INFOM451 - Conception d'Applications Mobiles

Rapport

Groupe Environnement

Auteurs :
Cyril Carlier
Axel Halin
Dorian Lecomte

9 mai 2016

Table des matières

1 Introduction

Ce projet, développé dans le cadre du cours *INFOM451 - Conception d'applications mobiles*, consiste en l'implémentation d'une interface pour l'interpréteur Bach-T. Dans un deuxième temps, il nous était demandé d'implémenter un langage permettant l'affichage d'image(s) en fonction du contenu du tableau.

2 Technologies

L'ensemble du projet a été réalisé en ***Scala***. Nous avons utilisé, pour l'interface graphique, la bibliothèque ***Swing Api*** (c.f : http ://www.scala-lang.org/api/2.11.2/scala-swing/).

Cette interface est construite au dessus de l'interpéteur Bach-T dont le code source nous as été fourni et dont une copie se trouve en annexe. Les modifications apportées au code source de base sont reprises et détaillées à la section suivante.

3 Contenu

3.1 Première Question

Cette première question concernait le design et l'implémentation d'une interface graphique. Des images d'une interface réalisée pour une extension de BachT nous étaient proposées en inspiration. Notre interface s'inspire grandement des images proposées et est composée de quatres fenêtres (nous détaillons dans cette section les trois principales, la quatrième sera présentée à la section suivante).

3.1.1 gui.scala

La classe ***gui.scala*** défini la fenêtre principale de l'application. Elle est définie comme suit :

— ***Content of the blackboard*** : *TextArea* affichant le contenu du tableau. Cet objet est mis à jour à chaque exécution d'une primitive ou d'un agent (autonome ou interactif).

— ***Clear*** : *Button* pour effacer le contenu du tableau ainsi que l'affichage du tableau.
— ***Token & Density*** : *TextFields* indiquant le nom ainsi que le nombre de tokens de la primitive.
— ***Primitives*** : *Buttons* pour exécuter une primitive. Ces primitives peuvent être de quatre types : tell, ask, get, nask. Elles sont exécutées via l'objet *BachTStore* de la classe ***bacht_cli.scala***.
— ***Agents*** : *Buttons* pour instantier un nouvel agent. Cet agent peut être soit autonome soit interactif (voir ci-dessous).

Cette fenêtre est visible à tout moment et sa fermeture entraine l'arrêt de l'application. Toute modification du tableau par un agent interactif ou autonome sera affiché sur cette page.

