation de ce projet sur base des cours pratiques que nous avons du suivre avant le commencement du projet. Le travail fourni par ce projet nous a permis de progresser sur plusieurs points :

- Apprentissage du langage Scala
- Apprentissage de la librairie JaCop
- Apprentissage de la programmation par contraintes avec Scala
- Pouvoir se détacher de la programmation impérative

Nous pouvons dire que nous sortons grandis de ce projet. Les compétences acquises sont non négligeables pour diversifier nos compétences et pour se rendre compte par la pratique que les autres techniques de programmation ne sont pas mauvaises tout au contraire. Nous avons pris énormément de plaisir à participer à ce projet.

Différents avis

Différents types de programmation

La programmation fonctionnelle a largement été abordée durant le cours. Nous avons pu nous rendre compte que Scala était particulièrement adapté pour ce type de programmation.

Après quoi, nous nous sommes très brièvement attardé sur la programmation logique. Ce dernier nous avait déjà été enseigné en 1^{ère} et 2^{ème} année avec le cours de Mathématique avec le logiciel Prolog.

Nous avons fini par la programmation par contraintes où Scala excelle dans son application.

On ne cachera pas que nous avons une nette attirance pour la programmation par contraintes qui a été moins abordés en cours pratique mais intégralement utilisé pour le projet. Nous nous sommes rendus compte que les possibilités offertes par cette dernière technique de programmation étaient à la fois puissante et simple d'application pour peu que le développeur parvienne à « penser » dans cette approche de programmation.

Réalisation du projet

La réalisation du projet a été particulièrement difficile au vu des nombreux projets que nous devions rendre dans les autres cours. Notre binôme s'accordait très difficilement au niveau des horaires. Il a donc fallu devoir travailler séparément et tenir au courant l'autre membre du groupe régulièrement sur l'avancée du projet.

Cependant, nous sommes parvenus à nous organiser grâce à des outils comme SVN, Freedcamp (gestionnaire de ToDo pour projet) et même un groupe Facebook puisqu'il s'agit d'une plateforme où l'un comme l'autre, nous passons beaucoup de temps.

Déroulement du cours

Nous étions au courant que cette année était très expérimentale pour M. Grolaux. Il y a forcément quelques points sur lesquels nous voulons nous attarder afin d'améliorer ce cours qui possède énormément de potentiel :

- Il serait peut-être judicieux de s'attarder d'avantage sur la programmation par contraintes et moins sur la programmation fonctionnelle si le projet porte sur la programmation par contraintes.
- Nous regrettons d'avoir programmer que très peu en Scala avant de nous lancer dans le projet. Il était alors très difficile de « penser » en programmation par contraintes.

Merci de votre lecture !