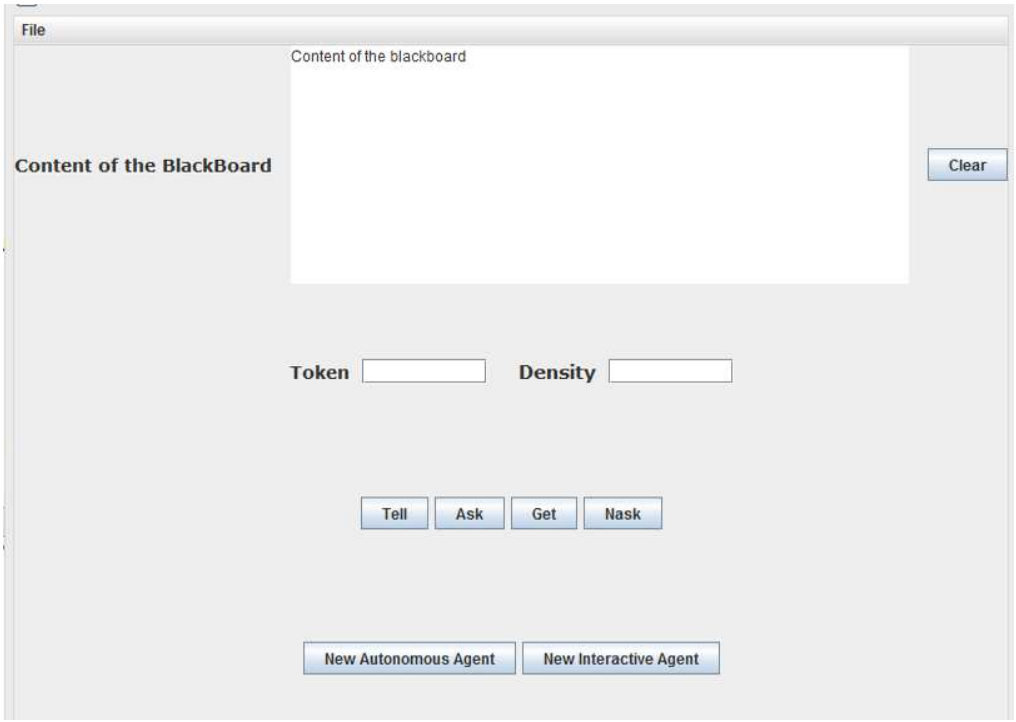

FIGURE 1 – gui.scala - Page d'accueil

3.1.2 AutonomousAgentGUI.scala

La classe ***AutonomousAgentGUI.scala*** défini la fenêtre relative aux agents autonomes. Un agent autonome permet à l'utilisateur de déclarer un

agent et de l'exécuter. Cette exécution peut se faire soit étape par étape, soit en une fois (auquel cas le tableau sera mis à jour à la fin de l'exécution et un message notifiera la réussite/l'échec de l'exécution).

La fenêtre est définie avec les composants suivants :

- ***Agent*** : *TextArea* pour déclarer un nouvel agent.
- ***Submit*** : *Button* pour valider l'agent. Une vérification est effectuée sur cet agent via la méthode de parsing *parse_agent* de la classe *BachTSimulParser*.
- ***Current Agent*** : *Label* affichant l'agent actuel. Cet agent est mis à jour à chaque exécution de la prochaine primitive.
- ***Run & Next*** : *Buttons* permettant respectivement d'exécuter entièrement l'agent ou d'exécuter la prochaine primitive.

Plusieurs instances d'agents autonomes sont possibles en parallèle. Ceux-ci partagent le même tableau.

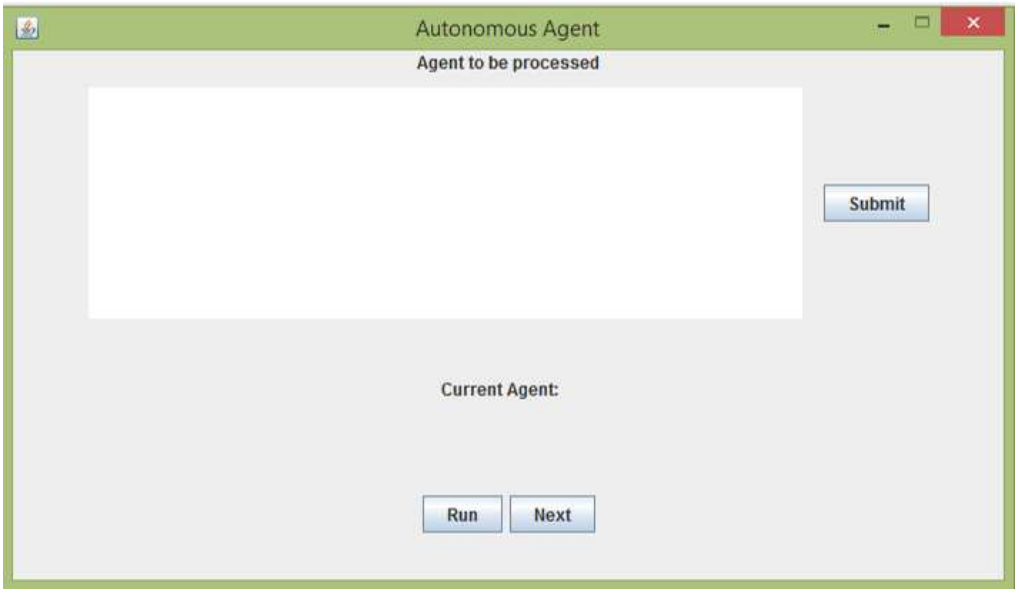

FIGURE 2 – AutonomousAgentGUI.scala - Ecran Agent Autonome

3.1.3 InteractiveAgentGUI.scala

La classe ***InteractiveAgentGUI.scala*** gère la fenêtre relative aux agents interactifs. Un tel agent permet à l'utilisateur de sélectionner lui même la primitive qu'il veut exécuter. Les primitives accessibles (c'est à dire les primitives dont l'exécution renvoit un *success*) sont accentuées par un bouton.

Elle est structurée de manière similaire à la classe de l'agent autonome :

- ***Agent*** : *TextArea* pour déclarer un nouvel agent.
- ***Submit*** : *Button* pour valider l'agent. Une vérification est effectuée sur cet agent via la méthode de parsing *parse_agent* de la classe *BachTSimulParser*.
- ***Current Agent*** : *BoxPanel* affichant l'agent actuel. L'agent affiché est un mix de boutons et de label où les boutons représentent les primitives pouvant être exécutées. Cet agent est mis à jour à chaque exécution de la prochaine primitive.

FIGURE 3 – InteractiveAgentGUI.scala - Ecran Agent Interactif

3.1.4 Modification de bacht_cli.scala

Nous n'avons que très légèrement modifié le code source de l'interpréteur *bacht_cli.scala*. Ces modifications sont majoritairement des ajouts avec une ou deux modifications.

Expr

Le premier ajout que nous avons fait s'applique à l'objet *Expr* ainsi qu'à ses sous-types.
Nous avons déclarer deux fonctions utilitaires : une redéfinition de toString() et une fonction remove(expression) permettant de supprimer une primitive de l'expression.

```
1  class Expr {
2    def remove(expression:bacht_ast_primitive) = this
3  }
4  case class bacht_ast_empty_agent() extends Expr
5  case class bacht_ast_primitive(primitive: String, token: String) extends Expr {
6    override def toString:String = {
7      "[ " + primitive + "(" + token + ")" + " ]"
8    }
9
10   override def remove(expression:bacht_ast_primitive) = if (expression.primitive.equals(
         primitive) && expression.token.equals(token)) new bacht_ast_empty_agent
11                                                         else this
12 }
13 case class bacht_ast_agent(op: String, agenti: Expr, agentii: Expr) extends Expr{
14   override def toString:String = {
15     "[ " + agenti.toString + " " + op + " " + agentii.toString + " ]"
16   }
17
18   override def remove(expression:bacht_ast_primitive) = {
19     op match {
20       case ";" => if (agenti.remove(expression).isInstanceOf[bacht_ast_empty_agent]) agentii
21                   else bacht_ast_agent(op, agenti, agentii.remove(expression))
22       case _ =>     bacht_ast_agent(op, agenti.remove(expression), agentii.remove(expression))
23     }
24
25   }
26 }
```

BachTSimulParser

La deuxième modification apportée concerne l'étape de parsing. Nous avons modifié le "case failure :NoSuccess" pour que la fonction retourne null plutôt que de lancer une erreur.
Lors de l'appel de cette méthode une vérification sur la valeur de retour est donc nécessaire (et cruciale) pour afficher un message à l'utilisateur en cas de parsing incorrect.

```
1   def parse_primitive(prim: String) = parseAll(primitive,prim) match {
2         case Success(result, _) => result
3         case failure : NoSuccess => null
4   }
5
6   def parse_agent(ag: String) = parseAll(agent,ag) match {
7         case Success(result, _) => result
8         case failure:NoSuccess => null
9   }
```

BachTStore

Enfin, les dernières modifications concernent le store. Nous avons ajouter deux nouvelles méthodes utilitaires permettant respectivement d'avoir le contenu du tableau en une chaine de caractères et de connaître la densité d'un token particulier.

```
1   def getContent:String ={
2       var res = ""
3       for((t,d) <- theStore) res += t+"("+d.toString+")"
4       res = "{" + res + "}"
5       res
6   }
7
8   def findDensity(token:String):Int = {
9       try{
10          val density = theStore(token)
11          density
12      } catch{
13        case e:java.util.NoSuchElementException => 0
14      }
15  }
```

3.2 Deuxième Question

Pour cette deuxième question, il nous était demandé de concevoir et implémenter un langage permettant d'afficher des images selon le contenu du tableau. Ceci a été réalisé au moyen de la classe *VisualBlackboard.scala*, composée d'un ***GridPanel*** et d'un ***ScrollPane*** permettant l'affichage de plusieurs images.

La gestion de l'affichage des images s'effectue simplement, au moyen d'un listener sur le contenu du tableau. Ainsi, à chaque mise à jour de celui-ci (que ce soit l'ajout ou le retrait d'un token), l'affichage des images est recalculé. Dans cette version, nous avons choisit un affichage simple : un conteneur

(un bocal en l'occurence) qui se remplit en fonction du nombre de token sur le tableau. Les figures 4 et 5 présentent les différents cas possibles (à savoir les cas où 0,1,2,3,4 ou 5 tokens sont présents sur le tableau).
Pour des raisons de simplicités, nous avons traité les cas où la densité est supérieure ou égale à 5 comme similaire (partageant une même image).

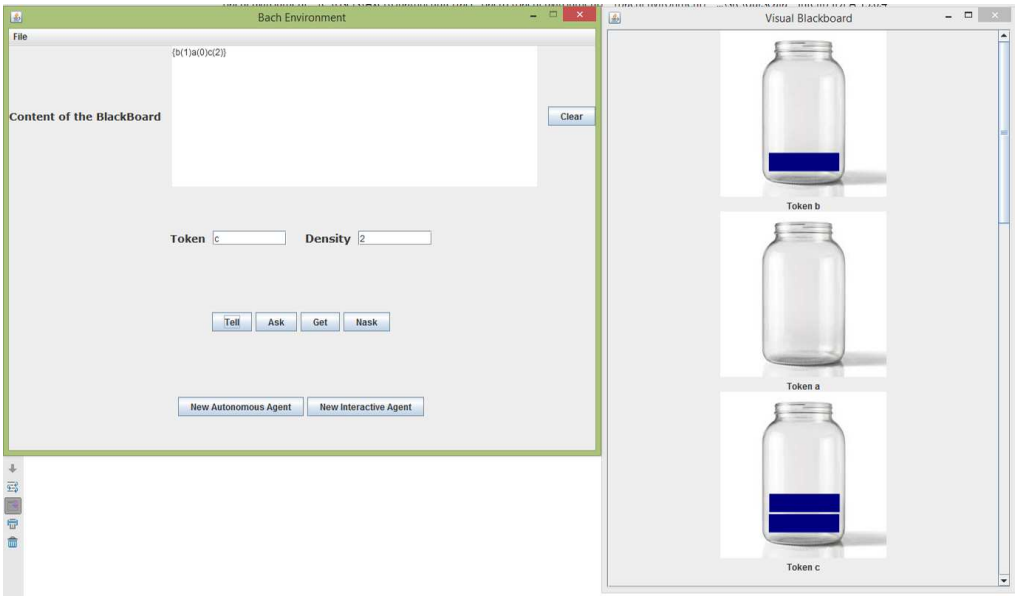

FIGURE 4 – VisualBlackboard.scala - 0 à 2 tokens

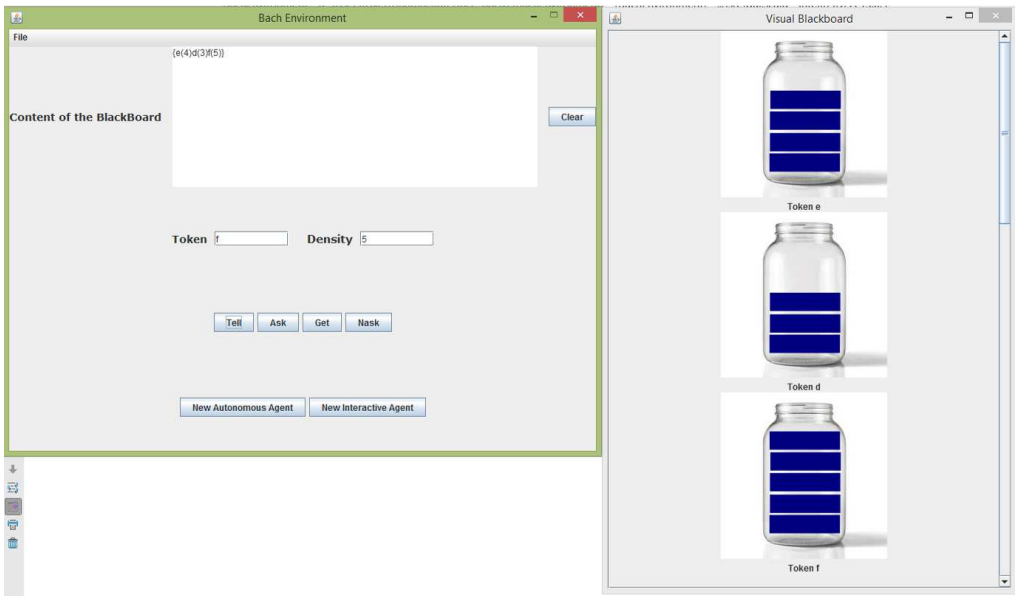

FIGURE 5 – VisualBlackboard.scala - 3 à 5 tokens